Publisher
Richard Bowles

Managing Editor
Stuart Douglas

Content Architect
Rahul Khanna
Mohan J Kumar

Program Manager
Stuart Douglas

Technical Editor
David Clark

Technical Illustrators
MPS Limited

Technical and Strategic Reviewers
Balint Fleischer
Rahul Khanna
Mohan J Kumar
Kshitij Doshi
Christian Le
John Ping
Mahesh Natu
Rafael De La Guardia
Kai Li
Mariette Awad
Patrick Yin. Chiang
Huaping Liu

Intel Technology Journal

ISBN 978-1-934053-48-5, ISSN 1535-864X

Intel Technology Journal
Volume 16, Issue 2

Publisher: Richard Bowles
Managing Editor: Stuart Douglas

Library of Congress Cataloging in Publication Data:

Printed in China
10 9 8 7 6 5 4 3 2 1

First printing: June 2012

INTEL® TECHNOLOGY JOURNAL
EXPLORING CONTROL AND AUTONOMIC COMPUTING

Articles

FOREWORD

Alan G. Ganek
Information Technology Consultant

IBM launched the Autonomic Computing (AC) initiative in October of 2001 as a call to action to address the massive complexity of information systems. The thesis was that the rate of advance of information technology was increasingly challenged by the complexity of deploying and operating useful information systems. The irony is that it is a problem born out of success: exponential increases in processor power, storage density, and communication speeds over the prior two decades had enabled incredible price performance improvements which, in turn, increased the appetite and demand for information technology. The concern was validated by numerous studies that indicated data center managers were typically spending over 70 percent of their budgets on maintenance and operations, severely limiting the ability to deploy new and better systems. The problem was simply stated as: ". . . the growing complexity of the I/T infrastructure threatens to undermine the very benefits information technology aims to provide"[1] and the response required was ". . . to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them."

At the time, the initiative was met with great fanfare from the press and industry analysts as well as major technology companies and academia. The technical community developed key architectural paradigms, open standards, and models of autonomic behavior for components and subcomponents as well as distributed systems as a whole. Nowadays the term *autonomic computing* is rarely heard in information technology conferences, press, and analyst reports. Clearly, it is no longer the industry buzz word of the day, which raises the questions: Was it just hype? A passing fad?

The answer is a resounding "*No!*" Continued focus and achievement in autonomic computing is alive and well in both the information technology industry as well as academia. The discipline of AC is a key, underlying technology in a wide range of industry initiatives, from hardware to software and from components to vast distributed systems. Perhaps the most visible example is cloud computing whose very viability depends upon self-managing technologies that enable the massive scale and dynamic operations inherent in deploying cloud computing. Sustained focus on self-management, higher levels of automation, and attacking complexity remains critical for the ongoing advancement of information technology.

I can think of no better demonstration of the ongoing health of autonomic computing than this issue of the *Intel Technology Journal.* The core concept of AC asserts that the key principles must be applied at every level of systems, from low level circuitry to end-to-end systems. As a leading provider of processing and communications technology, Intel shows in these articles AC at work across its broad spectrum of technology with impressive results. Applications of AC as diverse as fault diagnosis in circuitry, energy and power optimization, automated mechanical controls, chip design, wireless communications, and workload and resource management are presented. As a major force in the world of technology, Intel's continued commitment to this arena is welcome news for consumers of information technology, which really includes all of us.

Alan G. Ganek
Information Technology Consultant
Formerly:
Chief Technology Officer, IBM Software Group
Vice President, Autonomic Computing, IBM
May 2012

[1]Autonomic Computing Manifesto, IBM Research, October 2001

AUTONOMIC FOUNDATION FOR FAULT DIAGNOSIS

Contributors

Mahesh Natu
Data Center and Connected Systems Group, Intel Corporation

Narayan Ranganathan
Data Center and Connected Systems Group, Intel Corporation

Anil Agrawal
Data Center and Connected Systems Group, Intel Corporation

Increased integration, high speed interconnects, and new technologies like corrupt data containment are making fault diagnosis more challenging in Intel® Xeon® processor based server platforms. At the same time, users are expecting higher reliability, availability, and serviceability from Intel Xeon processor based systems. This article highlights some of these challenges in fault isolation and proposes an autonomic framework based on fault diagnosis capability of current and the future generations of the Intel server platforms.

Introduction

Autonomics is about components that manage themselves. The modern data center is a very complex, dynamic, and heterogeneous environment. Autonomic computing (AC) concepts have their roots in the biological systems and suggest a way to deal with such complexity. As a result, the concept of the autonomic data center generates lot of excitement in the industry. For example, automation is one of the three elements of Intel's Cloud 2015 Vision[1] and autonomics is a key component of automation. A true autonomic data center would be one that could operate itself such that it would meet stated business goals. Such a data center must translate business goals into an expected state and monitor the system state. If the observed state does not match the expected state, it generates and executes remediation plans as needed. Self-managing data centers reduce operational costs by minimizing human intervention. Designing a true autonomic data center is a very ambitious undertaking and must be decomposed into smaller problems. Generally, autonomic systems are said to have the following four attributes: self-configuration, self-optimization, self-protection, and self-healing. The self-healing aspect covers fault diagnosis and appropriate recovery actions. This article focuses on the fault diagnosis aspect and autonomic infrastructure built around fault diagnosis.

"Self-managing data centers reduce operational costs by minimizing human intervention."

A modern data center consists of thousands of compute nodes (servers) and storage units that are stitched together by a network fabric. These building blocks are also complex systems themselves. For example, a server system may contain multiple processors with several cores, gigabytes of memory, high performance I/O cards and many software components. The horizontal nature of the computer industry means that these hardware and software components are sourced from multiple vendors. It is virtually impossible to ensure that a system as complex as the data center will be error free. Much attention has been given to being able to predict failures[2,3], but even the best predictive failure analysis (PFA) cannot predict all failures in a modern data center. As a result, system reaction to a failure is very important. After an autonomic system

fails, it must be possible to correctly diagnose the fault and fault diagnosis must, therefore, be an essential part of data center automation. The sources of failures include hardware, software, and operator errors. Hardware errors can be attributed to design, production, environmental, or aging factors. Advance detection of errors when the system is operational can reduce the downtime by scheduling a preventive maintenance. While advance detection is important, it is also required that the analysis take into account the root cause (source) of the failure that may not be isolated during the observed warning or failure state.

"While advance detection is important, it is also required that the analysis take into account the root cause."

Fault diagnosis generally consists of the following three essential tasks[4]:

- Fault detection: Detection of the occurrence of faults in the functional units, which lead to undesired or intolerable behavior of the whole system
- Fault isolation: Localization of the fault to, say, a component. In some literature, the terms *fault isolation* and *fault diagnoses are* used interchangeably.
- Fault analysis or identification: Determination of the type, magnitude and cause of the fault. Determine whether the fault is transient or permanent. This phase also involves root cause.

Identification of a failing component allows timely recovery via replacement, either automated or manual. In autonomic computing parlance, this is an example of *self-healing*. Self-healing is the ability of a platform to effectively recover when a fault occurs. This self-healing can be either reactive or proactive. A reactive self-healing platform attempts to correct or isolate a fault once it has occurred. If a hard memory error can be isolated to a memory rank, a platform can map out the particular memory DIMM allowing the system to continue functioning or alert the data center administrator that the faulty DIMM needs to be replaced. Accurate fault diagnosis reduces the mean time needed for repair (MTTR) and thus increases system availability. The output of fault diagnosis can be utilized to drive changes to the design of the system or the failing component in a proactive manner.

"Self-healing is the ability of a platform to effectively recover when a fault occurs."

From an autonomic computing perspective, the modern data center calls for a hierarchical system model where the top level autonomic elements themselves are constructed from smaller autonomic elements and so on. A data center that is manageable relies on autonomic computing, server and networking building blocks that are able to perform fault isolation. These building blocks in turn require autonomic capabilities in their ingredients. Since Intel processors provide the brains for the majority of servers and storage units, Intel is embedding capabilities in these processors that improve fault isolation of these systems. This article covers autonomics embedded in Intel processors that aid in fault isolation.

Background

Autonomic computing provides a framework for self-configuration and self-healing. Fault diagnosis, specifically, in-field diagnosis, has received a lot of attention, such as, for example, Microsoft WHEA, APEI specification

"Higher core counts and a higher level of integration within the next generation of Intel processors continue to drive the need to build advanced autonomic fault diagnosis hooks."

as defined by ACPI industry standard[5], Oracle FMA, and various BIST techniques. This article attempts to connect autonomics techniques to fault diagnosis. Fault diagnosis capability is maturing and the second generation of Intel Xeon processors with integrated IBIST logic is being deployed now[6]. Higher core counts and a higher level of integration within the next generation of Intel processors continue to drive the need to build advanced autonomic fault diagnosis hooks. Such capabilities will help in improving the fault prediction capabilities and minimizing the time required for remote diagnostics.

Despite best efforts, hardware components do fail in the field. The objective of autonomic fault diagnosis is to establish the baseline (what is normal), monitor fault symptoms, detect errors, identify failed components, plan and execute service calls before an unplanned system failure occurs or to minimize service outage.

Figures 1 and 2 illustrate this concept using the autonomic computing framework at two levels. Figure 1 represents a data center (or a server pool within a data center) view. A data center, especially a cloud, can be abstracted as a provider of multiple services with mutually agreed upon service levels. As a result, data center level monitoring often boils down to measuring the service levels and determining if they meet the goals. If the service level falls below expectations, the data center automation software can examine server health logs and software error logs to determine the potential cause and work around it; possibly by finding an unused server to host the service. In order to manage complexity, data center level software needs to treat much of the individual node as a black box. In other words, more efficient management is possible if data center automation software can presuppose that the individual servers have autonomic capabilities such as the ability to isolate faults to an individual Field Replaceable Unit (FRU). Such a server is represented in Figure 2. The monitoring phase makes use of extensive error detection circuitry in the processor and other components. The section "Error Detection and Reporting" describes this circuitry in detail. The analysis phase examines the contents of error log registers and diagnoses the fault. Diagnosis includes determining whether the fault is permanent or not. The section "Challenges" goes over some of the challenging fault diagnosis scenarios. The planning and execution phases can attempt to work around permanent failure by various methods. If the server contains redundancy, the failing component can be mapped out on the fly. If the server does not contain redundancy, it may be possible to map out the component by rebooting in a degraded mode. If neither option is available, the system can be brought offline until the failing part is manually replaced by a technician.

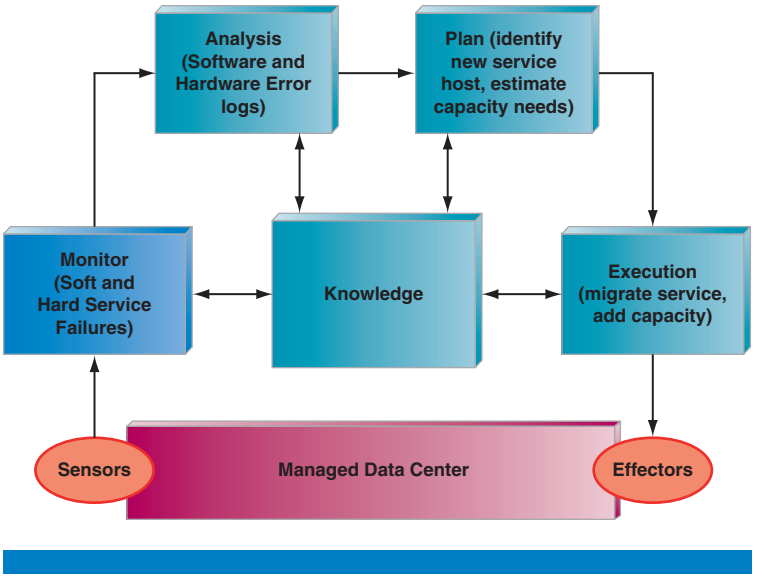


Figure 1: Autonomic foundation for fault diagnosis – data center context
(Source: Intel Corporation, 2012)

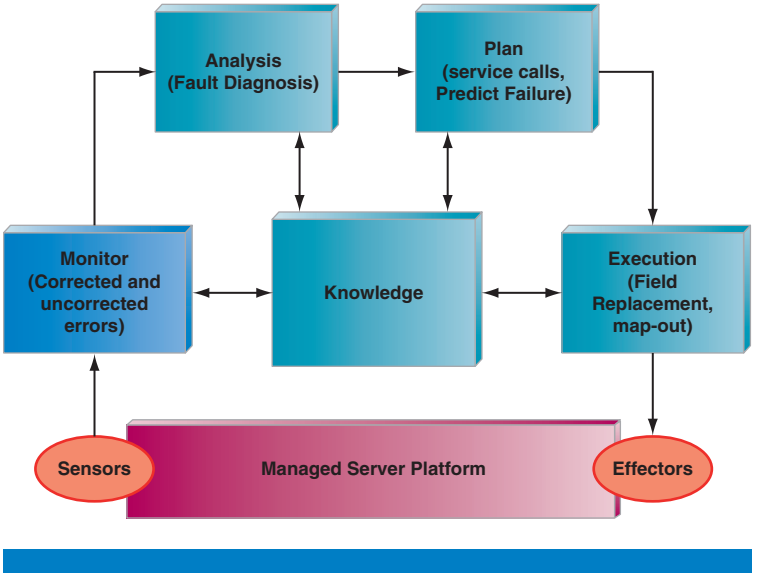


Figure 2: Autonomic foundation for fault diagnosis – server view
(Source: Intel Corporation, 2012)

Terminology

In order to expand further it is important to define a few key terms such as failure, fault, and error. We borrow this terminology from Salfner[2].

Failures are commonly defined as follows: a system failure occurs when the delivered service deviates from the specified service.

Failures are observable by the user, which can either be a human or another machine. Prior to observing a failure, things may go wrong inside the system, but as long as it does not result in corrupted output, there is no failure.

"A system failure occurs when the delivered service deviates from the specified service."

"Failures are observable by the user, which can either be a human or another machine."

"Once a fault has become visible it is called an error*."*

Faults on the other hand are the root cause of failures and are defined to be a defective (incorrect) state. Often faults remain undetected for some time. Once a fault has become visible it is called an *error*. Often errors are called a manifestation of faults. Figure 3 shows the relationship between faults, errors, and failures in the context of a managed server.

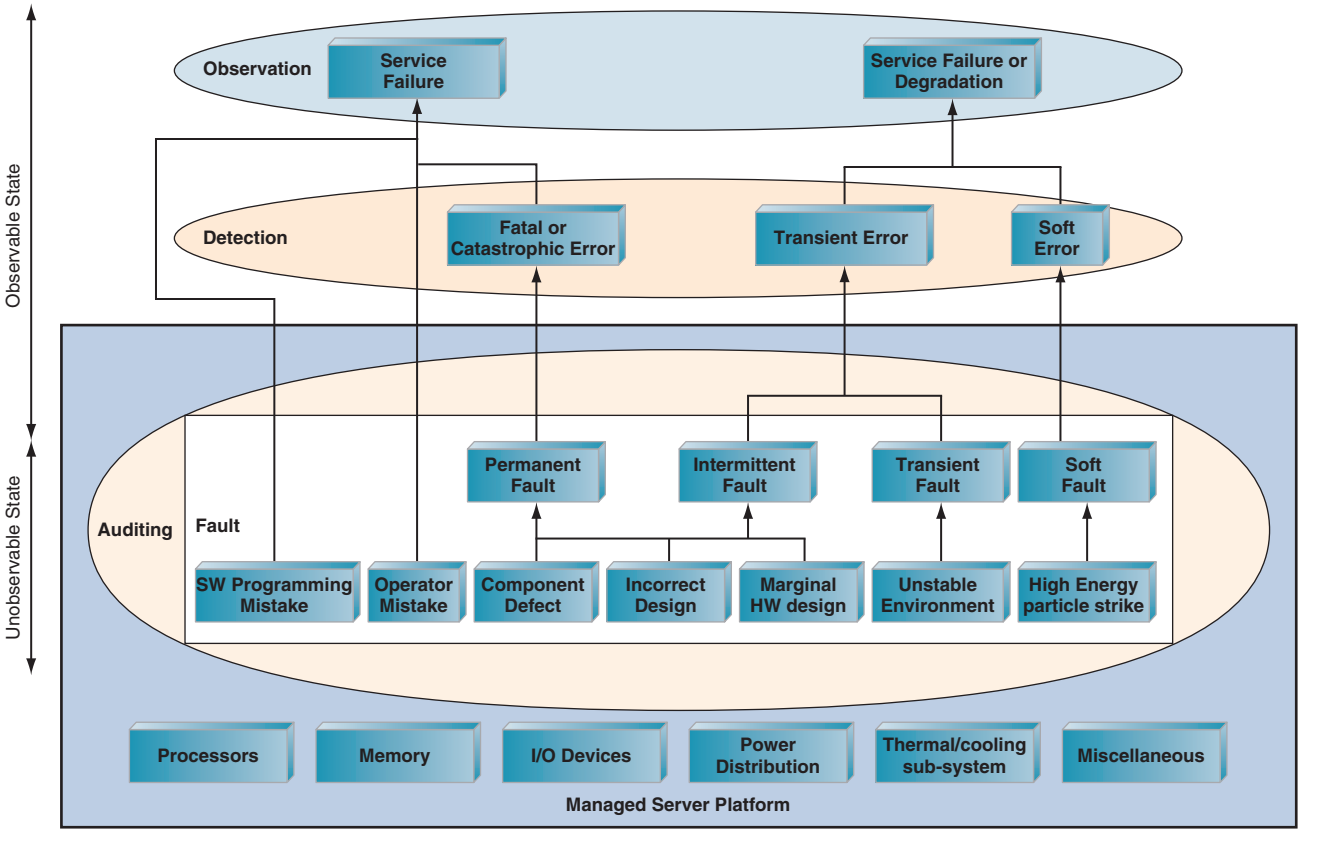


Figure 3: Relationship between fault, error, and failure (Source: Intel Corporation, 2012)

"Faults are unobserved defective states."

The key point is that faults are unobserved defective states. Faults can be made visible through any of the following three stages:

- Auditing (CRC, parity, checksums of data structures)
- Detection (various types of errors: corrected, recoverable, fatal) – Detection gets a lot closer to the source of a fault and is more precise in nature compared to symptoms.
- Monitoring symptoms – Symptoms, by their very nature, provide hints and are much less precise than detection. Monitoring symptoms becomes important when the ability to observe raw error sources is limited. Intel Xeon processors strive to provide firmware and software with good detection capabilities and thus minimize reliance on symptoms monitoring.

Such a design leads to better fault diagnosis. Due to this de-emphasis on monitoring, Figure 3 does not include a symptom monitoring block for clarity. However, there are error scenarios that are not handled by detection. For example, an ROB timeout (see the section "Instruction Retirement Watchdog Timeout") can be indicative of a faulty interconnect or faulty hardware or a software bug.

A good example for this is memory fault: consider a fault where one of the DRAM devices on a DIMM has failed. If read/write operations do not access that DRAM device content, the fault remains unobserved. Auditing would make it visible. Memory patrol scrubbing in Intel Xeon processors is an auditing technique. The memory patrol scrubbing engine walks through the entire memory in the background. Scrubbing would bring the device failure to light. In absence of ECC, such auditing would result in an uncorrected error and a system failure would occur. If features such as ECC and Single Device Data Correction are implemented (self-healing capabilities), the system can work around a manifestation of this fault. In all these cases, Intel Xeon processors detect and log the error. Once several DRAM devices have failed, and an attempt is made to read data from it, an uncorrected error is detected. In the case of "Independent memory channel" mode, this would lead to system failure. In case features like mirroring are implemented (called a self-configuring feature), then the desired service of data delivery can still be fulfilled and hence no failure occurs.

"Memory patrol scrubbing in Intel Xeon processors is an auditing technique."

Another interesting aspect of the precise definition of fault, error, and failure is that usually there is no one-to-one mapping among faults and errors: several faults may result in one single error or one fault may result in several errors. The same holds for errors and failures: some errors result in a failure and some errors do not. Even more complicated are cases where some errors only result in a failure under special conditions, and some faults may cause failures directly. Moreover, some faults remain inactive for the entire system lifetime.

"usually there is no one-to-one mapping among faults and errors."

For this reason, two distinct areas of research have evolved: root cause analysis and failure prediction. Having observed some misbehavior by one of the means shown in Figure 3, fault diagnosis tries to identify the fault that caused an error or failure, while failure prediction tries to assess the risk that the misbehavior will result in future failures. Fault isolation is similar to root cause in that it attempts to localize the source of failure to a specific component or module in the system.

Fault Isolation Capabilities of Intel® Xeon® Processor Based Platforms

Since the launch of Intel® Xeon® processor 5500 family products (code name Nehalem), Intel has been leading the industry in delivering efficient performance (per watt and per dollar) and scalable platforms. Besides higher core count and innovative power management, another dimension of Intel innovation is a drive towards higher level of integration (such as integrated

memory controller, graphic controller, and I/O module) within the same processor package. Besides the benefits of a higher level of integration, there is a challenge to overcome—a higher level of integration can lead to a higher probability of fault and reduced observability for platform debugging and fault isolation. Future process-shrink and operating voltage reduction will make the fault isolation even more challenging. In order to address this challenge, more sophisticated error handling capabilities will be required at the silicon level to improve the fault isolation capability, thus reducing the downtime and thereby improving the availability and serviceability. After successful launch of Nehalem product family, the Intel Xeon processor E7 product family was released with even more advanced RAS features addressing this challenge[7]. Intel has pursued a three-pronged strategy when it comes to fault isolation:

"Intel has pursued a three-pronged strategy when it comes to fault isolation."

- Enhancements in error detection and reporting – This covers machine check architecture, memory error reporting, and I/O error reporting. These capabilities allow software to capture detailed system state at the time of error. From this state, software can localize the fault to a component and attempt root cause analysis. This topic is covered under in the section "Error Detection and Reporting."
- Improvements in diagnostics capabilities – These include interconnect tests like IBIST. Software can trigger these on a failing system to get a more accurate diagnosis. The section "Interconnect Diagnostics" describes IBIST.
- Drive new standards that enable multiple software components to interact and share knowledge – Platform firmware and the operating system have visibility into different parts of the system. Platform firmware has a better understanding of the physical aspects (for example, types of DIMMs), whereas the operating system has more knowledge of the logical aspect (memory page allocation). The Firmware First model[5] enables a hierarchical error detection model that benefits the system. The section "Error Handling Software Models" goes over these.

This section first provides a review of the error classification, then describes the various error handling capabilities that exist in current generation of Intel Xeon processors targeted for compute infrastructure (such as cloud computing, high performance computing, and mission critical computing). It also draws a parallel between the Intel Xeon and Intel® Itanium® error handling architecture and highlights a few key challenges faced by the Intel Xeon processor architecture as Intel drives towards a higher level of integration with every generation of process shrink.

Error Classification

Errors can be classified within two broad categories: detected errors and undetected errors (Figure 4). Undetected errors are important because none of the error handling features would help in addressing such errors. Often undetected errors are classified as silent data corruption (SDC). One of the key objectives of a system designer is to minimize the SDC rate. As shown in Figure 4, not all undetected errors are critical; for instance, an undetected error within branch prediction logic may not impact the integrity of the computation.

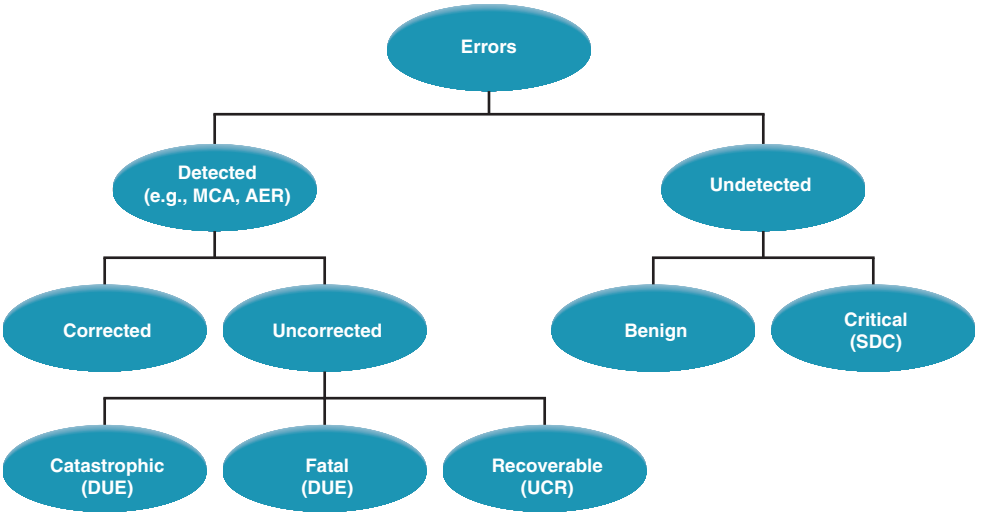


Figure 4: Error classification
(Source: Intel Corporation, 2012)

Once an error is detected, hardware will try to correct the errors, for example, memory single bit error (SBE) correction using ECC bits. This also includes the corrected errors that require firmware/software assistance, such as PCIe Link Layer Retry. In many cases, hardware/firmware/software may not be able to correct the errors; such errors are called as uncorrected errors (UCEs). Often uncorrected errors are classified as detected but uncorrected errors (DUEs). There are UCEs that may be recoverable with the help of system software and are classified as uncorrected recoverable errors (UCRs). The errors that are not software recoverable are called fatal errors since they prevent reliable system execution. Finally there are certain errors classified as catastrophic, where a system reset is required to bring system back to predictable state, such as processor internal errors (IERRs).

Error Detection and Reporting

This section covers the various error detection and reporting capabilities in Intel platforms.

Machine Check Architecture

Beginning with the Intel® Pentium Pro processor, Intel incorporated Machine Check Architecture (MCA) and has continued to enhance the MCA feature in subsequent processor families such as Intel Xeon and P6 family processors[18]. The Intel Itanium processor family innovated further and incorporated advanced Machine Check Architecture[9]. The MCA feature provides a mechanism for detecting and reporting hardware (machine) errors, such as: system bus errors, memory errors, parity errors, cache errors, and Translation Look-aside Buffer (TLB)errors. It consists of a set of model-specific registers

"The MCA feature provides a mechanism for detecting and reporting hardware (machine) errors."

(MSRs) that are used to set up machine checking and additional banks of MSRs used for recording errors that are detected. Figure 5 shows an example of MCA registers configuration.

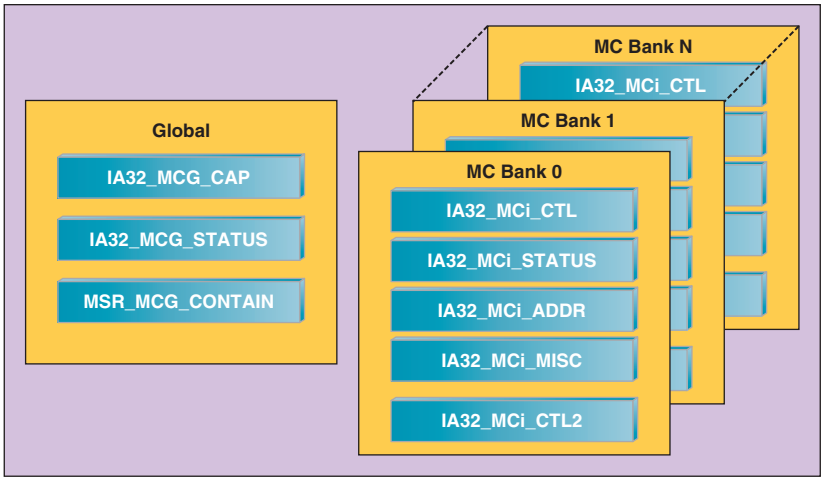


Figure 5: Machine Check Architecture error reporting registers (Source: Intel Corporation, 2012)

Once MCA is enabled, it will always log an error as soon as an error is detected. However, MCA will signal an error based upon the settings of the MCA Control register (IA32_MCi_CTL) and the types of error. Signaling of an error typically involves interrupts and assertion of an external pin.

The processor signals the detection of an uncorrected machine-check error by generating a machine-check exception (#MC), which is an abort class exception. The implementation of the MCA does not ordinarily permit the processor to be restarted reliably after generating a machine-check exception (MCE). However, the MCE handler can collect information about the machine-check error from the machine-check MSRs. Starting with the Intel Xeon processor 5500 family, the processor can report information on corrected machine-check errors and deliver a programmable interrupt for software to respond to MC errors, referred to as a corrected machine-check error interrupt (CMCI). Starting with the Intel Xeon processor E7 family, the processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected machine-check errors. The MCA handler flows within the Intel Xeon processor family differs from that in the Intel Itanium processor family and these are briefly described in the section "Error Handling Software Models."

"Starting with the Intel Xeon processor E7 family, the processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected machine-check errors."

Memory Error Reporting

Intel Xeon offers the additional capability of logging and signaling memory corrected errors for BIOS/firmware use. BIOS/firmware can program a threshold for the corrected error count. Once exceeded, the processor issues an interrupt to the BIOS/firmware allowing it to take appropriate action. Such threshold-based implementation is very simple and power efficient; however, it has its own limitation. Since typical servers operate 24/7 for many years, even in a highly reliable design, a certain level of corrected errors are unavoidable and will accumulate over time, thus triggering a "false alarm." In order to address this issue, the Intel Xeon processor has incorporated a more sophisticated algorithm known as the "leaky-bucket algorithm" (shown in Figure 6) where corrected errors are periodically decremented automatically by the processor. This leaky-bucket autonomic capability eases BIOS/firmware implementation of the memory failure prediction algorithm.

"Intel Xeon offers the additional capability of logging and signaling memory corrected errors for BIOS/firmware use."

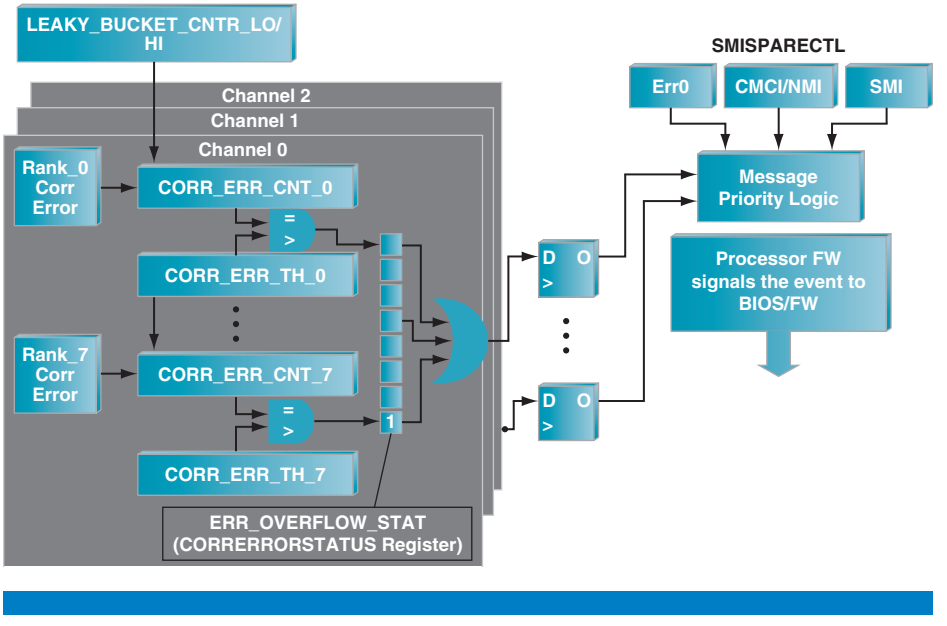


Figure 6: Memory-corrected error reporting using leaky-bucket algorithm (Source: Intel Corporation, 2012)

PCI Express* Advanced Error Reporting

The server platforms based on the Intel Xeon processor 5500 product family and Intel Xeon processor E7 product family used a dedicated chip called the IOH (Input Output Hub) for various input/output functions such as PCI Express* (PCIe) interconnects. This IOH chip incorporated advanced error reporting (AER) capability as per the PCI Express specifications. The IOH chip incorporated additional error logging capability above and beyond the errors defined in the PCI Express Specification. In addition to the PCI Express cluster, this chip also incorporated the Intel® QuickPath Interconnect (Intel® QPI) interface connecting

to the processor, cache structures for packet buffering and Intel® Virtualization Technology for Directed I/O (Intel® VT-d) functions. All these additional capabilities also required error detection and logging enhancements, which were also provided as part of the PCIe advanced error reporting. See Figure 7 for a

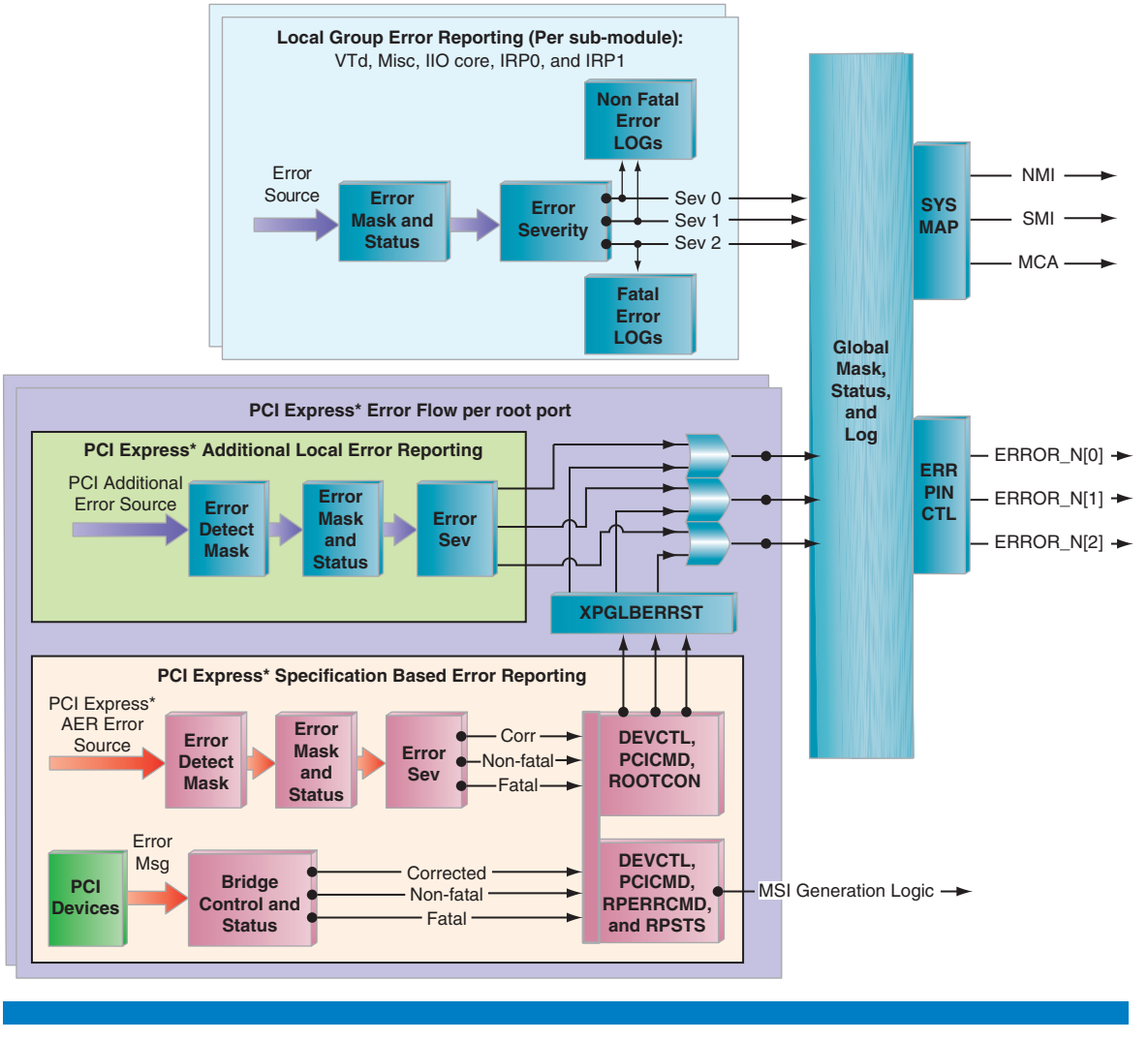


Figure 7: PCI Express AER and additional local error reporting (Source: Intel Corporation, 2012)

high level block diagram of the advanced error reporting. Following is a high level summary of IOH error detection and reporting capability:

- Detects and logs errors within coherency interfaces, PCI Express interfaces, ESI interface, IOH core logic, Intel VT-d, and miscellaneous.
- Provides the capability to mask error detection and reporting at the individual error level
- Local error reporting registers first log the status of corrected, nonfatal (recoverable), and fatal errors within these individual submodules.

- First and next error detection and logging for fatal and nonfatal errors. Additional header information is logged for the first reported error.
- Allows flexible mapping of the detected errors to different error severity
- Allows flexible mapping to various kinds of signaling: SMI, NMI, MCA, or ERROR_N[2:0]pins.
- Incorporates PCI Express specifications–based advanced error reporting with following key features:
    - Detects, logs, and signals errors received from the downstream devices. Signaling is done via Message Signaling Interrupt (MSI) at the local root port level. This is considered an architecturally defined error reporting mechanism and expected to be compatible with existing OS-based error handling software. Refer to Figure 8 for a list of errors detected. The processor is capable of reporting several additional internally generated errors.
    - Provides the capability to mask error detection, thus preventing further reporting to architecturally defined error handling software.
    - Allows flexible mapping of the detected uncorrected errors to different error severity (nonfatal and fatal).

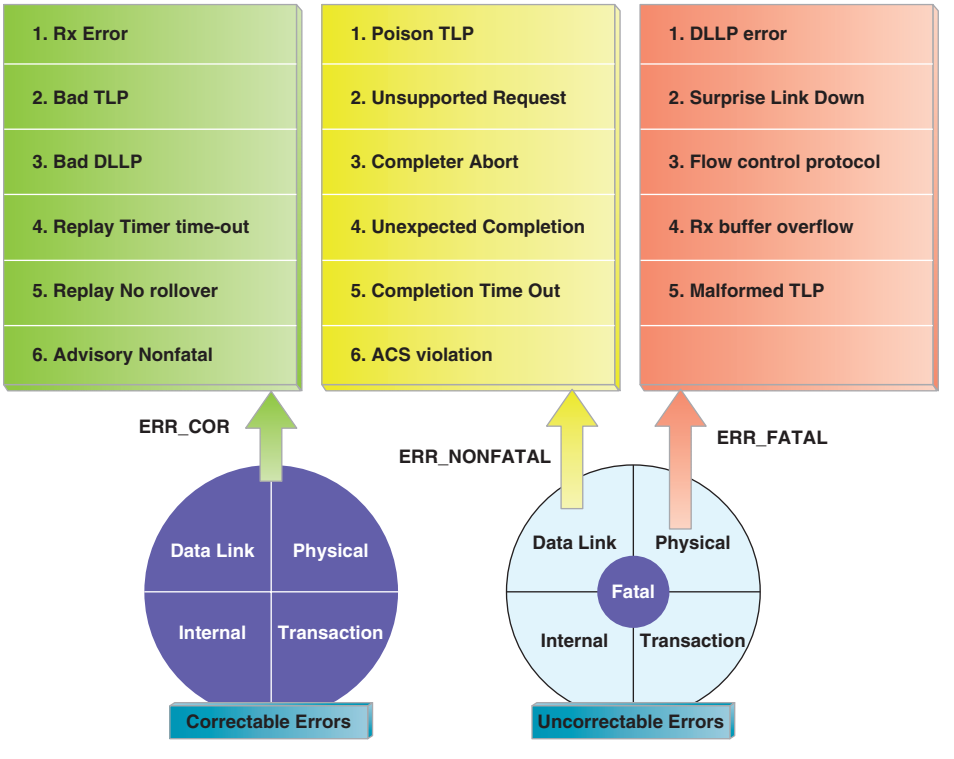


Figure 8: PCI Express specification–defined error types supported by Intel® Xeon® processors
(Source: Intel Corporation, 2012)

- AER error logging, signaling via SMI/NMI, subsequent error clearing, and rearming of the error logging logic attempts to accomplish following:
  - Coalesce multiple errors of the same type, severity, and from the same submodule to issue a single interrupt thus preventing multiple interrupts for same kind of error event
  - Prevent missing any interrupt due to an error from a different submodule, or of a different type and different severity.
  - Allows clearing of logged errors and simultaneously logging any new error thus preventing loss of any valid errors

In addition to implementing PCI Express specification defined error types, Intel Xeon platform IOH (Input Output Hub) incorporates logic to detect several additional errors types to improve the system serviceability. Figure 8 shows the PCI Express specification–based error types available in Intel Xeon platform IOH.

Interconnect Diagnostics

The industry is moving to faster interconnects. These naturally pose many challenges due to reduced electrical margins and they also lead to increased probability of interconnect-related faults. For example, faster interconnects are generally more susceptible to lane-to-lane crosstalk. At the same time, traditional methods for fault monitoring (such as test points) can no longer be used because they greatly perturb the interconnects to be monitored. Interconnect Built-In-Self-Test (IBIST)[6] enable detection of interconnect-related faults. At the highest level, IBIST is an on-die feature (integrated into Intel processors and chipsets) that enables chip-to-chip interconnect testing. It uses a finite state machine (FSM) to produce precise, deterministic, and arbitrary patterns on the I/O bus for testing purposes. It addresses both the static and high frequency fault spectrum associated with high performance bus topologies. When applied to PCIe, this means IBIST bypasses 8b10b encoding, stressing the "raw bus" margin without 8b10b encoding protection. In addition, pattern depth (120 bits) and width (2 to 4 lanes) are picked based on empirical link analysis and prior Bit Error Rate Testing measurement standards. The pattern richness is based on a prior known worst-case pattern and Bit Error Rate Testing methodology. For example, it is customary during early test chip and PCI Express specification development to run 2-4 lanes with different random and deterministic patterns. When applied to Intel QPI, IBIST supports protocol redundancy features that include failover clock, slow mode to fast mode, and so on. When applied to DDR, IBIST supports advanced DDR DIMM training patterns that allow the user to create and measure stress on address lanes.

"Traditional methods for fault monitoring (such as test points) can no longer be used because they greatly perturb the interconnects to be monitored."

"It is customary during early test chip and PCI Express specification development to run 2-4 lanes with different random and deterministic patterns."

As opposed to traditional diagnostic approaches using debug probe points, the on-line diagnostics approach provides a layered approach to access on-die features.

IBIST can greatly improve in-field diagnostics. Systems deployed at a customer location could fail due to interconnect-related faults, environmental stresses,

or silicon aging. Moving the failed system from the host environment to a service center may affect the reproducibility of the failure and may not be cost/time effective. In-situ debug enables more effective and rapid fault diagnosis isolation and allows execution of appropriate healing techniques to remedy the failure if feasible. An efficient fault isolation mechanism reduces the No Defect Found (NDF) conditions by lowering the field returns of expensive parts like processors and memory DIMMs. In addition, IBIST hooks in Intel silicon can be leveraged to proactively monitor interconnect health and configure tests remotely. This improves the availability of the system and creates an efficient serviceability environment. In case of mission-critical systems, emergency response can be immediate and failures can be prevented without delay.

"IBIST hooks in Intel silicon can be leveraged to proactively monitor interconnect health and configure tests remotely."

Error Handling Software Models

BIOS/firmware needs to be an integral piece of fault isolation because it carries knowledge about the specific hardware and has better visibility into the physical aspects of the server. On the other hand, the operating system manages allocation of compute resources to various applications and has better visibility and control into logical aspects of the server. Cooperation between these two entities leads to better fault diagnosis. As a result, Intel has been actively participating in definition of robust firmware–operating system interfaces[5,9]. Intel Xeon processor–based platforms and Intel Itanium processor–based platforms differ in their error handling firmware models. Intel Itanium processor–based platforms implement clearly defined hardware/firmware abstraction layers allowing a streamlined Firmware First Model (FFM) for error handling[8]. However, since Intel Xeon processor architecture has evolved over time it carries several legacy implementations that create challenges in implementing a streamlined FFM model for error handling. This section first briefly describes the Intel Itanium processor family firmware model and then draws a parallel with that of the Intel Xeon processor family firmware model. We also highlight a few key challenges in developing a robust FFM model.

"Intel Xeon processor–based platforms and Intel Itanium processor–based platforms differ in their error handling firmware models."

Intel® Itanium® Processor Family Firmware Model

The Itanium architecture defines three firmware layers: the Processor Abstraction Layer (PAL), the System Abstraction Layer (SAL), and the Unified Extensible Firmware Interface (UEFI) layer.

- PAL encapsulates processor functions that are likely to change between processor implementations so that SAL firmware and operating system software can maintain a consistent interface to the processor. These include non-performance critical functions such as processor initialization, configuration, and error handling.
- SAL is the platform-specific firmware component provided by OEMs and firmware vendors. SAL provides runtime services to the OS and provides a consistent implementation-independent interface to the operating system.
- UEFI is the firmware layer that provides a legacy-free API interface to the OS loader for boot and runtime services. SAL and UEFI isolate the OS and other higher level software from implementation differences in the platform.

PAL, SAL, and the OS work together to handle machine check aborts, processor corrected errors, and platform-corrected errors. Figure 9 provides an overview of how the firmware and OS interact for machine check handling.

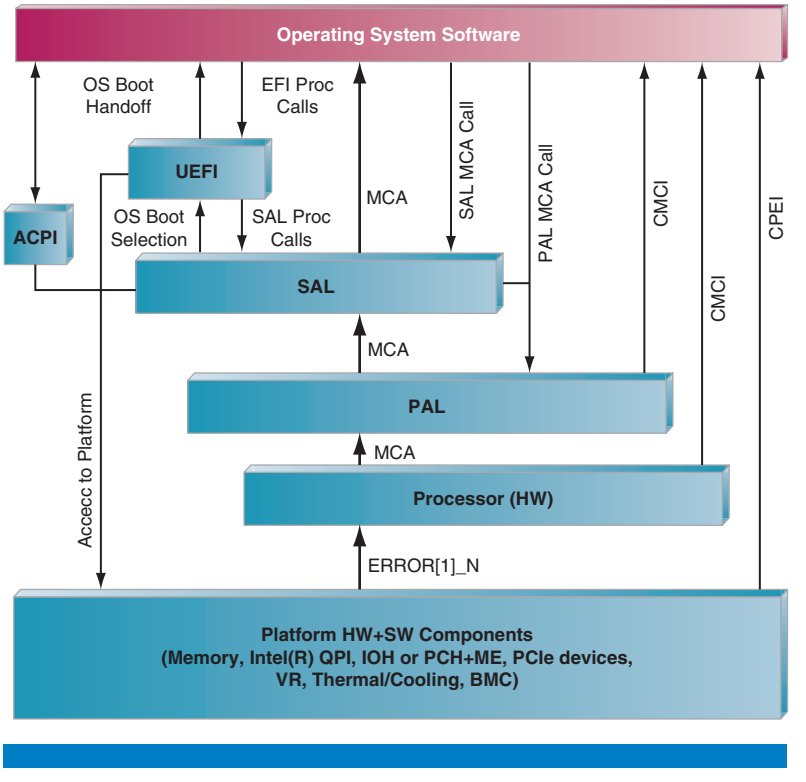


Figure 9: Intel® Itanium® processor family error handling firmware model (Source: Intel Corporation, 2012)

Intel® Xeon® Processor Family Firmware Model

The Intel Xeon processor product family error handling firmware model is shown in Figure 10. The processor firmware layer essentially includes the built-in firmware that provides an interface for BIOS/firmware to access various MSRs (Model Specific Registers) and CSR (Configuration Specific Register). As shown in Figure 10, one key differentiating aspect of the Intel Xeon processor error reporting architecture is the parallel logging and signaling of corrected and uncorrected errors to BIOS/firmware and OS. While this approach fosters industry innovation and retains compatibility with legacy software, it also causes an issue when both OS and BIOS/firmware attempt to develop similar diagnostics features accessing the same information from the processor, such as predictive failure analysis using memory-corrected error logs. In order to build a robust error handling model, it is more appropriate to develop a streamlined Firmware First Model. Intel Xeon processor E7 product family takes a step in this direction by implementing logic where MCE signaling would always result in, first, issuing a System Management Interrupt (SMI) to the BIOS/firmware, thus allowing it to process error logs before subsequently allowing an OS-based MCA handler to handle the errors that

"One key differentiating aspect of the Intel Xeon processor error reporting architecture is the parallel logging and signaling of corrected and uncorrected errors to BIOS/firmware and OS."

BIOS/firmware was unable to handle. A similar approach for corrected errors would also be desirable in the Intel Xeon processor families.

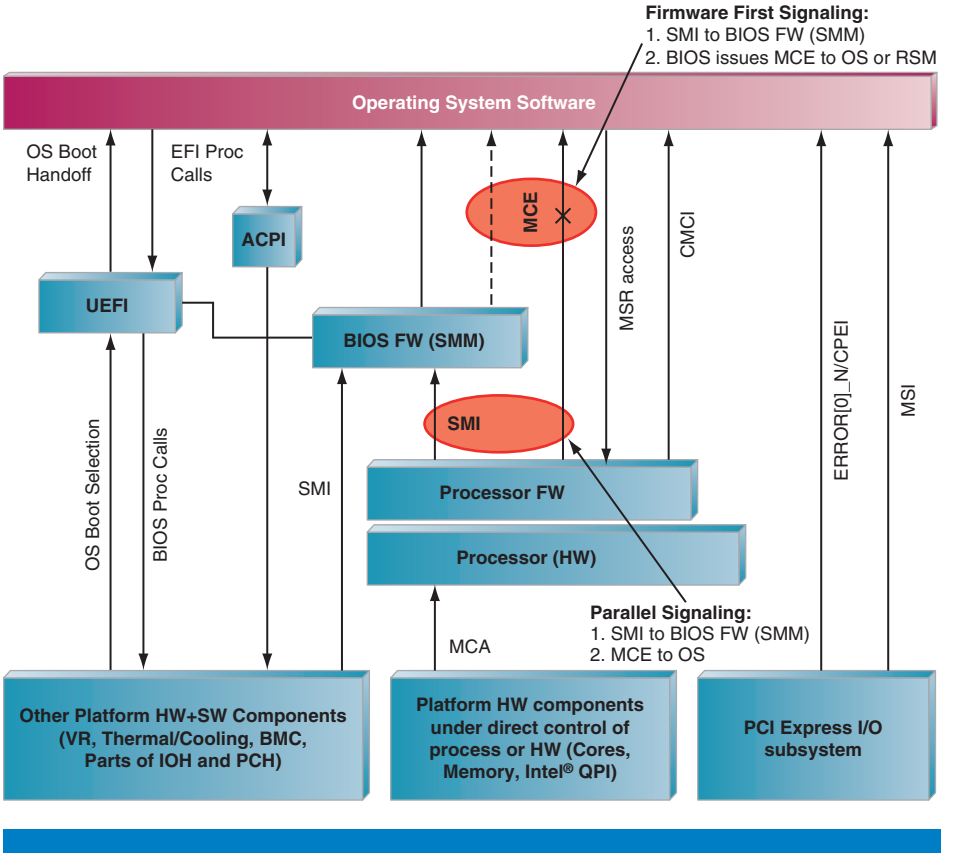


Figure 10: Intel®Xeon® processor family Error Handling Firmware Model (Source: Intel Corporation, 2012)

Challenges

Despite the extensive infrastructure that is described in previous sections, there remain certain challenges in the area of fault diagnosis. This section describes two such scenarios.

Instruction Retirement Watchdog Timeout

Processor instruction retirement watchdog timeout (also known as three-strike timeout) is one example of a failure scenario that is notoriously hard to diagnose[10]. From the system standpoint, even though the processor generally supports out-of-order execution, instructions are retired in order to ensure correctness of program execution. If the processor is unable to retire an instruction in 10–15 seconds, the processor determines that it is not able to make forward progress, asserts the CATERR pin, and stops execution.

Fault isolation for this error case is challenging for several reasons:

- Timeout is a *symptom* (see the "Terminology" section) and therefore less precise than detected errors. As with any symptoms, there is a chance that it

may not correspond to a real error. A large timeout value must be chosen so as to minimize false positives.

- Generally speaking, the time elapsed between a transaction failure and the corresponding error signaling is equal to the timeout value. Fault isolation involves walking backwards from the signaling point to the point of fault. In many cases, root cause analysis must consult the context prior to the fault; that is, capture transactions prior to the failing instruction. Since it is not possible to predict which instruction may timeout, proper analysis would require the system to keep a running log of transactions for the duration of the timeout. Since the processors and interconnect operate at GHz frequencies, it is generally not possible to keep such a running log. Moreover, such analysis might require visibility into the state of other internal structures (like internal queues) around the time of failure, which can be both challenging and difficult, if not impossible, to reconstruct at a later time.

"Empirical evidence shows that the source of the error is often outside the processor even though the processor is the one reporting the error."

- Since the point of fault and signaling could be far apart in time, it is difficult to sort out component interdependencies. Empirical evidence shows that the source of the error is often outside the processor even though the processor is the one reporting the error. For example, it has been observed that an I/O card hang can manifest itself as a retirement timeout. When an I/O card hangs, the CPU instruction that accesses it (for example a load operation or I/O port read/write) will not complete and it triggers a retirement watchdog timeout. It is possible that the I/O card hang could be either the result of a hard error in the I/O card or buggy software that placed the card in a bad state or something else. This problem is mitigated by implementing a hierarchical timeout scheme that provides better insight at component boundaries.
- Capturing system state at the signaling point itself may be challenging. The timeout is an unrecoverable error condition and may cause system pathways to be blocked. As a result, the error registers may either be inaccessible or the system may not be able to guarantee reliable access to the error log registers. This issue is often addressed by providing dedicated, sideband pathways to the error registers. If an I/O card hang is responsible for the failure, the error detection logic on the I/O card itself is also affected and may not log any event. A system reset can reopen sensor access, but a reset may destroy part of the system state and thus hamper fault diagnosis.

"Fault isolation and determining root cause often requires intimate knowledge of processor internals."

- Intel processors and chipsets contain various hooks that can be used for identifying faulty components in these cases. However, fault isolation and determining root cause often requires intimate knowledge of processor internals. These internal details often change from one generation of the processor to another. In many cases, Intel struggles with making such internal details of the chip available to external parties, even under nondisclosure agreement (NDA). In some cases, Intel customers may not have any interest and/or may not have the resources necessary to process these details. In addition, exposing raw data without any analysis and abstraction does not align well with the principles of an autonomic

computing system. An autonomic computing system is expected to perform and manage itself as much as possible and hide complexity from an external agent. Khanna et al. propose Diagnostics Hardware Abstraction[6] (DHA) as a solution. When applied to processors and chipsets, DHA can provide a generic set of APIs that provide platform manufacturers access to fault diagnosis primitives without having to worry about internal implementation details like registers. DHA enables portability across platforms and architectures and more accurate fault isolation. One example of a DHA API could be a function that examines the internal state of the processors upon an instruction retirement watchdog timeout and identifies the most likely source of the fault.

Data Poison Forwarding

Data poisoning (known as Corrupt Data Containment) is a method for synchronously signaling an uncorrectable error with the data from a source of data to its destination. The data poisoning technique is commonly used in mainframes and high-end servers[8]. It is now finding its way into x86 processors. Data poisoning generally provides two benefits[11]: (1) reduction in the number of false DUEs and (2) better error localization. Typically, the data poison indication is a bit per a certain granularity of data. The data granularity is implementation dependent. The source of the data, upon discovering an uncorrectable error on that data, would set the poison bit and this poison indication would travel synchronously with the data to its destination (the requester of that data). Upon receiving poisoned data, the onus of what to do is on the final consumer (receiver) of that data. As an example, the core, doing a load operation, is the final consumer of that data. The consumer has three options:drop the data and signal an error, drop the data without signaling an error, or consume the data without signaling an error. As an example, a core that gets poisoned data on a demand read should signal the error, whereas for a prefetch read, it may drop the data and not signal an error with no immediate consequences. As another example, a graphics device that gets poisoned data may consume the poisoned data since an error on a single pixel does not have to bring the whole system down.

"Data poisoning (known as Corrupt Data Containment) is a method for synchronously signaling an uncorrectable error with the data from a source of data to its destination."

When data poisoning is enabled, the producer of the data does not signal an uncorrectable error. It signals a lower severity, lower priority error and relies on the consumer to signal a high priority/high severity error. The poisoned packet may travel through one or more intermediate agents before it reaches the consumer. For example, the packet may travel over Intel QPI or may be temporarily stored in cache hierarchy. These intermediate agents are able to observe the poison bit but will not report a high severity error.

"When data poisoning is enabled, the producer of the data does not signal an uncorrectable error."

Even though data poisoning reduces false DUEs, it introduces challenges in implementing strong fault containment. Poisoned data can cross component boundaries and the detection may be deferred until it is consumed. Addition of sensors at key component boundaries can provide the missing data pieces, but connecting these pieces for fault diagnosis purposes remains challenging because it requires a deep understanding of how transactions flow inside of the processor.

For example, the entity performing the fault diagnosis may need to understand the complex caching policies and performance enhancements such as prefetching. A robust framework such as DHA is expected to address this challenge.

Future

Increased computing demand is driving the complexity of computing systems used in modern data centers. From the time these sophisticated computing systems are powered on, it is expected that these systems would operate 24/7 for several years. During the whole lifespan, the state of the computing system changes dynamically. Classical reliability theory and conventional methods rarely consider the actual state of a system and are therefore not capable of reflecting the dynamics of runtime systems and failure processes. The distinction between "healthy" and "broken" is often indistinct and fuzzy, and often there is a gradual transition between these two states; a system often does not break down recognizably but deteriorates over time. Thus we can say there is a fuzzy zone, a degraded state, separating acceptable and unacceptable behavior of a system, which again depends on user preferences and environmental changes. To allow for the dynamic properties of modern computer systems, online failure prediction incorporates measurements of actual system parameters during runtime in order to assess the probability of failure occurrence in the near future in terms of seconds or minutes. Simultaneously, modern systems are often designed to be fault tolerant and include hooks to facilitate both manual and automatic reconfiguration and repair from events that cause the system to violate its requirements and functionality. Going forward, it is our intent to pursue areas like the ones outlined below to improve overall fault diagnosis, system reliability, and health.

"Classical reliability theory and conventional methods rarely consider the actual state of a system and are therefore not capable of reflecting the dynamics of runtime systems and failure processes."

Improved Fault Classification

Successful reconfiguration or repair largely depends on accurate fault diagnosis; that is, correctly identifying the modules exhibiting the observed erroneous behavior. Many algorithms, including those discussed in earlier sections of this article, exist for autonomic diagnosis. More importantly, merely identifying the hardware modules affected by or exhibiting the faulty behavior is not sufficient and it is desirable to further classify the faulty behavior as either permanent or transient or induced from another module. Further, the fault treatment for these cases often needs to be different. For example, with a transient fault, where a module might only momentarily be prone to behaving erroneously, one might choose to deal with it by allowing future use of the affected module after recovering any data error caused by the transient fault, but not so for a permanent fault. It is not uncommon for modules to be replaced as faulty but later proven to be free from permanent faults, when tested in the repair shop. Many vendors including processor vendors and DIMM vendors report that they cannot find any defects in the returned parts.

"It is not uncommon for modules to be replaced as faulty but later proven to be free from permanent faults."

Treating transient faults as permanent can, thus, have a high cost. The cost of unnecessary component replacement includes the cost of the component itself, the labor cost, and indirect costs resulting from system downtime.

Customers are often not aware of these hidden costs and insist on replacing any component that has exhibited an uncorrectable error.

Previous work in this space[12] suggests that in many computer systems, transient faults are often the cause of errors in a great majority of cases. Published measures of the ratio between the frequencies of transient and permanent faults can vary from 4 to 1000. However, discriminating between transient and permanent faults is often difficult. Designers have used several techniques, spanning from simple retry to *thresholding*. They count errors, and when the count crosses a preset threshold a permanent fault is assumed[13,14]. For channel errors, a retry is generally effective. Pizza et al. have proposed a procedure based on Bayesian inference and takes into account factors like test coverage and rate of occurrence of faults to achieve a more optimal fault classification[15]. More work is needed in this area before customers relax the current requirement to swap out any component with an uncorrected error.

"Published measures of the ratio between the frequencies of transient and permanent faults can vary from 4 to 1000."

Efficient Distributed Diagnosis

Probing is a mechanism that is commonly used to get information from the system in order to monitor its health. In order to achieve better system health monitoring and improved diagnosis of the system on-line, it is necessary to improve the observability of the system by deploying more and specialized probes, sensors, and diagnostic agents on system components. Given the highly interdependent nature of system components, it is extremely important to make these diagnosing agents work together with consistency and efficiency, or efficient distributed diagnosis. The general idea is to use sensor readings together with mathematical models of the system to predict the health of the system and generate real-time actionable information.

Optimal Placement of Sensors

While it is generally desirable to have as many observation points as possible, there is usually an upper limit imposed due to impact on system cost, functionality, and performance. Hence, it is important to have an optimal placement of the available sensors and agents in order to maximize the ability to detect a failure as close to the error source as possible. One needs to identify what the key points in the system are for adding these sensors. This subject of optimal sensor placement (OSP) has been addressed extensively in literature for aerospace structures, process control industry, nuclear power plants, and physical infrastructure like bridges. It might be instructive to study the above in order to learn what might apply to computer systems. For example, as mentioned in the earlier poison forwarding discussion, optimally placing poison sensors at key component boundaries can help with better poison fault isolation.

"While it is generally desirable to have as many observation points as possible, there is usually an upper limit imposed due to impact on system cost, functionality, and performance."

Better Error Correlation and Analysis through Modeling Techniques

System components are largely interdependent. Hence, we need to employ modeling techniques that can represent and identify these interdependencies while isolating the faulty component. The predictability model should incorporate all components irrespective of the health coverage in order to predict the fault sequence and transitions. This can be done by creating the

model along with a profile that correlates the states based on observation points from covered components. Training then hardens the profile and evaluates missed and correct predictions. Many different models are possible and have been explored. Patterson-Hine et al. propose a coarse-grain, graphical dependency model in the context of an aviation subsystem[16]. In that case, the interfaces and dependencies between subsystems and their components were determined and modeled using multisignal flow graphs. The multisignal modeling methodology is a hierarchical modeling methodology where the propagation paths of the effects of a failure are captured in terms of a directed graph. Propagation algorithms convert this graph to a single global fault dictionary for a given mode and state of the system. This dictionary contains the basic information needed to interpret test results and diagnose failures reported by the monitoring system. Multisignal modeling allows the modeler to hierarchically describe the structure of a system and then specify its functional attributes via signals. This is ideally suited for building accurate low-cost models that can be used by a reasoner in real-time to interpret test results and assess system health. Khanna et al. proposed use of models like HMM (Hidden Markov Model) to correlate the system test-point observations, mortality characteristics, and state transitions to predict the most probable hidden fault state sequence[17]. Such a modeling scheme consists of creating the HMM components comprised of observed states, hidden states, and HMM profiles. Observed states are created using the RAS indicators or observation points (for example, BIT Errors per PCI Express transactions). Hidden states are created by identifying the clusters of failure-prone components that can be inferred by the observed state probabilities. An HMM profile that consists of transition probabilities and observation symbol probabilities is created by training using initial data and re-estimation with the system usage on multiple systems. HMM is found to be more sensitive to change detection than pure discriminative methods and also increases the quality of the model by constantly updating the temporal correlations. Key steps involved in HMM modeling are:

"Multisignal modeling allows the modeler to hierarchically describe the structure of a system and then specify its functional attributes via signals."

"HMM is found to be more sensitive to change detection than pure discriminative methods."

- Creating observed states that are analytically or logically derived from the RAS indicators. These RAS indicators are test-points spread all over the system. For example, there are BER sensors for interconnects, thermal sensors for sockets, voltage sensors, and so on.
- Creating hidden states by clustering the homogeneous behavior of single or multiple components together. These components are comprised of compute nodes, I/O nodes, memory devices connected by interconnects, power rails, switches, and so on.
- Creating a hidden state transition probability matrix using prior knowledge or random data. This prior knowledge combined with long term temporal characteristics form an approximate probability of failed components transitioning from one failure state to another in the same component or a different component.
- Creating an instantaneous observation probability matrix that indicates the probability of an observation, for a given hidden state.

References

[1] Intel Corporation, "Intel Cloud 2015 Vision."

[2] Felix Salfner, "Event-based Failure Prediction: An Extended Hidden Markov Model Approach", Dissertation, 2008.

[3] Felix Salfner, Steffen Tschirpke. "Error Log Processing for Accurate Failure Prediction", USENIX Workshop on Analysis of System Logs.

[4] S. X. Ding, *Model-based Fault Diagnosis Techniques Design Schemes, Algorithms, and Tools*, Springer Publications, 2008.

[5] ACPI Specification 5.0. www.acpi.info

[6] Rahul Khanna and Mohan J. Kumar, *A Vision for Platform Autonomy: Robust Frameworks for Systems*, Intel Press, 2011.

[7] Intel Corporation, "Xeon® E7 Processor - RAS Servers White Paper"

[8] N. Quach, "High availability and reliability in the Itanium processor," IEEE Micro, vol. 20, no. 5, pp. 61–69, 2000.

[9] Distribute Management Task Force, "System Management BIOS Specification."

[10] Intel Corporation, "Processor Reorder Buffer (ROB) Time out Debug Guide", October 2010.

[11] Sridharan et al., "A Taxonomy to Enable Error Recovery and Correction in Software," Workshop on Quality-Aware Design (W-QUAD).

[12] D. P. Siewiorek and R. S. Schwartz, *Reliable Computer Systems Design and Evaluation,* Bedford, MA, Digital Press, 1992.

[13] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and F. Grandoni, "Discriminating Fault Rate and Persistency to Improve Fault Treatment", in Proc. FTCS-27, Seattle, USA, 1997, pp. 354–362.

[14] L. Spainhower, J. Isenberg, R. Chillarege, and J. Berding, "Design for Fault-Tolerance in System ES/9000 Model 900", in Proc. FTCS-22, Boston, Massachusetts, 1992, pp. 38–47.

[15] Pizza et al., "Optimal Discrimination between Transient and Permanent Faults," High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International, Nov 1998

[16] Patterson-Hine et al., "A Model-based Health Monitoring and Diagnostic System for the UH-60 Helicopter."

[17] Khanna et al., "Control Theoretic Approach to Platform Optimization using HMM", InTech, 04/2011

[18] Intel Corporation, "Intel® 64 and IA-32 Architectures Software Developer Manuals"

Mahesh Natu is a platform software architect in the Data Center and Connected Systems Group at Intel Corporation. He has been with Intel Corporation for 16 years. He can be reached at mahesh.natu@intel.com

Narayan Ranganathan is a platform software architect in the Data Center and Connected Systems Group at Intel Corporation. He has been with Intel Corporation for 13 years. He can be reached at narayan.ranganathan@intel.com

Anil Agrawal is a senior platform applications engineer in the Data Center and Connected Systems Group at Intel Corporation. He has been with Intel Corporation for 15 years. He can be reached at anil.agrawal@intel.com.

COORDINATED OPTIMIZATION: DYNAMIC ENERGY ALLOCATION IN ENTERPRISE WORKLOAD

Contributors

Martin Dimitrov
Software and Services Group,
Intel Corporation

Kshitij Doshi
Software and Services Group,
Intel Corporation

Rahul Khanna
Software and Services Group,
Intel Corporation

Karthik Kumar
Software and Services Group,
Intel Corporation

Christian Le
Intel Architecture Group,
Intel Corporation

"This article describes a process of observation, modeling, and course corrections that is successful in achieving autonomic power control in an Intel® Xeon®E5-2600 server machine."

Controlling how much power server machines draw has become increasingly important in recent years. The accuracy and agility of three types of actions are critical in power governance: (1) selecting which hardware elements must run at what rates to meet performance needs of software, (2) assessing how much power must be expended to achieve those rates, and (3) adjusting the power outlay in response to shifts in computing demand. Observing how variations in a workload affect the power drawn by different server components provides data critical for analysis and for building models relating quality of service expectations to power consumption. This article describes a process of observation, modeling, and course corrections that is successful in achieving autonomic power control in an Intel® Xeon®E5-2600 server machine meeting varying response time and throughput demands during the execution of a database query workload. The process we describe in the article starts with fine-grained power-performance observations permitted by a distributed set of physical and logical sensors in the system. These observations are used to train models for various phases of the workload, with accuracy between 97 and 98.5 percent. Once trained, system power, throughput, and latency models participate in optimization heuristics that redistribute the power to maximize the overall performance/watt of the server.

Introduction

It has become vital to sharply curtail the power that servers consume during periods of low utilization. The volume of information that must be processed in real time has been growing geometrically[18] over the past few years, requiring peak processing capabilities to rise in concert. Despite superior performance per watt that newer platforms deliver, handling peak loads continues to require higher power delivery and heat dissipation capacities per cubic meter in enterprise IT and datacenter facilities, with 63 percent of the total cost of ownership going towards powering, cooling, and electricity delivery infrastructure[15]. In contrast to the traditional focus on delivering the highest throughput or lowest response time unconstrained by power, these realities have made it a more compelling proposition to minimize the amount of energy consumed in relation to computational work performed while meeting responsiveness targets. In particular, dynamically conserving power when some machines do not need to be at full utilization translates directly into cost savings and creates greater allowance for other, more power-constrained, servers.

We use the term *power optimization* to describe the act of targeting and achieving high levels of power normalized performance at the application level.

For a software application, such as a business transactions service or a content retrieval service, the performance metrics that are significant are the numbers of requests serviced (throughput) and the turn-around delay (response time) per request. The unqualified use of the term *performance* in this article will be about these application level qualities-of-service metrics.

Optimizing power entails multiple dynamic tradeoffs. Typically, a system can be represented as a set of components whose cooperative interaction produces useful work. These components may be heterogeneous or be presented with heterogeneous loads, and vary in their power consumption and power control mechanisms. At the level of any component—such as a processing unit or a storage unit, power needs to be increased or decreased on an ongoing basis according to whether that component's speed plays a critical role in the overall speed or rate of execution of programs. In particular, different application phases may have different sensitivity to component speeds. For example, a memory bound execution phase will be less impacted by CPU frequency scaling than a CPU bound execution phase. Under execution reordering that most modern processors employ, the degree to which a program benefits from out-of-order execution varies from one phase to another. Moreover, the rate at which new work arrives in a system changes, and as a result, the overall speed at which programs have to execute so that they can meet a given service level expectation varies with time. Thus the power-performance tradeoffs that are needed have to occur on a continuous basis.

"Power needs to be increased or decreased on an ongoing basis according to whether that component's speed plays a critical role in the overall speed or rate of execution of programs."

Arguably, given the self-correcting and self-regulating aspects common in most systems today, software driven power-performance should be unnecessary. For example, Rotem et al.[26] present power control algorithms that transition CPUs and DRAMs into lower frequencies or into ultra-low power modes during low activity periods. While the circuit level self-regulation is highly beneficial in transitioning components to low power states, software needs to wield policy control over when and which activity should be reduced in order to facilitate transition of hardware into power saving modes, as discussed next.

Harnessing power savings on less busy servers is a delicate task that is hard to delegate to hardware based recipes. Servers are typically configured for handling high rates of incoming work requests at lowest possible latencies; and therefore it is not uncommon for them to have many CPUs and large amounts of physical memory over which computations and data remain widely distributed during both high and low demand periods. Due to the distributed nature of activities, slowing down a single CPU or a single DRAM can have unpredictable performance ramifications; thus it can be counterproductive to push some part of a server into ultra-low power operation[19]. At the other extreme, when power approaches saturation levels, hardware is ill-positioned to determine or enforce decisions about which software activities can tolerate reduced performance and which ones must continue as before. Thus software must share with hardware the responsibility of determining when and in which component power can be saved.

"Software must share with hardware the responsibility of determining when and in which component power can be saved."

"We implement an optimization heuristic that redistributes the power to maximize the overall performance/watt of the server."

In this article, we propose an autonomic solution for fine-grained control over power performance tradeoffs for server configurations. The solution, as we describe in the section "Monitoring and Data Refining" and sketch in Figure 1, consists of ingredients to observe, analyze, plan, and control the dynamic expenditure of power in pursuance of an application level performance objective that is specified as a *Service Level Agreement* (SLA). As described in section "Experimental Setup and Results," we use a time-varying database query workload running on a recent generation Intel® Xeon® server, which is an E5-2600 class machine[22]. We simultaneously change the power allocation to CPUs and DRAM, and gather performance and power readings through a set of distributed physical and logical sensors in the server. Using these observations, we train models for various phases of the workload. Based on our models, we implement an optimization heuristic that redistributes the power to maximize the overall performance/watt of the server. Experimental measurements show that our heuristic improves performance and power as needed or as permitted by performance objective. The article is organized as follows: the following section, "Related Work" summarizes and compares the related work. The next section, "Background," explains the modeling and optimization planning approach. This is followed by the section "System Architecture for Autonomic Power-Performance Control." "Experimental Setup and Results" describes the experimental results, and the article concludes with "Summary and Future Work."

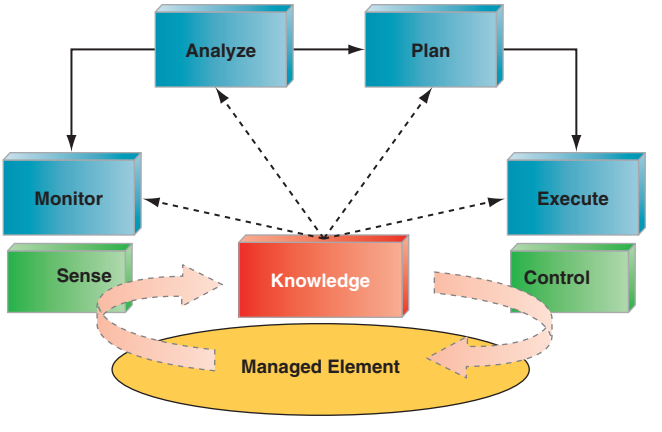


Figure 1: Architectural elements of autonomic control (Source: *A Vision for Platform Autonomy: Robust Frameworks for Systems* (Intel Press, ISBN 978-1-934053-25-6))

Related Work

A vast body of research examines relationships and tradeoffs between processor performance, power, and thermal events. We classify the work related to this contribution as follows:

Platform performance events and power consumption: Several researchers examine the usage of performance event and activity counts for predicting power

consumption. For example, Bircher et. al.[10] identifies a set of microprocessor performance events to estimate total system power. Bellosa et al.[11] demonstrate strong correlations among performance events and power consumption for Pentium® II. David et al.[12] utilize activity counters to predict DIMM power and use the prediction to control DIMM power budget with a Runtime Average Power Limiting (RAPL) approach. Economou et al.[13] correlate AC power measurements with user-level system utilization metrics. Kang et al.[14] show the use of optimized search algorithms and machine learning techniques in a processor design exploration problem to reduce time needed for determining the best configuration. Our work extends such approaches by considering quality of service (QoS) parameters (such as *Throughput* and *Response Time*) for an enterprise application.

- *Using SLAs to obtain power savings:* Several approaches use QoS-based metrics to drive power management in different systems, including real-time systems[6][3][8][9][5], web servers[1], and parallel processing systems[7]. Flautner et al.[3] modify the Linux kernel to save energy by delaying task execution, while ensuring that all tasks meet their deadlines. The implementation of such a mechanism, however, is based upon a priori knowledge of task executions (the deadlines are effectively a proxy for a module that provides performance feedback), while ours is a dynamic scheme. Workload consolidation has been explored as a means to obtain power savings while maintaining SLAs. For example, Sharma et al.[1] explore energy savings in web server clusters by consolidating load onto fewer servers, and turning off the remainder or keeping them in low-power states during low load conditions, so long as the SLAs are not violated. This approach is complementary to our work as Sharma et al.[1] save energy by limiting high activity to a few servers, while we target similar efficiency at the level of individual servers. Hayamizu et al.[4] implement an SLA-based hardware correction scheme that is similar to ours in principle. However, their tuning mechanism reactively adjusts the frequency of operation, resulting in some performance oscillations; in contrast, our mechanism is less volatile as it observes and learns from workload behavior, over a period of time. Other approaches supplement CPU performance feedback with cache miss rates (or metrics to track memory behavior); using the miss rates to build statistical dependence between frequency operating points and memory performance and power consumption[2][8][9].

"Several approaches use QoS-based metrics to drive power management in different systems, including real-time systems[6][3][8][9][5], web servers[1], and parallel processing systems[7]."

Our work differs from the previous approaches in this aspect: we propose to use component energy metrics to show strong correlation between them and quality of service parameters such as *Throughput* and *Response Time* in an enterprise workload, using a machine learning approach. Many of the traditional workload analysis methodologies consist of building upon simulation results obtained from isolated components, involve manual alignment of telemetry data, and include off-line post-processing. These often result in long analysis times, over-corrections, suboptimal tuning and larger guard-bands. In our work we systematically address the issues related to the dynamic collection, processing, and analysis of time-series telemetry

"We propose to use component energy metrics to show strong correlation between them and quality of service parameters such as Throughput *and* Response Time *in an enterprise workload, using a machine learning approach."*

data obtained in time aligned fashion from a variety of physical and logical sensors in the system. Furthermore, we propose exploiting that correlation to redistribute energy amongst the components on the basis of a machine-learning model that is trained online.

Background

We will be concerned with identifying relationships between the total power expended (P), and two measures of performance: response time (R) and throughput (T). In this section, we describe the synthesis and optimization techniques used to build these relationships. Model synthesis uses the following:

- Fine-grained and time-aligned power readings at multiple power rails of the primary components (CPU and DIMM).
- System-level readings corresponding to three quantities, each averaged over a small time interval: (1) P, the total system power, (2) T, the application level throughput, and, (3) R, the response time experienced by requests.

The power readings obtained are aligned with the {P, T, R} tuples, and this entire data collection is then used to divide the {P, T, R} space into classes (phases) so that within each class or phase, a linear function can relate P, T, and R to the power readings. These linear relationships are used in optimization planning, whose objective might be to minimize P (the total system power), or maximize T (the application level throughput), subject to R (the response time) not exceeding a specific threshold. Learning continues online and therefore as workload evolves or changes, the models adapt, and optimization planning adapts as well. The following two sections delve further into the model synthesis and optimization planning operations.

"Linear relationships are used in optimization planning, whose objective might be to minimize P (the total system power), or maximize T (the application level throughput), subject to R (the response time) not exceeding a specific threshold."

"Learning continues online and therefore as workload evolves or changes, the models adapt, and optimization planning adapts as well."

Model Synthesis: Support Vector Machines (SVM)

Support Vector Machines (SVM) technique may be employed to divide the {P, T, R} space into different phases and to obtain linear relationships governing the {P, T, R} variables in each phase. SVM is a computationally efficient and powerful technique invented by Boser, Guyon, and Vapnik[20] that is employed for classification and regression in a wide variety of machine learning problems. Given a data collection relating a set of training inputs to outputs, an SVM is a mathematical entity that accomplishes the following: (a) it describes a hyper-plane (in some higher dimension) whose projection into the input space separates inputs into equivalence classes so that the inputs in a given class have a linear function that maps them to outputs that is distinctive for that class, (b) the hyper-plane whose projection is the SVM, maximizes the distance that separates it from nearest samples from each of the classes, thus maximizing the distances between classes; subject to a softness margin described next, (c) a softness margin that allows a bounded classification error, whereby a small fraction of the inputs, that should be placed one side of the projection are instead placed within a bounded distance on the other side (and therefore misclassified); this margin allows a pragmatic tradeoff between having

a high degree of separation between classes (that is, better distinctiveness) and having too many outliers.

"We consider ten power readings obtained from a set of five sensors per processor, in our two processor machine."

For our analysis in this article, we consider ten power readings obtained from a set of five sensors per processor, in our two processor machine. Equations (1) express each of {*P*, *T*, *R*} as linear functions of the five power readings per processor within each given class or phase. Of the five sensors per processor, three sensor readings (V_{ccp}, V_{tt}, and V_{sa}) yield power going into three broad groupings of functions on the processor, while two sensor readings, both referred to as V_{ddq} measure power in DIMMs that are connected to and controlled from the processor. The explanations of the various subscripts associated with these sensor readings are deferred to Table 1, in the section "Monitoring and Data Refining." Variable *J* represents a given class; and $\{P_J(t), R_J(t), T_J(t)\}$ represents a tuple from sample numbered *t* in the training set; and the various power readings associated with that sample are represented by $V^*(t)$ in equations (1) below:

CPU power readings (6) **Memory power readings (4)**

$$\begin{aligned} P_J(t) - K_P^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \alpha_{Pk}^{ij} V_K^i(t) + \sum_{DRAM\ CH=i} \beta_P^{ij} V_{ddq}^i(t) \\ R_J(t) - K_R^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \alpha_{Rk}^{ij} V_K^i(t) + \sum_{DRAM\ CH=i} \beta_R^{ij} V_{ddq}^i(t) \\ T_J(t) - K_T^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \alpha_{Tk}^{ij} V_K^i(t) + \sum_{DRAM\ CH=i} \beta_T^{ij} V_{ddq}^i(t) \end{aligned} \quad (1)$$

where the phases *J*, constants, K_P^*, K_R^*, K_T^*, and the coefficients α_*^* and β_*^* are all estimated through SVM regression technique. The use of RAPL[12] technique can allow us to simplify equations (1) as we explain further in the section "Experimental Setup and Results":

CPU power readings (2) **Memory power readings (4)**

$$\begin{aligned} P_J(t) - K_P^J &= \sum_{CPU=i} \alpha_P^{ij} V_{RAPL}^i(t) + \sum_{DRAM\ CH=i} \beta_P^{ij} V_{ddq}^i(t) \\ R_J(t) - K_R^J &= \sum_{CPU=i} \alpha_R^{ij} V_{RAPL}^i(t) + \sum_{DRAM\ CH=i} \beta_R^{ij} V_{ddq}^i(t) \\ T_J(t) - K_T^J &= \sum_{CPU=i} \alpha_T^{ij} V_{RAPL}^i(t) + \sum_{DRAM\ CH=i} \beta_T^{ij} V_{ddq}^i(t) \end{aligned} \quad (2)$$

where $V_{RAPL}(t)$ equals the sum, $V_{ccp} + V_{tt} + V_{sa}$, of the power spent in processor activities.

Optimization Planning

Energy and performance models have a number of degrees of freedom and conflicting objectives that are difficult to optimize collectively. For example consider the following objectives: (a) best performance/watt (b) staying within a power limit (c) response time <= an SLA threshold. Conflicts can manifest among these objectives, for example, with considerations such as:

- How to obtain a given throughput within a *system power budget?*
- How to obtain a given throughput under a *response time threshold?*

"Also in the general case, performance is affected by both the power spent in processors and in DIMM modules."

In the common case P (total system power) is affected by both performance targets: throughput and response time. Also in the general case, performance is affected by both the power spent in processors and in DIMM modules. Thus optimization planning must grapple with meeting a compound objective: one in which power expended towards one objective may, in general, come at the cost of another. As described later, our experimental setup escaped this particular complexity. However, for completeness, in the next paragraph we sketch how variant objectives can be targeted simultaneously.

Multi-Objective Optimization (MOP): A good introduction to MOP can be found in David et al.[27], and the reader can skip this paragraph without loss of continuity. Once the coefficients of the linear estimation model for power, throughput, and response time are synthesized, multi-objective optimization can proceed with an adaptive weighted genetic algorithm (AWGA). In a genetic algorithm, a successful outcome is defined to be one that that redistributes power in such a way that power, response time, and the reciprocal of throughput are all meeting the viable limits. More generally, a set of fitness functions $\{f_n\}$ one per objective n, determines the optimality of a candidate setting (that is, a vector describing the distribution of power among components) for each of the objectives. In AWGA, for a population ϕ of candidate settings $\{\mathbf{x}\}$, $F_n^{max} = \max(f_n(\mathrm{x})|\mathrm{x} \in \phi)$ and $F_n^{min} = \min(f_n(\mathrm{x})|\mathrm{x} \in \phi)$ compute respectively the fitness bounds for each of a set of n=1,2,…,N objectives, where each $\mathbf{x}$ in ϕ is a vector whose fitness function represents a feasible power distribution among components such as CPUs, DIMMs, and so on. One may then choose an N-objective fitness function F that evaluates an aggregate fitness value. For example, in case of AWGA, F can be chosen as

$$F = \sum_{n=1}^{N} \frac{f_n(\mathbf{x}) - F_n^{min}}{N \cdot (F_n^{max} - F_n^{min})}$$

An evolutionary algorithm selects parents from a given generation of ϕ (usually employing elitism) from which to produce power-feasible offspring as new candidates for the next generation. In the space of objectives, F_n^{min} and F_n^{max} represent extreme points that are renewed at each generation. As the extreme points fitness bounds $\{(F_n^{min}, F_n^{max})|n=1,2,..N\}$ are renewed at each generation, the contribution (weight) of each objective is renewed accordingly.

System Architecture for Autonomic Power-Performance Control

"Achieving power-efficient performance and abiding by power and performance constraints calls for real-time feedback control."

Achieving power-efficient performance and abiding by power and performance constraints calls for real-time feedback control. As Figure 1 depicts, an autonomic system implements continuous feedback-based course corrections with following provisions:

- *Monitoring* infrastructure, to sample or quantify physical and logical metrics, such as power, temperature, activity rates, and, to obtain statistical moments of the metrics.

- *Analysis* modules to distill relationships among monitored quantities—such as between power and temperature and performance, and to determine whether one or more operational objectives are at risk.
- A *planning* element to formulate a course of action such as suspending, resuming, speeding up, or slowing down various parts of a system, in order to effect a specific policy choice—for example, to limit power or energy consumed or to improve performance.
- A capability to *execute* the formulated plan, and thereby to *control* the operation of the system.

The section "Background" discussed the analysis and planning ingredients listed above. Usually, a knowledge base supplements analysis and planning. The knowledge base may be an information repository that catalogs the allowable actions in each system state, or it may be implicit in the logic of the analysis, planning, and control capabilities. In a system designed for extensibility, the knowledge base would typically incorporate an adaptive mechanism that tracks and learns from prior decisions and outcomes. The next section, "Monitoring and Data Refining," discusses the monitoring ingredients, and "Power Control Mechanism" discusses the control ingredients, by using our system setup as an example implementation to draw upon.

Monitoring and Data Refining

Fine-grained and lightly intrusive power-performance monitoring is a key element of an adaptive power management infrastructure. While our setup has a rich external capability for plumbing component power as we will detail next, modern processors and platforms such as the E5-2600 series machines provide internal logical sensors that can be used to estimate component power with requisite accuracy. The data produced by raw monitoring is refined and then used as feed for analysis and planning described earlier. We will describe the refinement procedures alongside monitoring detail in this section.

"Modern processors and platforms such as the E5-2600 series machines provide internal logical sensors that can be used to estimate component power with requisite accuracy."

The ideal monitoring mechanism operates in real time (that is, reports as recent data as possible) and is not subject to the behavior(s) being monitored. In our setup, logical sensors at the OS and software levels provide a near real-time information stream consisting of rates at which common system calls, storage accesses, and network transfers proceed. These logical sensors are supplemented with power sensing through physical sensors. We use two externally powered capabilities:

- A *Telemetry bus* is used to collect data from physical (hardware) and logical (software) sensors and send it to a monitoring agent. In particular, power sensing is accomplished by sensing four types of voltage regulator (VR) outputs at each processor chip, as summarized in Table 1.
- The *Monitoring agent,* to which the telemetry data is sent processes the data, organizes it as a temporally aligned stream of power and performance statistics, and transmits the stream to a remote machine for further storage or analysis.

"A Telemetry bus *is used to collect data from physical (hardware) and logical (software) sensors and send it to a monitoring agent."*

Signal	Description
V_{CCP}	For each multi-core processor socket, the sum of the power drawn into that processor's cores.
V_{SA}	For each multi-core processor socket, the power drawn by a system agent, an entity responsible for power distribution and control to the rest of the socket.
V_{TT}	For each multi-core processor socket, the power drawn for socket level caching and data movements which include power taken up in I/O and shared L3 cache.
V_{DDQ}	Each multi-core processor socket has several DRAM interfaces. VDDQ measures the power drawn for memory attached to these interfaces. Two signals per processor, each covering one pair of channels at that processor sum up to provide power expenditure for DRAM that the processor controls.

Table 1: Power Sensing Capabilities
(Source: Intel Corporation, 2012)

"The monitoring infrastructure in our system provides us with the ability to obtain five distinct power readings per multi-core processor."

The monitoring infrastructure in our system provides us with the ability to obtain five distinct power readings per multi-core processor. The first three are described in the first three rows in Table 1, and they together add up to the total power consumed by each processor. The three readings do not include the power for the memory ranks that are controlled by the processor. Each processor controls four memory channels with multiple DIMMs per channel; each pair of memory channels furnishes one V_{DDQ} signal as shown in the fourth row of Table 1; summing those two V_{DDQ} readings gives the power expended in memory subsystem at each processor.

The data collected by these sensors is refined through successive transformations as shown in Figure 2, and described below:

- *Sensor Hardware Abstraction (SHA) Layer:* This layer interacts with the sensors and communication channels. It uses adaptive sampling so that measurements are only as frequent as needed, and it eliminates redundancies.
- *Platform Sensor Analyzer:* This layer removes noise, and, isolates trends, which makes it easier to incorporate recent and historical data as inputs in further processing.
- *Platform Sensor Abstraction:* This layer provides a programming interface for flexible handling of analyzed sensor data by control procedures implemented above it.
- *Platform Sensor Event Generation:* This layer makes it possible to generate signals. Signals facilitate event-based conversations from control procedures, thereby allowing further control to be hosted in a distributed set of containers (such as local or remote controller software, OS modules, and so on).

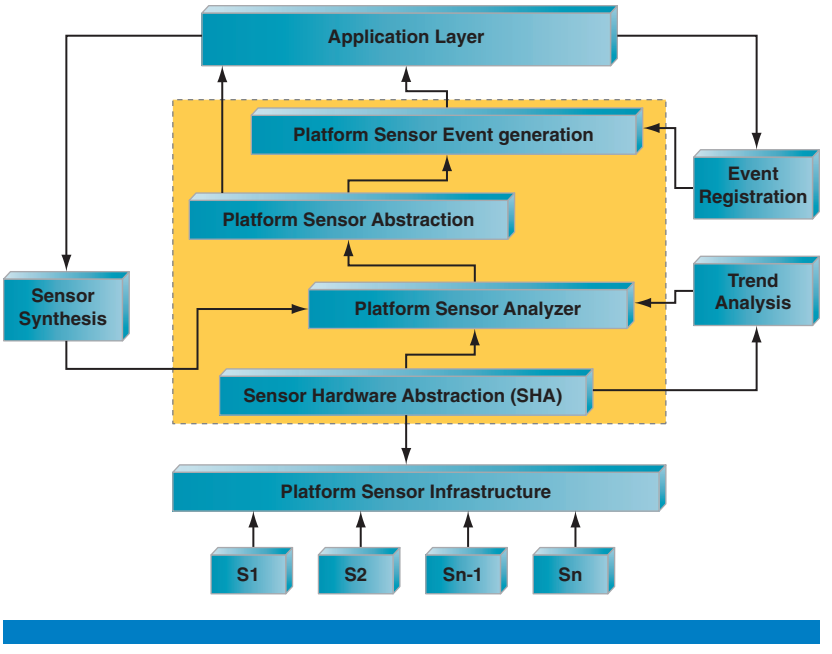


Figure 2: Sensor network model: sensor network layered architecture (S1, S2 . . . Sn represent platform sensors: temperature, power, and so on) (Source: *A Vision for Platform Autonomy: Robust Frameworks for Systems* (Intel Press, ISBN 978-1-934053-25-6))

The successive refinements described above bridge the gap between the raw data that sensors produce and the processed, orderly stream of performance and power readings and alerts that software modules can receive and analyze further.

Estimation

While a machine can be readily upholstered with a metered power supply to sense total power, an instrumentation capability that yields the fine-grained decomposition of power as shown in Table 1 requires nontrivial effort. Besides, adding many physical power sensors in production machines is neither necessary nor practical in terms of cost.

"Event counting capabilities in modern machines provide a potent alternative means of estimating component power, when direct measurement is not practical."

Event counting capabilities in modern machines provide a potent alternative means of estimating component power, when direct measurement is not practical. One simple yet accurate way of estimating the power draw for recent CPUs is to project it on the basis of utilization and P-state residencies, based on trained models. Such training can be made more accurate by including execution profiles that capture what fraction of instructions fall into each of a small set of categories (such as SIMD, Load/Store, and ALU). DRAM power can similarly be estimated on the basis of cache miss counts, or DRAM operations that are counted at memory controllers and tracked through processor event monitors. How accurately can one tie the power consumptions to such proxy measures of power is a question we take up as part of our future work; we note here that solutions such as the Intel® Intelligent Power Node Manager[24] or the Intel® Data Center Manager[25] can also provide measurements of power that we obtain through the added-in power

and temperature sensors in our experimental setup. DRAM power estimation allows us to measure DRAM energy at DIMM granularity with sufficient accuracy, which enables efficient control of DRAM RAPL states. Efficient control of DRAM energy allows us to not only reduce the cost of hardware infrastructure, but also improves the energy efficiency by reducing guard-bands otherwise required to compensate for under-prediction. Furthermore, over-prediction can also be reduced to avoid any performance degradation.

Power Control Mechanism

Once the desired power allocation among components is identified, the next step is to implement the allocation. The processor provides at least two ways by which privileged software can modify its power draw. The first is to change the P-states and C-states as described in [23]. The second, a less intrusive method, is to change the average power level using a control available in recent systems such as the E5-2600 series. This control is known as *Runtime Average Power Limiting* (RAPL) capability for CPUs and DRAM modules.

"A less intrusive method, is to change the average power level using a control available in recent systems such as the E5-2600 series. This control is known as Runtime Average Power Limiting *(RAPL) capability for CPUs and DRAM modules."*

CPU RAPL provides interfaces to set a power budget for a certain time window, and let hardware meet the energy targets[21]. Specifying power limit as an average over a time window allows one to represent physical power and thermal constraints. Privileged software can use the RAPL capability by programming to an interface register the desired average level of power to which the hardware can guide the processor via its own corrective frequency adjustments[21] over a programmable control window. The window size and the power limit are selected so that either at a single machine level or at a data center level, correction in a machine's power is driven quickly. In practice, the window size can vary between milliseconds and seconds; the former to satisfy power delivery constraints the latter to manage thermal constraints. Note that by setting window size to one, RAPL can be used to limit instantaneous power when necessary. The RAPL concept extends to memory systems as well[22], aided by the integration of the memory controller into each multi-core processor in several recent versions of Intel platforms. Although CPU and memory energy can be regulated individually, it is possible to build a coordinated approach where power regulation is a part of a joint optimization function. While more details of RAPL technology are beyond the scope of this article, Intel[21] and David et al.[12] may be consulted for more information.

Experimental Setup and Results

The workload used in our study is the query-only portion of the Transaction Processing over XML (TPoX) benchmark[16] version 2.0, with the Express-C edition of IBM DB2 database management software[17]. As the workload driver for TPoX draws very modest computational effort from the machine when

using the query-only workload, for configuration simplicity the workload driver is co-hosted together with the database management software on the system under test; in any case, this can be changed easily. The choice of TPoX benchmark as the workload for this study is motivated by its ability to impose the kind of broad front stress on the computing system that is representative of a modern enterprise—with its large numbers of threads, complex concurrency interactions, and appreciable memory footprint. IBM DB2's self-tuning memory manager eliminated the need to perform any fine-tuning in our setup as we varied the imposed workload. The system under test employed two CMP processors, from the Intel Xeon E5-2600 series. The machine was furnished with 64 GB DDR3 DRAM. A single Intel SATA solid-state disk drive with a capacity of 160 GB provided the mass storage for database tables and log files, with sufficient random I/O throughput to eliminate disk wait times during workload execution.

"The choice of TPoX benchmark as the workload for this study is motivated by its ability to impose the kind of broad front stress on the computing system that is representative of a modern enterprise."

For analysis and optimization planning, we simplify the model to reduce the complexity of equations (1). The perturbations in system power (P) and those in the application level response time and throughput that result from experiments in which memory power changes to any degree are negligible in comparison with those in which CPU power changes result. In part this is due to low sensitivity of workload performance to the bandwidth and latencies of DRAM accesses; in part it is due to the much smaller dynamic range of power variation that is possible for memory compared to that which is possible for the CPUs. Thus the second term(s) on the right hand side of equation (1) gets absorbed in the constants (K*) on the left hand side. This reduction leads to the following change from the equations (1):

CPU power readings (6)

$$\begin{aligned} P_J(t) - \bar{K}_P^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \bar{\alpha}_{Pk}^{ij} V_k^i(t) \\ R_J(t) - \bar{K}_R^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \bar{\alpha}_{Rk}^{ij} V_k^i(t) \\ T_J(t) - \bar{K}_T^J &= \sum_{CPU=i} \sum_{k=ccp,tt,sa} \bar{\alpha}_{Tk}^{ij} V_k^i(t) \end{aligned} \quad (3)$$

A second simplification arises from the use of the RAPL technique. Under CPU RAPL, hardware takes on the responsibility of ensuring that the sum, $V_{ccp} + V_{tt} + V_{sa}$, is maintained at the specified CPU RAPL value, for each of the two processors (each processor is a multi-core chip). While we continue to obtain the full set of power sensor readings (that is, V_{ccp}, V_{tt}, and V_{sa}) as input to model training, the individual variations in V_{ccp}, V_{tt}, and V_{sa} are not as useful in training as their sum (since it is the sum that can be controlled).

Hence in the SVM model formulation instead of fitting three separate coefficients per processor ($\bar{\alpha}^*$ in equations (3)), one for each of V_{ccp}, V_{tt}, and

V_{sa}, we train one coefficient per processor ($\tilde{\alpha}^*$ in equations (4) below) that multiplies their sum, $V_{sum} = V_{ccp} + V_{tt} + V_{sa}$:

CPU power readings ($V_{sum} = V_{ccp} + V_{tt} + V_{sa}$)

$$
\begin{aligned}
P_J(t) - \tilde{K}_P^J &= \sum_{CPU=i} \tilde{\alpha}_P^{ij} V_{sum}^i(t) \\
R_J(t) - \tilde{K}_R^J &= \sum_{CPU=i} \tilde{\alpha}_R^{ij} V_{sum}^i(t) \\
T_J(t) - \tilde{K}_T^J &= \sum_{CPU=i} \tilde{\alpha}_T^{ij} V_{sum}^i(t)
\end{aligned}
\tag{4}
$$

Model Training: Procedure and Evaluation

For the model training phase, we collected fine-grained and time-aligned readings from the power monitoring sensors described earlier, and from a database performance module that kept track of response times and request completion rates. These readings provided the input-output training vectors, $\{P_*(t), R_*(t), T_*(t)$, and $V_{sum}^i(t)\}$ as denoted in equations (4). The training data was obtained under a cross-product of two sets of variations:

"We modified the TPoX workload driver so that it would cause time-varying demand on the machine."

- *Variation of Demand:* We modified the TPoX workload driver so that it would cause time-varying demand on the machine. The modification consisted of using a different "think-time" parameter at different times; the parameter controls how long each of a number of threads in the workload driver waits between the completion of a previous request and the issuance of a new request.
- *Variation of Supply:* We varied the CPU and memory RAPL settings, thereby varying the supply of power to CPU and DRAM. As we noted earlier in this section, the variation in memory power had marginal effects on system power, and on throughput and response time. We ascertained it thus: in order to check for any correlation or dependency in the 10 predictors (component energy variables) of the original equation 1, we selected the best among the predictors for throughput and response time, and then tested sequentially how the addition of the next potential predictor could improve accuracy, and wound up with VDDQ variables as superfluous in this experimental setup.

We executed TPoX with think-time varying from 0 to 100. For each think-time CPU RAPL limits were varied between 20 W and 95 W. SVM model training on the basis of this data was then used to categorize the data into distinct phases (J), following which the SVM model parameters for each phase, $\{\tilde{K}_R^J, \tilde{K}_T^J, \tilde{\alpha}_P^{ij}, \tilde{\alpha}_R^{ij}, \tilde{\alpha}_T^{ij}\}$ were evaluated.

The SVM based classification yielded decomposition into three phases shown in Figure 5. Accordingly, three different sets of modeling parameters (that is, for J = 0, 1, 2) in equations (4) relate CPU RAPL parameters to total system power, throughput, and response-time outcomes. Figures 3 and 4 show the close agreement between estimated and measured results from the training. Figure 3 shows how the total wall power estimated on the basis of RAPL

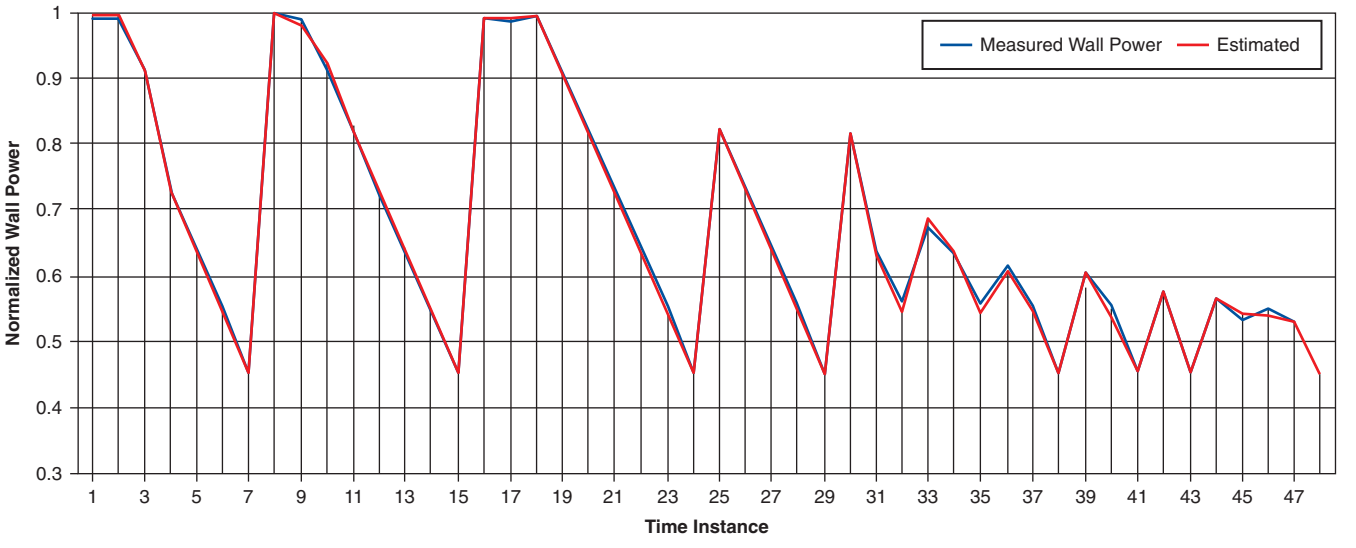


Figure 3: Wall power measured versus estimated (as function of component power)
(Source: Intel Corporation, 2012)

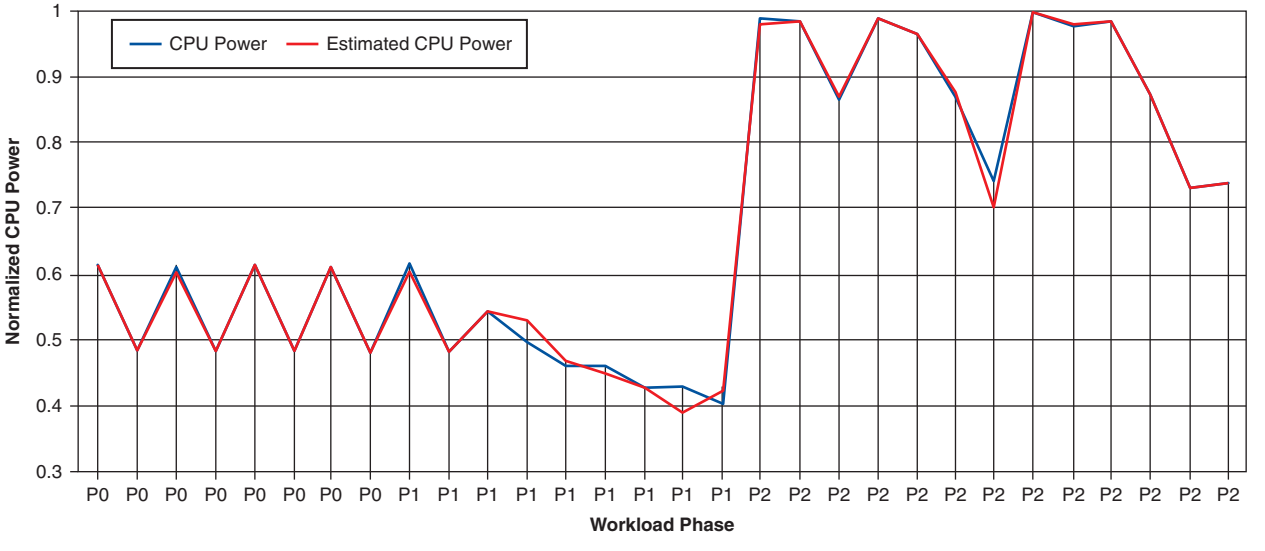


Figure 4: CPU power actual needed versus estimated. Estimated CPU power is phase-wise, and based on the throughput and target latency requirements.
(Source: Intel Corporation, 2012)

parameters via the first equation in (4) compares with that actually measured. The comparison is shown under variations in demand and supply that were introduced as described earlier in this section; the time instance values on the x-axis have no particular significance except as sample points.

Figure 4 also shows how in each of three phases the measured and estimated power values compare; in this case, the estimation is drawn in two steps: first,

the CPU RAPL values are estimated based on desired throughput and response time by inverting the last two of the equations (4) and then the first equation of (4) is used to derive estimated system power. The graphs show excellent agreements between measured and estimated values at most sample points.

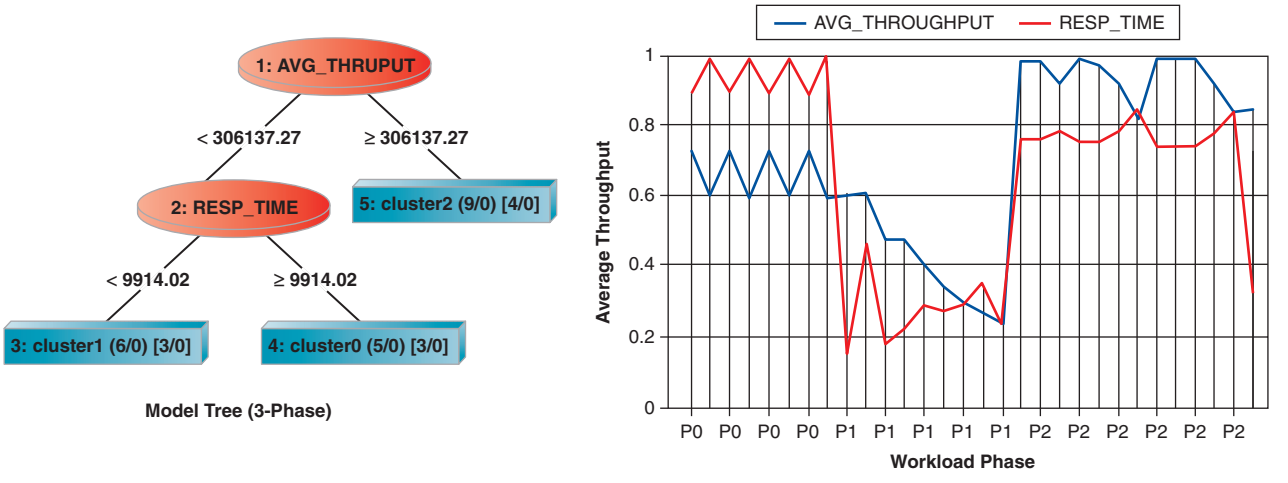


Figure 5: Model tree depicting three phases (P0, P1, and P2) in the workload characterized by throughput and response time
(Source: Intel Corporation, 2012)

"On an average our machine learning regression function supported by SVM machines delivers accuracy between 97 and 98.5 percent."

On an average our machine learning regression function supported by SVM machines delivers accuracy between 97 and 98.5 percent. Each phase is trained for its own performance and latency model coefficients.

Optimization and Control

Figure 6 depicts an example consisting of four different workload conditions in which a server may exist at some point. On the x-axis, *tt* 00, *tt* 10, and *tt* 20 stand for three different think times of 0.0 ms, 10.0 ms, and 20.0 ms respectively. The y-axis is used to show response times. The red colored multi-segment line in Figure 6 connects four workload points (W_1, W_2, W_3, and W_4). These four workload points are four randomly selected perturbations in demand and supply: for example, W_1 results from setting a think time of 20.0 ms and a CPU RAPL value of 40 watts; W_2 results from a think time of 0.0 ms (driving a higher arrival rate than W_1) and with a CPU RAPL value of 50 watts, and so on.

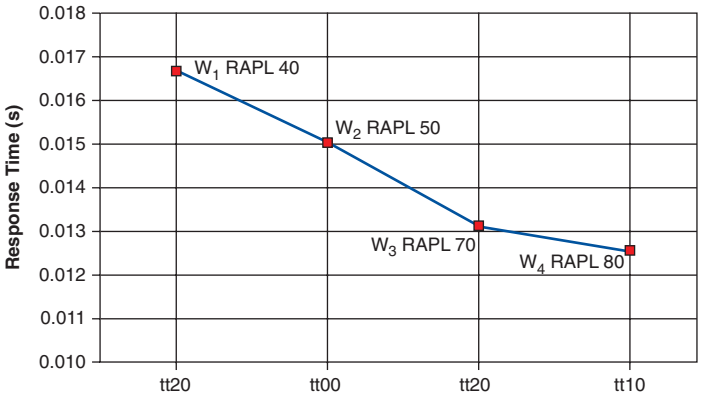


Figure 6: Response time at four arbitrarily selected points reflecting four possible TPoX workload and server conditions (Source: Intel Corporation, 2012)

If none of the response times for (W_1, W_2, W_3, and W_4) in Figure 6 exceeded a desired performance objective—say a Service Level Agreement (SLA) target of $R = 20.0$ ms, then it would be desirable to save power by reducing performance so long as the higher response times still stay below the target of 20.0 ms. On the other hand, if at any of these workload points the response time exceeds a desired threshold, then it would be desirable to improve performance by increasing the power at that point, in order to meet the SLA. In general, an SLA may spell out throughput and response time expectations, and may include details such as the fraction of workload that must complete within a threshold amount of response time under differing levels of throughput. For ease of description, we consider a simple SLA setting: that the response time, averaged over small time intervals (that is, 1 second), not exceed a static target value of 14.0 ms; this is shown in Figure 7 by the blue line, $R = 0.014$.

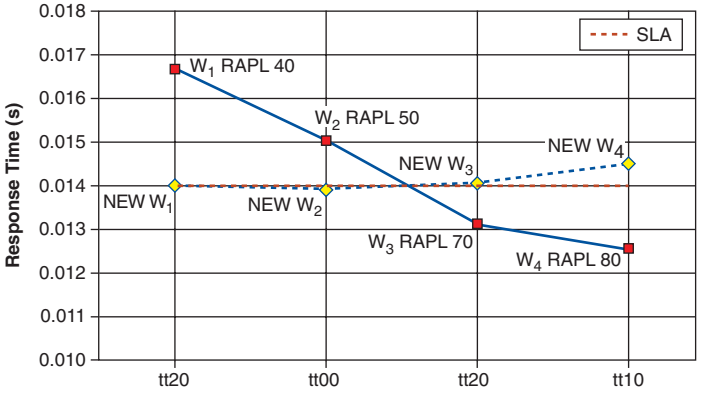


Figure 7: Illustration of improvement in the response time using proactive control of CPU power using CPU RAPL. (Source: Intel Corporation, 2012)

"In order to reduce frequent course correction a control policy may permit overshooting the SLA target by a small margin in either direction."

"Phase-aware CPU power scaling yields significant power reduction at all performance levels relative to isolated tuning."

Figure 7 illustrates new workload points (shown in green) that result from proactive power-performance control through the use of a trained SVM model. New RAPL settings (higher CPU power) computed using the trained model reduce the response times for W_1 and W_2 from their previous values (by 15 percent and 7 percent respectively) to new values that are much closer to the SLA, as shown in Figure 7. Similarly, the model training produces lower CPU power settings for W_3 and W_4 that lead to power savings at the cost of higher response times and to 11.5-percent improvement in energy efficiency. Incidentally, the new setting for W_4 misses the SLA target by a small but not negligible margin, which could force a re-computation of the CPU RAPL setting in a next iteration. Note that in order to reduce frequent course correction a control policy may permit overshooting the SLA target by a small margin in either direction. In the example of Figure 7, the new RAPL settings for W_1 and W_2 reduced response times by 15 percent and 7 percent respectively and the new RAPL settings. In this way phase-aware CPU power scaling yields significant power reduction at all performance levels relative to isolated tuning.

Summary and Future Work

This article described an autonomic approach for fine-grained control over power-performance tradeoffs at a single server level. It is comprised of observing, analyzing, planning, and controlling the dynamic expenditure of power while maintaining an application level performance objective. We used a time-varying database query workload on state-of-the-art database management software running on current generation hardware as the case study vehicle for our example.

In summary, coordinated budgeting using phase-aware optimization can be used to maintain system balance between performance and power-efficiency targets. Experimental setup allows continuous monitoring of workload and planning energy allocation by predicting the effects on performance. A reconfigurable power allocation infrastructure directs power-control requests to each component.

Future work will expand the value proposition of the approach to multiple machine configurations, at the rack and data-center level. Since it would entail measurement and control over a larger set of local and global objectives, we will mix *estimation* alongside the use of hardware sensors, to simplify monitoring. The expanded set of objectives will include a mix of workloads, with compound SLAs covering response times, throughputs, and arrival rates, and we anticipate the inclusion of multi-objective optimization techniques [27] to satisfy diverse requirements.

References

[1] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, Z. Lu, "Power-aware QoS Management in Web Servers," in *In Proceedings of the 24th IEEE Real-Time systems Symposium (RTSS.03),* Cancun, pages 63–72, 2003.

[2] X. Fan, C. Ellis, A. Lebeck, "The Synergy Between Power-aware Memory Systems and Processor Voltage Scaling," in *Workshop on Power-Aware Computing Systems*, pages 164–179, 2002.

[3] K. Flautner, T. Mudge,"Vertigo: Automatic Performance Setting for Linux," *SIGOPS Oper. Syst. Rev.*, *36*(SI):105–116, 2002.

[4] Y. Hayamizu, K. Goda, M. Nakano, M. Kitsuregawa, "Application-aware Power Saving for Online Transaction Processing using Dynamic Voltage and Frequency Scaling in a Multicore Environment," in *Proceedings of the 24th international conference on Architecture of computing systems*, ARCS'11, pages 50–61, Berlin, Heidelberg: Springer-Verlag, 2011.

[5] C. Isci, G. Contreras, M. Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 359–370, Washington, DC, USA, IEEE Computer Society, 2006.

[6] W. Kim, D. Shin, H. Yun, J. Kim, S. Min, "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-time Systems", in *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, RTAS '02, Washington, DC, USA, IEEE Computer Society, 2002.

[7] J. Li, J. Martinez.,"Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors,"*International Symposium on High-Performance Computer Architecture,* 0:77–87, 2006.

[8] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, R. Rajkumar, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," in *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, pages 35–44, 2002.

[9] C. Poellabauer, L. Singleton, K. Schwan,"Feedback-based Dynamic Voltage and Frequency Scaling for Memory-bound Real-time Applications," in *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 234–243, Washington, DC, USA, IEEE Computer Society, 2005.

[10] W. Bircher and L. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," in *IEEE International Symposium on Performance Analysis of Systems and Software*, April 2007, pp. 158–168.

[11] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems," in *ACM SIGOPS European Workshop*, September 2000.

[12] H. David, E. Gorbatov, U. Hannebute, R. Khanna, C. Le, "RAPL: Memory Power Estimation and Capping," in *ACM/IEEE International Symposium on Low Power Electronic Design*, 2010.

[13] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan, "Full-System Power Analysis and Modeling for Server Environments," in *Workshop on Modeling, Benchmarking, and Simulation* (MoBS), 2006.

[14] S. Kang and R. Kumar, "Magellan: a Search and Machine Learning-based Framework for Fast Multi-core Design Space Exploration and Optimization," *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008.

[15] R. Bianchiny and R. Rajamony, "Power and Energy Management for Server Systems," in *IEEE Computer* 2004.

[16] TPoX, http://tpox.sourceforge.net/

[17] DB2 Express-C Edition http://www-01.ibm.com/software/data/db2/linux-unix-windows/edition-express-c.html

[18] IDC, "The 2011 Digital Universe Study," http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm

[19] D. Meisner, B. Gold, T. Wenisch, "PowerNap: Eliminating Server Idle Power," in *ASPLOS* 2009.

[20] B. Boser, I. Guyon, V. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory (COLT '92)*, ACM, New York, NY, USA, 144–152.

[21] Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3A.

[22] Intel® Core™ i7 Processor Family for the LGA-2011 Socket, Datasheet Volume 2.

[23] S. Siddha, "Multi-core and Linux Kernel," Intel Open Source Technology Center

[24] Intel® Intelligent Power Node Manager, http://www.intel.com/content/www/us/en/data-center/data-center-management/intelligent-power-node-manager-general.html

[25] Intel® Datacenter Manager (DCM), http://software.intel.com/sites/datacentermanager

[26] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, E. Weissmann, "Power Management Architecture of the 2nd Generation Intel® Core™ Microarchitecture, formerly code named Sandy Bridge" http://www.hotchips.org/archives/hc23/HC23-papers/HC23.19.9-Desktop-CPUs/HC23.19.921.SandyBridge_Power_10-Rotem-Intel.pdf

[27] A. Konak, D. W. Coit, A. E. Smith, "Multi-Objective Optimization Using Genetic Algorithms: A Tutorial," Reliability Engineering & System Safety, volume 91, issue 9, Elsevier, 2006.

Author Biographies

Martin Dimitrov obtained his B.S. degree in computer science from Bethune–Cookman College in 2004 and his PhD in computer science from the University of Central Florida in 2010. Martin joined Intel in 2010 as a systems engineer. Currently, Martin works in enabling and optimizing enterprise software for Intel server platforms. In addition, Martin is an active researcher in the Greenpoint initiative, which aims at optimizing system energy through collaborative software-hardware approaches. Martin can be contacted at Martin.P.Dimitrov@Intel.com

Kshitij Doshi is a principal engineer in the Software and Services Group at Intel Corporation. He has a bachelor of technology degree in electrical engineering from the Indian Institute of Technology (Mumbai) and a master's degree and PhD in computer engineering from Rice University. His research interests span operating systems, optimization of performance, power, and energy in enterprise solutions, database architectures, and virtual machines. He can be contacted at kshitij.a.doshi@intel.com

Rahul Khanna is a platform architect at Intel Corporation involved in development of energy efficient algorithms. Over the past 17 years he has worked on server system software technologies including platform automation, power/thermal optimization techniques, reliability, optimization, and predictive methodologies. He has authored several technical papers and book chapters in the areas related to energy optimization, platform wireless interconnects sensor networks, interconnect reliability, predictive modeling, motion estimation, and security, and he holds 27 patents. He is also the co-inventor of the Intel IBIST methodology for high-speed interconnect testing. His research interests include machine learning based power/thermal optimization algorithms, narrow-channel high-speed wireless interconnects, and information retrieval in dense sensor networks. Rahul is a member of IEEE and the recipient of three Intel Achievement Awards for his contributions in areas related to advancements of platform technologies. He is the author of book *A Vision for Platform Autonomy: Robust Frameworks for Systems*. He can be reached at rahul.khanna@intel.com

Karthik Kumar is a software engineer in the Software and Services Group at Intel Corporation. He obtained a bachelor's degree in engineering from Anna University (India), and a master's degree and PhD in computer engineering from Purdue University. His research interests span energy and performance optimization in computer systems. He can be contacted at karthik.kumar@intel.com

Christian Le is server power and thermal architect in Intel's Data Center and Connected Systems Group. He has spent 16 years designing system thermal and power management solutions. He is currently focused on datacenter power optimization and platform autonomics technologies. He can be reached at Christian.le@intel.com

A ROBUST AUTONOMIC FRAMEWORK FOR MEMORY THERMAL, POWER, AND THROUGHPUT MANAGEMENT

Contributors

Christian Le
Intel Architecture Group,
Intel Corporation

Robin Steinbrecher
Intel Architecture Group,
Intel Corporation

Felipe Ortega
Intel Architecture Group,
Intel Corporation

Christopher Cox
Intel Architecture Group
PC Client Group,
Intel Corporation

Rahul Khanna
Software and Solutions Group,
Intel Corporation

Mrittika Ganguli
Data Center and Connected Systems Group, Intel Corporation

Advances in technologies are driving integration of more features and higher performance into smaller chip designs that are resulting in increasing power and thermal density and design complexity. Mobile computing devices require higher performance with more battery life while the need for higher performance servers places more demands on data center–server efficiency due to rising costs of energy, operation, and infrastructure. Holistic dynamic thermal and power management explicitly couples thermal and power management from the chips to computer to server and to the data center. Additional challenges are placed on hardware and software development and validation time as design complexity and density increase. A common architecture for power and thermal management facilitates more efficient hardware and software development, validation, and reuse across segments. This article proposes a common memory open and closed loop thermal management (OLTM, CLTM) integrated thermal and power capping with a technique called running average power limiting (RAPL) architecture. These traditional power and thermal features are described as part of an autonomic framework for power and thermal management integrated with advanced interrupts/signaling and power limiting (RAPL) concepts. We introduced efficient RAPL that enforces power limits over a sliding time widow, while minimizing performance impact in highly dynamic and transient data center workloads. We also introduce the concept of a standard software interface through standard configuration architecture (SCA) that produces a uniform telemetry and event signaling infrastructure across client and server segments. We also introduce the design considerations for a data center management data aggregation and workload autonomics framework that can make use of power, thermal, and throughput constraints to influence reduction of maintenance costs, to drive greater efficiencies for more flexibility, and to dynamically scale resource pools.

Introduction

An explosion of Internet growth is expected due to the exponential growth in the number of globally connected users, computing devices, and new emerging segments. Contents will drive more performance, density, and energy efficiency in data center and computing devices[1][2]. According to an EPA report to Congress[3], the annual electricity use by data centers for 2011 is projected to be nearly double from 60 billion kWh to more than 120 billion kWh, representing a USD 7.4 billion annual electricity cost and an increase from 1.2 percent to 3 percent of electricity consumption in the US. The operational cost for cooling and providing power to IT equipment is equivalent the cost of equipment; the gap between operational versus capital costs will widen with decreasing cost of computing and increasing demand and cost of electricity.

Memory power and thermal cooling is a significant portion of server power consumption and better controlling these two key aspects via autonomic functions is important to the future of green data centers. As an example, Figure 1 depicts projection for a typical Intel® Core™ i7 generation server where both processor and memory constitute a significant portion of the platform power, ranging from 32 percent at idle to 61 percent when active. Given that CPU and memory provide high dynamic range, it is important to address both when designing platform power limiting. Since servers rarely operate at their peak capacity, efficient power capping is deemed as a critical management component of modern enterprise computing facilities. Advancements on the thermal and power management of processors[4] have resulted in features such as temperature sensor feedback for fan speed control, dynamic voltage and frequency scaling for proportional computing, and power meter and power control accessible in-band to the OS[5] and out-of-band to node or data center agents[6] but stops short of common interface. A holistic approach to power and thermal management requires a robust, auto-discoverable framework that enables policy-driven features using software standards, silicon hooks, and manageability containers to accommodate thermal/power budget and performance tradeoffs. Common framework facilitates more efficient hardware software co-design through reuse and fosters software innovations scalable across the computing continuum. Since memory is the less traveled path, in this article we present a robust *thermal throttling framework* for memory that standardizes and enhances the throttling features present in Nehalem (NHM), Sandy-Bridge (SNB) families to:

- Maximize reusability across segments and across platforms
- Balance efficient reuse of existing art and new development
- Minimize cost for any particular segments
- Define common memory power and thermal throttling algorithms
- Running average power limiting (RAPL)
- Leverage validation across generations and architecture divisions
- Scale the architecture and interfaces across servers, desktop and mobile clients

"Memory power and thermal cooling is a significant portion of server power consumption and better controlling these two key aspects via autonomic functions is important to the future of green data centers."

"A holistic approach to power and thermal management requires a robust, auto-discoverable framework that enables policy-driven features using software standards, silicon hooks, and manageability containers to accommodate thermal/power budget and performance tradeoffs."

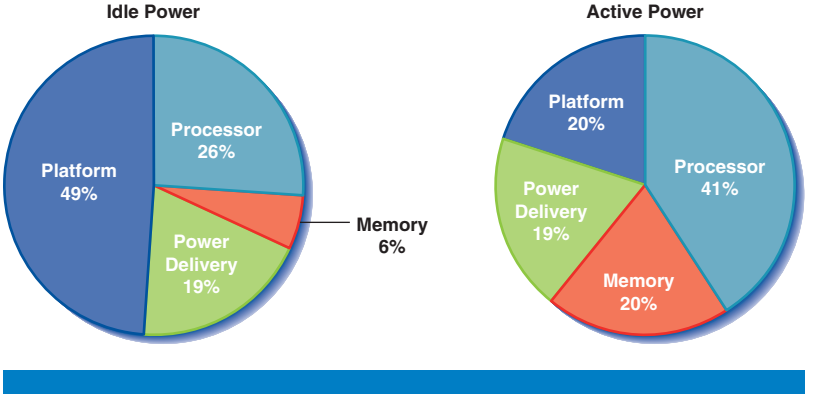


Figure 1: Intel® Core™ i7 server power projections at idle and active modes (Source: Intel Corporation, 2012)

Related Work

Thorough examinations of thermal and power relationships have demonstrated the complex interdependence between thermal, power, and performance. Shah[7], Sharma[8], and Patel[9] demonstrated the importance and sensitivity of facility layout with respect to cooling and the relative difficulty in predicting dynamic thermal loading for optimal design and efficiency. They described how the horizontally scalable topology is driven by Internet computing and the thermal impact as the density of this segment increases in scale and power density and also addressed thermal management and multisystem resource management as a must-have application to the success of this segment.

Proportional energy computing equates to adjusting energy consumed relative to the amount of work being performed. Barroso[3] discussed the value of energy and computing power utilization and the need to improve server energy proportionality profiles. They called to the developer community to come up with a metric at non-peak activity as well as to employ heuristics to refine design through characterization of a system energy performance. Ahuja[10] conducted experiments and projected reliable server operations at higher data center ambient temperature up to 40°C to reduce cooling demands on data centers to facilitate higher ambient operations recommended by the American Society of Heating, Refrigerating and Air Conditioning Engineers (ASHRAE). Shah et al.[7] analyze the hot and cold air mixture using the second law of thermodynamics and present a metric of energy loss where thermal manageability and energy efficiency are both considered simultaneously.

"Although much focus has been given to server and data center energy efficiency and cooling over the past decade, little progress has been made in the standardization of the power and thermal interconnect scheme between the computing device and data center."

Although much focus has been given to server and data center energy efficiency and cooling over the past decade, little progress has been made in the standardization of the power and thermal interconnect scheme between the computing device and data center. Recent works by Khanna, et al.[13][14] proposed dynamic closed loop thermal management framework integrated into Sandy Bridge-EP with a multiple dynamic thresholds scheme to optimize performance and minimize fan energy. And they investigated a memory containerization software scheme to allocate frequently executed object codes into temporal-spatial memory domains (from ranks-to-channel) in order to optimize power consumption. Another novel approach to a DRAM power sensor, employing an estimation and calibration scheme for use in an energy-efficient running average power limiting algorithm in software was proposed by David, et al.[15][16]. These pioneering works pave the way for integrated thermal and power management.

"Our approach gears the architecture towards common memory architecture for power and thermal management unified by power metric and standardized through a scalable interface across desktop, mobile, and server/workstation segments."

The proposal in this article advances on these dynamic thermal management (DTM) and power limiting techniques. Our approach gears the architecture towards common memory architecture for power and thermal management unified by power metric and standardized through a scalable interface across desktop, mobile, and server/workstation segments. We propose a common memory open loop and closed loop thermal management (OLTM, CLTM), thermal and power interrupts and event signaling, open loop and closed loop power limiting (OPL, CPL), running average power limiting (RAPL), and a standard software interface with standard configuration architecture (SCA).

Thermal and Power Autonomic Framework

Complexity in emerging computing systems requires the holistic solution to manage competing dynamics of power, thermal conditions, RAS, resource monitoring, and locality. Autonomics features are solutions built using software-based models and industry standards that can enable automated detection, optimization, correction, and tuning dynamically while making smart decisions to enable high service reliability. The goal of an automated system is to create expert hardware/software models that can automatically perform proactive actions to converge to an optimal solution by analyzing the specified policies and the current state of context the system is running.

"Autonomics features are solutions built using software-based models and industry standards that can enable automated detection, optimization, correction, and tuning dynamically while making smart decisions to enable high service reliability."

The autonomic computing paradigm is modeled after the human autonomic nervous system, where changes in behavior of the system are brought back to an equilibrium state with closed loop control processes. An autonomic system adapts through a set of behaviors that promote stability through managing the system essential variables within their viable limits. The elements of autonomics as defined by Khanna are:

- *Telemetry bus* – Retrieve RAW sensors as well as send the control messages to the respective devices using an efficient interconnect.
- *Monitor agent* – Organize the RAW sensor data, synthesize the statistical characteristics and distribute the data internally or externally.
- *Analysis agent* – Analyze the local power consumption of each device along with the corresponding performance. The analysis agent 1) builds a database of historic trends that can be used to make future decisions, and 2) trains the model by taking a proactive action where power allocation to a random device is incrementally changed and the corresponding performance impact is measured.
- *Control agent* – propagate power control message to the controlled device in a timely manner. The control agent is specific to a device and the power control methodology. A control function identifies the dynamic range of the power control and the granularity at which it can be controlled.
- *Performance analysis agent* – creates statistical model of the performance data to study the performance impact due to device power variation. It measures the system performance with respect to the workload and evaluates the fitness function that is the function of the change in performance between successive measurements.

The goal of autonomic computing is to limit human intervention to extraordinary situations and instead enables general policies and rules as input for a self-management process. Figure 2 is a high level diagram of architecture for a power, performance thermal autonomic. Thermal and power management are the fundamental functional unit of an autonomic application, which contains executable code, exports functional interfaces, behavioral attributes, constraints, and control mechanisms. Thermal and power managed elements have their own private operators that are not visible to other managed elements

"Thermal and power management are the fundamental functional unit of an autonomic application, which contains executable code, exports functional interfaces, behavioral attributes, constraints, and control mechanisms."

"The system configures and reconfigures itself undervarying and unpredictable conditions."

"The system detects suboptimal behaviors and optimizes itself to improve its operational characteristics while keeping the complexity hidden."

and can only be accessed through control functions exported via functional interfaces.

- *Self-configuring* – The system configures and reconfigures itself undervarying and unpredictable conditions.
- *Self-optimizing* – The system detects suboptimal behaviors and optimizes itself to improve its operational characteristics while keeping the complexity hidden.
- *Context awareness* – The system has the ability to understand its operation environment and ability to react to the environmental changes.
- *Open standards* – Autonomic applications should be built upon open standards so that they can be ported across heterogeneous environments consisting of multiple hardware and software. Consequently, it should be built on open protocols and interfaces.

The thermal and power autonomic elements represent the initial two stages of maturity as defined by IBM.

- *Basic* – The expert knowledge of managed elements and its environment is embedded with the IT professional requiring human intervention on even trivial functions.
- *Managed* – Scripting and expert tools automate data sensing, execution, and reporting operations. Once the information is collected, it is analyzed by individual experts to formulate plans and decisions.

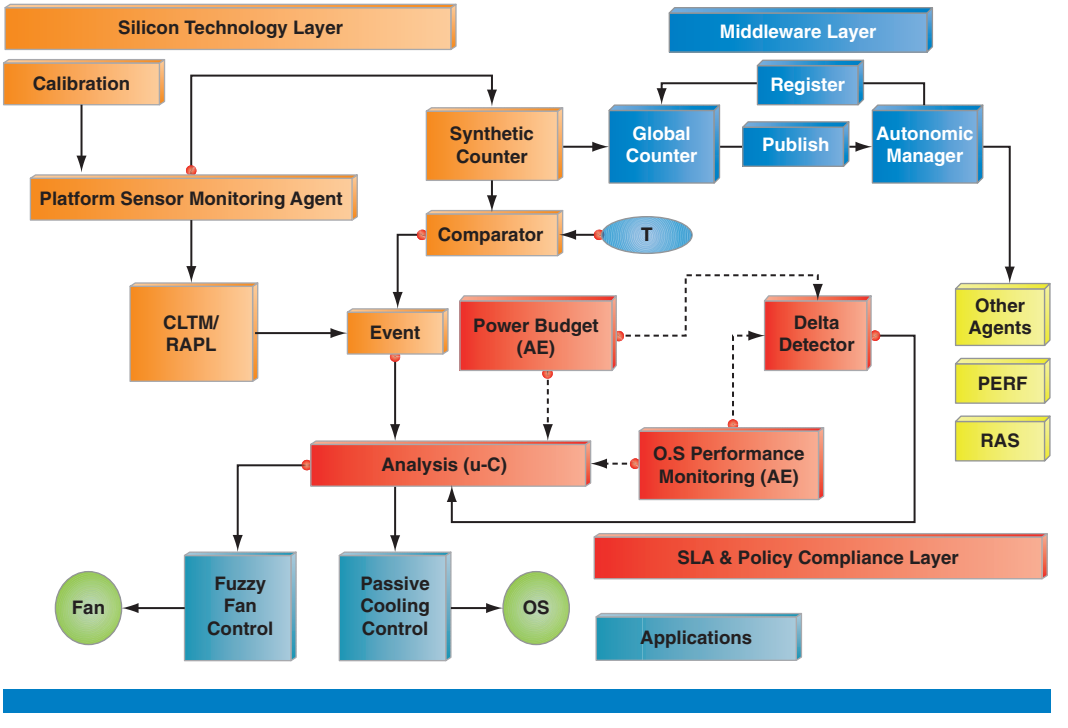


Figure 2: Thermal and power autonomics[31]
(Source: *A Vision for Platform Autonomy: Robust Frameworks for Systems*, Intel Press, 2011)

Thermal and power management functions forms fundamental ingredients of the autonomic infrastructure that play an optimization game to achieve energy efficiency for a given work while operating under multiple constraints. We will discuss the key features and interfaces to these elements, how to make them ubiquitous to platforms, how to export them as open standards compatible software for IT business processes, and how to automate power control as a function of anticipated utilization, performance degradation, available wall power, and cooling capacity.

Fundamentals of Thermal Design

Any cooling system must ensure that each and every component meets its specification. Most components have damage, functional, and reliability temperature specifications. The thermal management of the server ensures compliance to these specifications while taking the appropriate actions to manage that compliance. Figure 3 summarizes the thermal limits and the actions to be taken if they will be exceeded.

"The thermal management of the server ensures compliance to these specifications while taking the appropriate actions to manage that compliance."

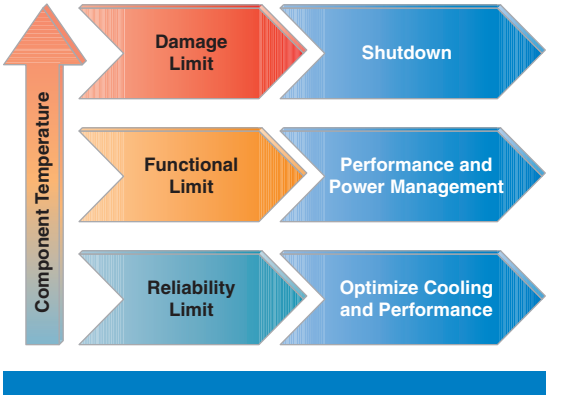


Figure 3: Thermal management (Source: Intel Corporation, 2012)

When designing the cooling and thermal management system, one must comprehend the design load conditions. For processors this is well defined using the thermal design power (TDP) methodology. Intel characterizes workloads and sets the TDP for each SKU to ensure all reasonable workloads can be supported. For memory a similar process is employed based on likely workloads. Bandwidth targets are defined based on these workloads so that the thermal design can be sized appropriately. Memory vendor data is then used to translate that bandwidth into TDP levels for each DIMM type. One bandwidth target for a specific memory speed translates into widely different TDP levels. A DRx8 DIMM may dissipate less than 5 W while an LR-DIMM can consume 15 W for the same bandwidth. The thermal engineer must create a reasonable thermal design that can cover all supported configurations.

Well-designed systems will use power and thermal management features to ensure compliance to component specifications. Figure 4 shows a processor in a system that will be used as an example. The three parts of the equation and the design influences for each are:

"Well-designed systems will use power and thermal management features to ensure compliance to component specifications."

- *System ambient* – inlet temperature to the system: defined in ASHRAE's publication "Thermal Guidelines for Data Processing Environments," this includes any rack effects that can increase the air temperature delivered to the IT Equipment (ITE)
- *Air heating* – increase in air temperature due to upstream heat sources, affected by component placement, upstream component power dissipation, air movers, local air delivery
- *Self heating* – increase in component temperature above local ambient due to the heat dissipated on the device of interest, driven by component packaging, power dissipation and thermal solution (such as a heat sink)

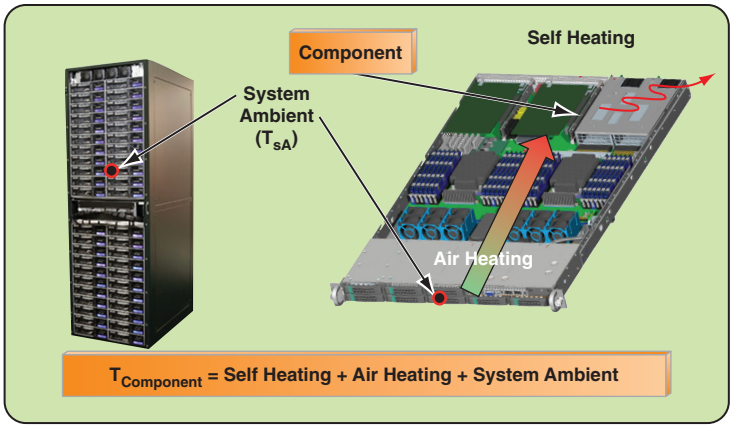


Figure 4: Example component in a system and rack (Source: Intel Corporation, 2012)

"The thermal engineer considers all supported system configurations along with the design requirements such as redundancy."

The thermal engineer considers all supported system configurations along with the design requirements such as redundancy to ensure that the server can adequately meet the specifications based upon the target load requirements.

Two important thermal and power characteristics are nonlinear and, as a result, weigh heavily in the overall power efficiency of the cooling system:

- Fan power is proportional to the cube of airflow (and fan speed), and
- Component and heat sink convective thermal performance is proportional to the inverse of airflow.

The combination of the two characteristics can cause extremely high fan power consumption depending on the driving component thermal characteristics. Optimization between performance and power can become highly complex as a result.

An optimal design can be created by preferentially using features aggressively or nonaggressively. The combination of the feature settings must support the customer's usage.

"Thermal management can be adjusted to preferentially favor performance, power efficiency, high reliability, or acoustics."

Thermal management can be adjusted to preferentially favor performance, power efficiency, high reliability, or acoustics. In some cases the action taken overlaps with that used between these preferences. These designs may overlap, but they all involve tradeoffs between fan speed, component temperatures, acoustic output, power consumption, and performance. Table 1 summarizes key attributes of the designs.

Design or Policy	Fan Speed		Power Management		Result
	Increasing Temperature	Decreasing Temperature	State	Throttling	
Performance	Aggressive	Non-aggressive	Limited	Protective	Best performance at all times
Power Efficient	Optimized	Optimized	Moderate to aggressive	Optimized	Least power consumption
High Reliability	Aggressive	Nonaggressive	Moderate to aggressive	Opportunistic	Reduced down-time
Acoustic	Nonaggressive	Nonaggressive	Moderate to aggressive	Opportunistic	Quiet, non-annoying operation

Table 1: System Thermal Management Tuning
(Source: Intel Corporation, 2012)

The controls for tuning to address these design preferences are fan speed, power states, and throttling. In all cases thermal, power, or activity sensors prompt the response to be taken. Some servers enable the customer to choose the design or "policy" to be used by the server management hardware and firmware at boot time. In an extremely sophisticated design the management controller or another entity could track and learn the usage of the equipment and change settings to either better optimize the server or notify the owners that their server is not optimally configured. They could also suggest changes to the settings, and ensure changes are approved before implementation.

Common Framework Power and Thermal Management

The growth in the Internet has put considerable pressure on data center cooling and power delivery capacity, which has driven up fixed infrastructure costs and operational expenses. Servers rarely operate at their peak capacity and efficient power capping is deemed as a critical component of modern enterprise computing environments. Conventional practice is to overdesign power and thermal characteristics on the conservative side due to complex feature/thermal/power interdependencies and lack of optimal system power modeling and/or system heuristics. Overdesigning at the system component level results in overprovisioning of computer room air conditioning (CRAC) in the data center as well as power supply sizing and operating efficiency[9][20]. Data collected by the Green Grid Association[21] on data center power usage effectiveness (PUE) indicate peak and average efficiency of 40–50 percent compared to theoretical or design power, which led to overprovisioning of cooling and oversizing space by upwards of 50 percent. This data suggests much optimization can be achieved by incorporating well-designed closed loop system thermal and power management.

"The growth in the Internet has put considerable pressure on data center cooling and power delivery capacity, which has driven up fixed infrastructure costs and operational expenses."

The challenge in facilitating a holistic approach requires defining thermal and power telemetry with control mechanisms that are ubiquitous and hierarchical within a data center. For example, analysis of power/performance dynamics requires observation not only at chip granularity, but also at node and rack granularity. At node granularity, resource equilibrium is maintained as a result of complex interaction between competing silicon components. This is analogous to inter-node interaction that maximizes the performance for a given power/thermal budget. The fundamental goals of efficient energy management

"The challenge in facilitating a holistic approach requires defining thermal and power telemetry with control mechanisms that are ubiquitous and hierarchical within a data center."

are to maximize energy used while guaranteeing that the power consumption is never so high that the chips in a platform exceed its junction temperature limit. Limiting platform memory power is a critical requirement for platform power budgeting capabilities. Power budgeting allocates power amongst different platform components to maintain an overall platform power limit.

A high level block diagram of common memory thermal and power limiting high level architecture is shown in Figure 5. The architecture is comprised of dynamic closed loop thermal management (D-CLTM) and dynamic open loop thermal management (D-OLTM), both converging with platform power limiting capability employing a running average power limiting (RAPL) algorithm. The dynamic closed loop thermal management (D-CLTM) scheme is based on traditional closed loop thermal throttling (CLTT) with the capability for the software to reconfigure based on platform power and thermal heuristics. Dynamic thermal management enables the highest performance at lowest power implementation by reducing thermal guard bands, which maximizes the energy efficiency. Well-defined states initiate thermal management actions as a result of thermal events and finer well-managed fan speed control along with thermal protection through throttling. Thermal throttling must be driven by the need to protect both data and component health.

"Dynamic thermal management enables the highest performance at lowest power implementation by reducing thermal guard bands, which maximizes the energy efficiency."

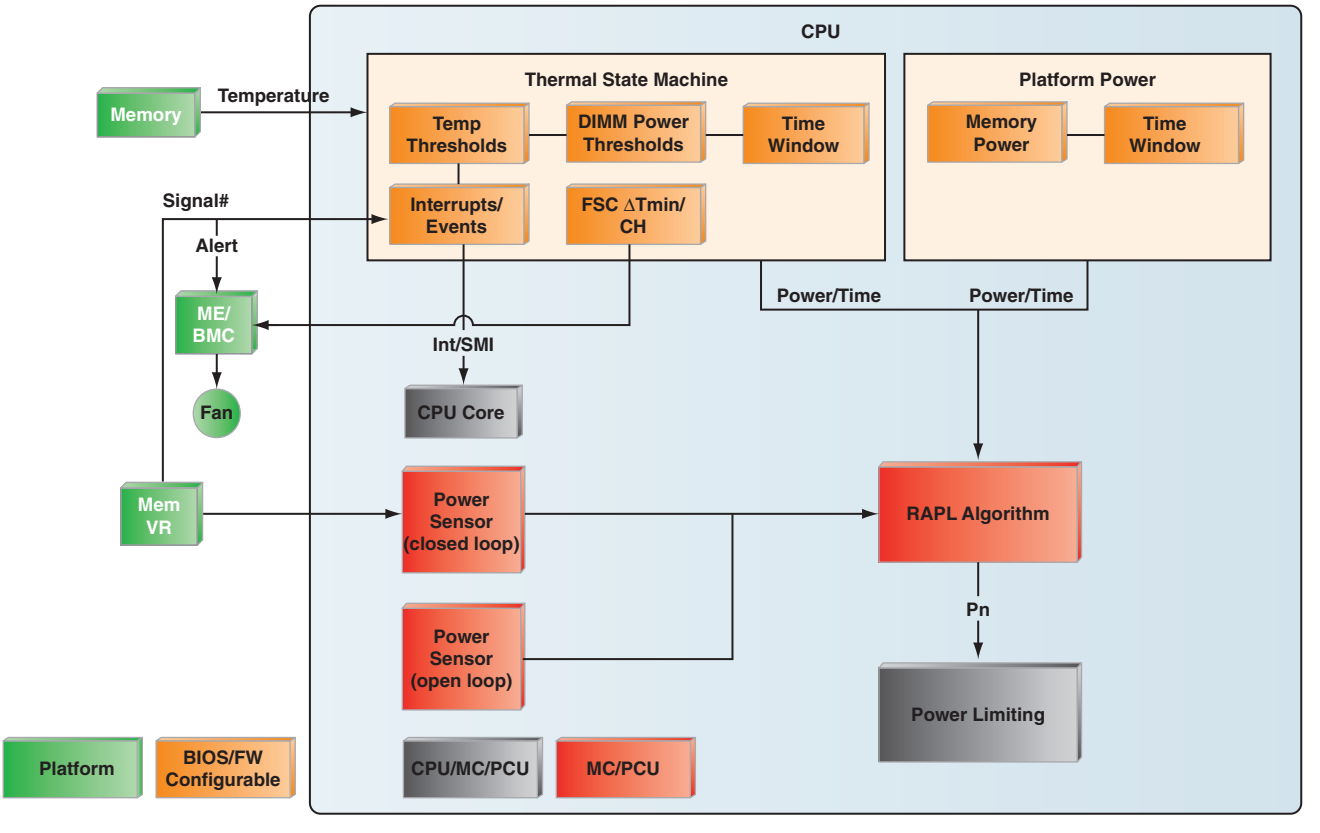


Figure 5: Common framework for thermal and power management with running average power limiting (Source: Intel 2010)

The thermal state machine manages memory temperature data and provides minimum temperature margin of the hottest DIMM in the channel for platform fan speed control. A thermal sensor from memory subsystem (or equivalent) provides closed loop feedback to the processor to manage memory power or throttle activity to protect the memory from exceeding memory temperature limits. The thermal state machine also contains programmable registers for temperature DIMM thresholds, DIMM thermal limits, and the time window. For platforms without memory temperature sensing capability, DIMM power meter and running average power limiting algorithms serve as the virtual thermal closed-loop process control. The thermal state machine also contains the necessary logic and hardware for external memory thermal events such as memory VR signal# or equivalent input to trigger throttling. Out-of-band alerts are monitored by a platform management controller to trigger temperature polling action for fan speed control. Single bidirectional signal# can be encoded for input and output mode. Signal# event temperatures are programmable to correspond to different DIMM temperature thresholds. Interrupts are generated upon thermal threshold crossing to the OS via SMI/MSI for in-band platform thermal management.

"The thermal state machine manages memory temperature data and provides minimum temperature margin of the hottest DIMM in the channel for platform fan speed control."

Memory RAPL Architecture

Power capping provides benefits in the data center, acting as a safety valve by protecting the power distribution hierarchy against overdraw and enabling effective usage of the available power and thereby increasing rack population. Dynamic power capping is a primary power control requirement that must be addressed by a power management solution. Running average power limiting (RAPL) is a feature for limiting the power consumption to a programmable level of various hardware elements based on the energy consumed over a programmable time window. RAPL heuristically controls memory power while maximizing bandwidth and smoothing the effects of bandwidth limiting. Efficient enforcement of power limits over a time widow reduces performance impact for highly dynamic and transient data center workloads. Rather than setting instantaneous limits, RAPL maintains energy credits, which are traded to fulfill memory performance demands and accumulated when that demand is low. If the average workload memory bandwidth requirements are within the specified power limits, the system will not experience any performance degradation even though its memory demand over short periods of time may well exceed the average power limit.

"Power capping provides benefits in the data center, acting as a safety valve by protecting the power distribution hierarchy against overdraw and enabling effective usage of the available power."

Memory RAPL architecture is comprised of three principal components: power measurement logic, a power limiting algorithm, and memory power limiting control. Figure 6 illustrates memory RAPL architecture where power measurement logic provides an accurate mechanism for measuring memory power. Calibrated weights can also be used to implement a cost-effective memory power measurement scheme. An alternative power metering scheme can be implemented through instrumentation in the voltage regulator (VR) with capability for accurate power metering. A power limiting algorithm

tracks memory energy consumption over a sliding time window and determines available power budget for the next interval. The algorithm aims to deterministically maintain a power limit while maximizing memory bandwidth and performance.

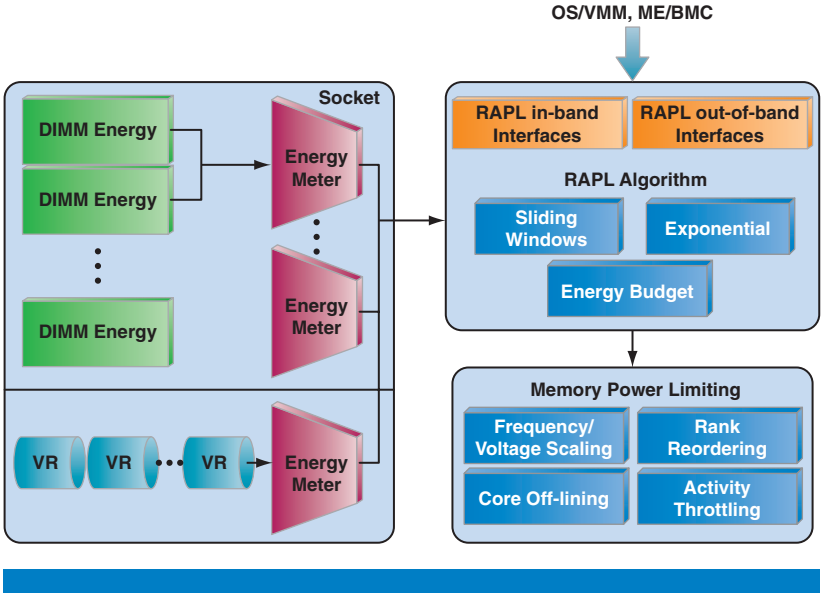


Figure 6: Memory RAPL architecture (Source: Intel Corporation, 2010)

"Multiple power limits may be active at any time, where each may be specified on a different component of memory and at different time scales."

Multiple power limits may be active at any time, where each may be specified on a different component of memory and at different time scales (for instance, power delivery versus component thermals versus battery life or data center power/thermals). For example, Figure 7 shows platform power budgeting policy may set a power limit of 50 W over 100 milliseconds to allow some burst of memory traffic while setting a lower component power limit of 75 W over a ten-second time window to control the long time window average power. RAPL technology provides the following benefits versus its predecessors and alternatives:

"Improved performance and correctness – Enforces mechanisms to maximize performance/responsiveness within any power limit, and guarantees correctness."

- *Improved performance and correctness* – Enforces mechanisms to maximize performance/responsiveness within any power limit, and guarantees correctness (deterministic power consumption limits) when required. This is primarily achieved by moving low-level policy into the core logic where we can take advantage of hardware granularity (temporal, spatial).
- *Decoupling* – Provides a key feature (memory power limiting) that fully decouples their external policy from our internal implementation. External agents no longer limit power consumption using explicit memory throttling registers, thereby freeing hardware to do this more intelligently than could be done externally and without exposing available capabilities externally.
- *Encapsulation* – Enhances capabilities and policies within our core logic without impacting or needing to (re) enable the ecosystem. Infers both time-to-market and differentiation opportunities.

- *Standardization* – Standard memory power limiting mechanism and interface across internal and external agents. We're extending the Intel ISA to include the notion of power limiting. RAPL exposes the time interval to software and platform making it dynamically configurable. Furthermore, it allows multiple limits to be set simultaneously to meet different thermal and power constraints that arise in real physical deployments. Setting the time interval statically and choosing the lowest common denominator can either cause power excursions or unnecessary performance degradation.

"RAPL exposes the time interval to software and platform making it dynamically configurable."

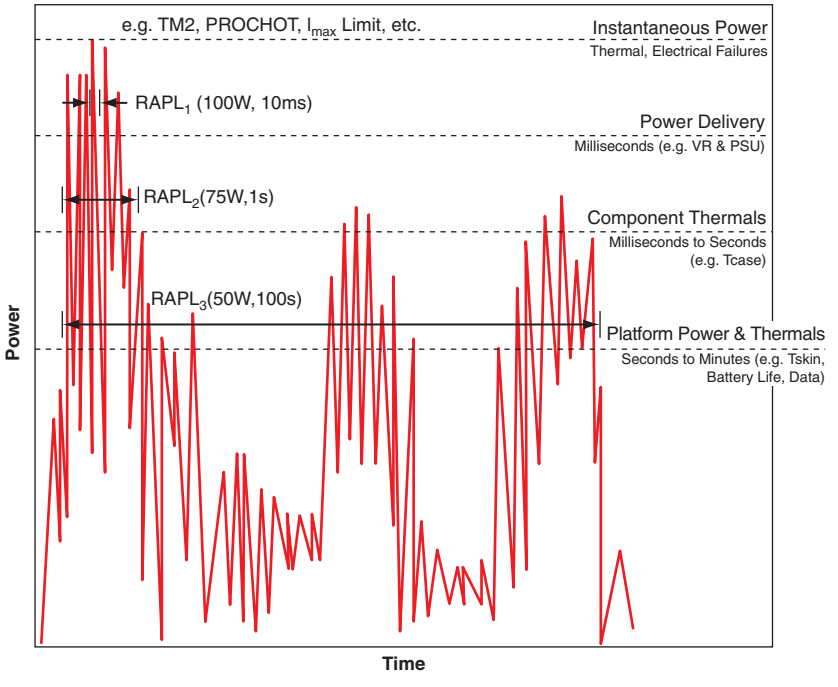


Figure 7: Open and closed loop thermal and power limiting usages (Source: Intel Corporation, 2010)

Memory Thermal Management and RAPL

For servers, memory power limiting with RAPL can be part of platform power budgeting for data center optimization such as Intel® Intelligent Power Node Manager[17] by limiting system power consumption so rack density may be optimized. RAPL may additionally be used in platforms to limit DIMM power as part of memory thermal management. For clients, memory RAPL can be used to enforce basic battery life policies, enhance thermal management[18], and pave the way to more advanced policies and efficient execution.

"For clients, memory RAPL can be used to enforce basic battery life policies, enhance thermal management."

Platforms generally support four interfaces (MSR, MMIO, PECI/PCS, and SCA) that allow both in-band and out-of-band programming of the RAPL DDR domain power limits. Memory RAPL limits can be set by many internal and external policy agents to limit memory power. Traditionally, it is the responsibility of policy agents to implement any advanced control algorithms

should there be desire and benefit. External in-band agents may include BIOS, OSPM, and OS-based agents[18]. External out-of-band agents may include PCH Management Engine, Baseboard Management Controller (BMC), Embedded Controller (EC), and System Management Controller (SMC).

The complex control interface means that RAPL must be able to support multiple limits at multiple timescales applied by multiple agents—as well as a policy to resolve potentially conflicting power limits. To accomplish this, memory RAPL algorithm operates as a policy on top of a set of memory power limiting mechanisms. It systematically determines the maximum available energy budget for different memory components using currently applied power limits, recent workload behavior and the measured/calculated power consumption. The CPU uses this available energy to limit memory power, changing interface speed (such as DRAM frequency) or restricting memory bandwidth (such as core off-lining and activity throttling).

"RAPL algorithm operates as a policy on top of a set of memory power limiting mechanisms."

"The CPU uses this available energy to limit memory power, changing interface speed or restricting memory bandwidth."

Domains exposed and managed by memory RAPL include socket (all memory attached to a processor for client) channel and DIMM. The processor/memory-controller should contain the necessary logic/firmware, controls, and interfaces (such as PCU, internal and external power sensors, and control registers) to implement the underlying algorithms and policies. The number of domains and power limits exposed in and out-of-band directly impact complexity and cost of the memory RAPL feature.

Memory Power Limiting

The domain-specific energy budget computed by the RAPL algorithm is passed to the memory power limiting logic. The power limiting logic is responsible for ensuring that the specified energy budget is not exceeded during the next time interval. The power limiting logic can use several mechanisms to limit memory power including rank reordering, frequency scaling, core off-lining and activity throttling. Each mechanism entails a different power-performance tradeoff for different workloads and their use should be defined within a policy that aims to maximize memory performance within specified energy constraints. The benefits of RAPL are illustrated in Figure 8 where RAPL allows for transient spikes in memory bandwidth at hardware time scales while maintaining average power over a time window specified by software. This allows the processor to deliver the required bandwidth to a workload when memory demand goes up and accumulate energy credits when demand is low or processor is idle. Setting a fixed bandwidth threshold to limit memory power would have a more adverse impact on system performance.

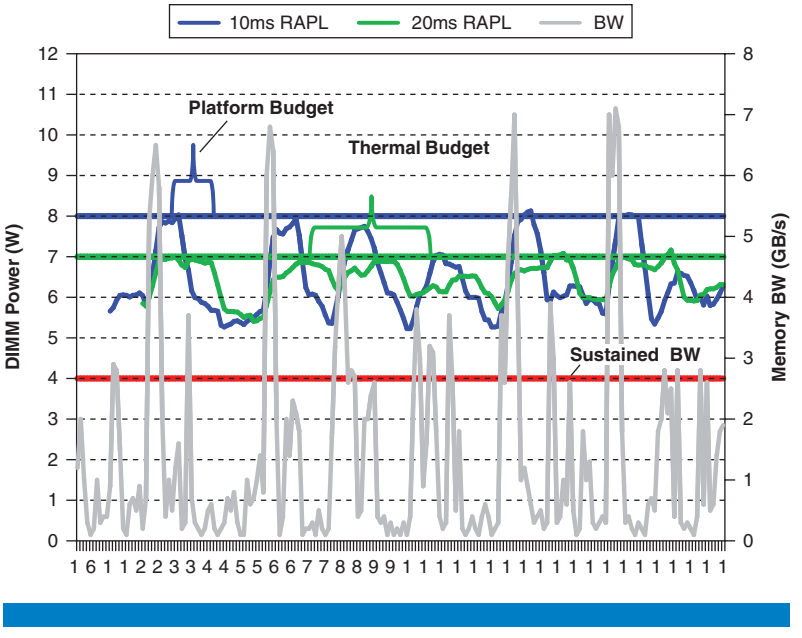


Figure 8: Memory RAPL power and thermal usage benefits (Source: Intel Corporation, 2012)

Standard Configuration Hardware Abstraction

Although sensor technology has matured over the last few years, it still lacks interoperability standards for representation, resource allocation, and constraint detection. Traditionally, most of the techniques used are specific to a given architecture, application, or devices. Sensors can have specific response characteristics that may be necessary elements in the quality of measurements. These characteristics may depend upon design aspects or operating conditions. For example, current measurements may exhibit high inaccuracies at lower utilization, but fewer errors at higher utilization. Hence, this information can be used to calibrate the sensor at higher utilization to reduce the overall error rate. Software support for sensor solutions has been proprietary for individual applications or device needs. The software designer is often faced with the task of re-implementing the sensor characteristics with new, emerging, or conflicting architectures. This has the potential to create software redundancy and to limit reuse. Therefore, it is essential to be able to interact and cooperate between autonomous entities in meaningful ways without too much complexity. This requires a comprehensive framework capable of measuring, quantifying, and describing the sensor's properties and its statistical behavior in a dynamic environment.

"It is essential to be able to interact and cooperate between autonomous entities in meaningful ways without too much complexity."

Common framework thermal and power autonomics defines the standardized set of registers required for performing power/thermal throttling functions. These registers are configurable resources in the platform that are mapped via PCIe[19] configuration space. Standard Configuration Architecture (SCA) is a novel methodology to utilize PCI vendor configuration space (VSEC IDs) to allow discoverability and feature standardization across platform segments and

generations. SCA provides a standard and consistent software-friendly interface mechanism for product features. Each feature or group of relevant features is co-located in one logical device, which then becomes easily configurable by the software. In a traditional approach, each new platform would redefine common features and how they are formatted in the configuration registers. The constant change reduces the sustaining cycle and increases the development effort greatly.

In general, SCA allows standardized discovery, organization, and consistency of layout that result in consistent implementation at the feature level while reducing platform software costs and enabling OS support. This section defines various abstraction classifications that employ common infrastructure mapped using the OEM-Standard PCIe capability structure. A common software framework deals with various abstractions with an ability to shape (or configure) the function based on various inputs. These abstractions perform the management functions required to successfully (a) identify the thermal and power capabilities, (b) securely upload/execute/modify functional/control parameters, (c) are signaled upon a trigger condition, and (d) configure the system parameters for execution containers, result storage, and scheduling. Architecturally, these abstractions can be classified into the following categories:

"A common software framework deals with various abstractions with an ability to shape the function based on various inputs."

- *Discovery* – These interfaces describe the attributes, granularity, and operational domain of operation. For example, the thermal threshold feature of a CPU can be controlled on an individual thread or collection of threads that are programmed individually.
- *Observation* – These interfaces provide the performance, power, and thermal statistics that constitute the feedback loop within the control loop.
- *Control* – These interfaces provide the ability to configure the operating environment for process control (like energy throttling and task throttling) as well as result collection. For example, power thresholds can be configured in such a manner that they trigger a preconfigured policy (such as throttling).
- *Status* – This interface describes the summary behavior of the domain. Status attributes are compressed to reduce redundant polling of the individual component in order to capture the statistics of the component that is behaving out of policy.
- *Interrupt* – This interface allows the configuration of the thresholds that define the trigger attributes of the system interrupts. Interrupts can be configured according to the usage. Interrupts can be classified as (a) System Management Interrupt (SMI), (b) Message Signaled Interrupts (MSI), (c) Out-Of-Band Interrupt Signal (for example: CPUHOT).

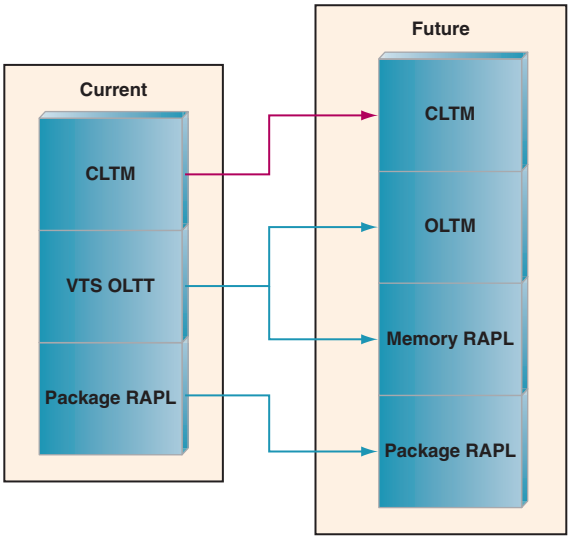


Figure 9: Evolution of thermal management and power capping
(Source: Intel Corporation, 2011)

Figure 9 illustrates a gradual evolution of memory thermal management features for CLTM, OLTM with virtual temperature sensor (VTS) and RAPL from current to future Intel architectures with underlying register format and location changed. The SCA block diagram depicted in Figure 10 follows the

conventions of PCIe capability structure headers[19] combined with Intel VSEC ID for thermal management bar pointing to MMIO (client) or CSR (server) space where functional registers for thermal and RAPL are exposed. SCA methodology aims to standardize and employ a scalable register definition for future expansion.

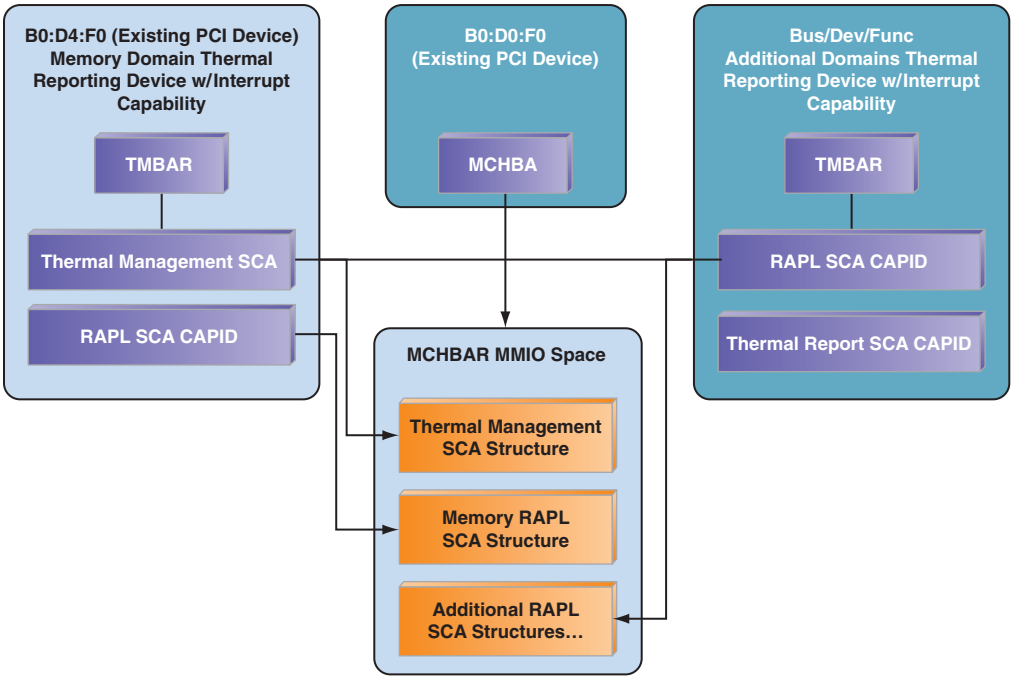


Figure 10: Standard configuration architecture for thermal management and RAPL (Source: Intel Corporation, 2011)

Thermal and Power Events

Event processing is accountable for processing an event cloud in an effort to establish a meaningful pattern, sequence of events, or a situation. It employs heuristics that relate temporal properties of events, correlation between events, event-driven processes, and so on. A computation-oriented event processing is responsible for runtime evaluation of a stream of data entering the system. For example, runtime computation of exponential averaged data in response to inbound events falls in this category. A detection-oriented event processing is responsible for identifying events patterns or situations. For example, identifying a workload pattern based on the distribution of burst in traffic falls in this category. An event is normally an asynchronous signal from hardware that indicates the need for attention to the subscriber of an event. A hardware event (interrupt) causes the processor to save its context and execute a registered interrupt handler. Thermal and power events reduce the amount of data flow over the communicating channels and are integral to the scalable architecture in the future data center. An application can set up signals with dynamic properties. A signal handler (or interrupt handler) can

"Event processing is accountable for processing an event cloud in an effort to establish a meaningful pattern, sequence of events, or a situation."

poll the data it is interested in for a short duration of time eliminating the need to continuously poll, reducing software overhead, and reducing hardware power consumption. A summary of programmable DDR thermal events are as follows:

- Thermal status change for policy-driven thresholds
- DDR refresh rate change to 2x
- Assertion of MEMHOT# input pin(s)
- Policy-free temperature thresholds (for active and passive cooling heuristics)

"Using event mechanisms, we can reduce the amount of data flow over the communicating channels."

Using event mechanisms, we can reduce the amount of data flow over the communicating channels. An application, based on its understanding of the actionable thresholds (sensor averages and so on), can set up signals with dynamic properties. These signals, when triggered, cause the software execution of a signal handler (or interrupt handler) that can poll the data it is interested in for a short duration of time. Upon understanding the cause of the alert, it can take an actuator action and optionally change the signal properties for the next trigger.

Various properties of signaling are:

- *Signal Type* represents the type of signal that needs to be propagated when a certain threshold policy is fulfilled. These types include System/Platform Management Interrupt (SMI/PMI), Non-Maskable Interrupt (NMI), Machine Check Interrupt (MCI), Message Signal Interrupt (MSI), SSP Interrupt (ARC, ME, and so on), and Out-Of-Band management Interrupt.
- *Signal Attributes* represent the attributes that define the complex (and measurable) threshold of a component whose address is defined by the device path. The fulfillment of the complex threshold crossing triggers an alert that is routed according to the Signal Type settings.
- *Signal Handle* represents the aggregation of multiple signals requested by various applications operating independently

"Since hardware supports only one set of threshold registers per device, it is therefore necessary to create a multiple instance model in a software middleware."

In many cases (particularly in Intel architecture), registers related to event mechanisms can be triggered based on thresholds and hysteresis. Mainly these triggers invoke system-level interrupts such as System Management Interrupt, SMBUS alerts, Machine Check Interrupt, and special signals (MEMHOT, PROCHOT, and so on). But since hardware supports only one set of threshold registers per device, it is therefore necessary to create a multiple instance model in a software middleware execution container that can handle multiple thresholds and selectively notify multiple applications. Hardware triggers can therefore be used intelligently to reduce the polling by the event service provider (ESP) execution container. The ESP handler gets triggered at the highest and lowest thresholds registered by multiple applications. As illustrated in Figure 11, once triggered, the handler compares each registered threshold and notifies the application if the threshold conditions are met. Additionally it evaluates the current thresholds and resets the triggers for future invocation.

A common framework proposed the following signals and interrupts allows ubiquitous management of power and thermal features:

"Signals and interrupts allows ubiquitous management of power and thermal features."

- *Signals – MEMHOT# (I/O)* – As an output this signal indicates that a memory throttling event is occurring. As an input this signal an external agent can force the memory controller to throttle in order to decrease memory power.
- *Interrupt (SMI)* – SMI is a legacy system interrupt that is broadcast to all the cores. It stalls the CPU (and OS) for a short duration, where the SMI interrupt handler can perform its service routing for any cooling functions. This interrupt is generated by Memory-Controller upon any thermal threshold crossing and causes the OS (or BIOS) to enter in SMM mode where an Interrupt Service Routine (ISR) decides upon taking any cooling action. These cooling actions extend from task throttling, identification of HOT channels (or uneven distribution) that can be rectified by OS dispatcher or memory allocator. Thermal thresholds related to SMI are reactivated upon crossing the hysteresis levels.
- *Interrupt (MSI)* – Message Signaled Interrupts (MSI) are software interrupts that allow the device to write a small amount of data to a special address in memory space. The chipset will deliver the corresponding interrupt to a CPU.
- *Interrupt (SCI)* – Hardware power management events trigger an OS-visible interrupt called a system control interrupt (SCI). Operating systems handle simple SCI interrupts (for example, fixed-feature power button state change) directly. Complex SCI interrupts are handled by the OS using AML code associated with the interrupt.

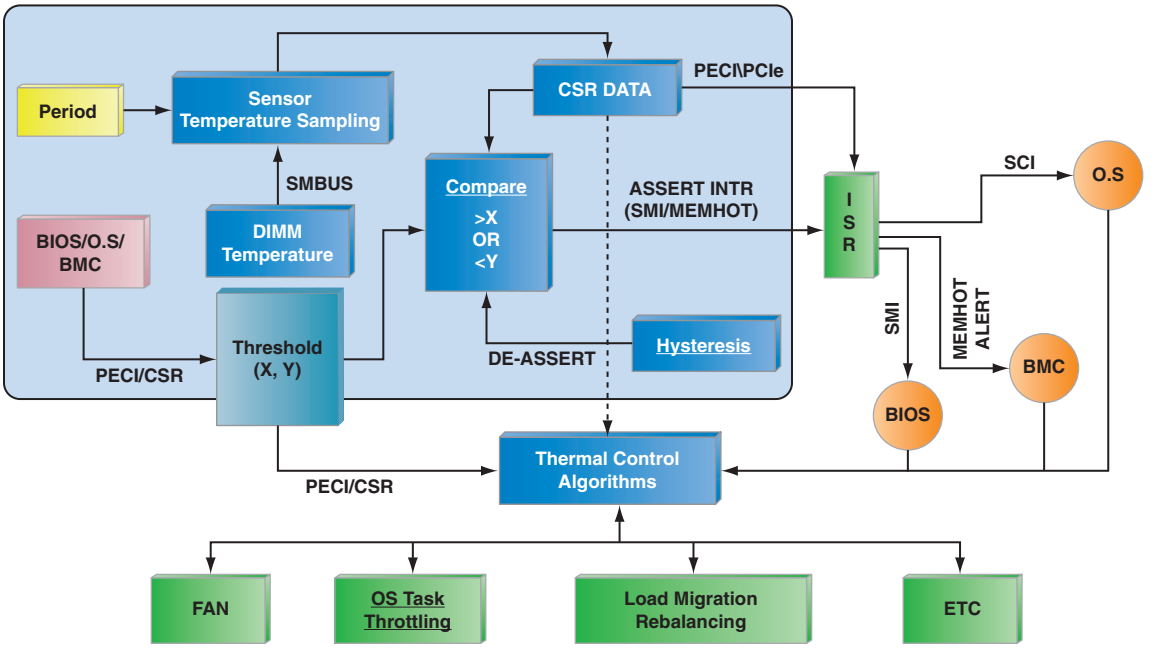


Figure 11: Interrupts and event signalings (Source: Intel Corporation, 2010)

Any thermal or power event can generate any one of these interrupts that allows the manageability to be hosted from any operational container.

Data Center Abstraction: Policy-based Thermal, Power, and QoS Management Using Optimal Scheduling

"Racks of server hardware, power supplies, network switches in a cloud data center generally operate with a set of business driven policies, intended to meet service level agreements."

"Self-managed system needs to be able to handle any changes in its environment with minimal human intervention."

Management policies are a set of rules used to manage distributed computing environments in a scalable manner. Racks of server hardware, power supplies, network switches in a cloud data center generally operate with a set of business driven policies, intended to meet service level agreements (SLAs), reduce maintenance costs, and drive greater efficiency in usage of compute resources and energy. All the managed elements must work in tandem, governed by business rules and management policies. Propagating these rules or policies, device by device, is not scalable. Since the number of servers per data center can run into tens of thousands and the number of customers using the cloud-hosted services on the hardware could run into hundreds of thousands, policy adherence via monitoring and management needs to be autonomous using software and tools[22][23]. To support the communication of data, management needs to be through standard protocols on all devices. This self-managed system needs to be able to handle any changes in its environment with minimal human intervention. The management system caters to conflicting goals of the end user who is interested in receiving the SLA and the cloud and data center provider who is interested in optimal usage of infrastructure. A non-policy management environment requires extensive monitoring, different software tools for different kinds of monitoring, and a coordination layer between them, which adds complexity, validation effort, and time for the ISVs. Hence cloud service providers (CSPs) tend to rank taking advantage of platform features low on their priority list.

Specifically, one of the cloud usages to address is the ability to map specific workloads to specific hardware to meet quality of service or throughput requirements.

Consider a very common Infrastructure energy efficiency policy and a SLA-based policy with a simple usage model:

Infrastructure Policy 1

IF (power consumed by rack1 >= 9.5 wkVA)
THEN Take action to maintain power by rack1

Action: Maintain power consumed by rack < 9.5 wKVA

This policy is implemented by infrastructure management software in the following manner:

```
MaintainPowerofRack()
{
GetpowerConsumedbyEachServer()
```

CheckLoadOnServer()
ActivatePowerPolicies()
PowerOffUnusedServer()
…..//more actions
}

Business Policy 2:

Premium Service SLA: XX IOPS per second, YY compute operations per second, DD IO bandwidth, ZZ Mbps memory bandwidth.

Policy 2 usage in resource pool creation for premium service:

IF ((serverX system compute operations per second >= YY)
AND (serverX system IOPS per second >= XX)
AND (server system memory bandwidth per second > ZZ))
THEN Include serverX in Premium Service pool

Policy 2 usage in resource workload scheduling enforced by a resource management component

IF (serverX memory bandwith < ZZ)
THEN Schedule workload on ServerY

Now consider how both Policy 1 and Policy 2 are used in a load balancing action:

If ((serverX system compute operations per second >= YY)
BUT (power consumed by rack1 >= 9.5 wkVA))
THEN LookforBestServerinList()//List of all servers fit for premium policy
IF (found)
THEN Move workload to next best server in Premium Service pool

As shown, Policy 1 and Policy 2 impact the optimal operation of racks and each server in the rack. They also impact the energy usage and effective allocation and usage of resource. The components affected by these policies are the cloud workload scheduler, resource usage monitoring tools, and the load balancer to maintain the performance SLAs and power usage. To locate the right platform (CPU, chipset and board), traverse the DC hierarchy to its location to set policies and aggregate the monitored data back to a meaningful high level metric. This section will introduce:

1. A data center resource monitoring framework
2. A basic template for a workload map
3. An autonomic control mechanism at every level to meet a defined efficiency metric

"To locate the right platform (CPU, chipset and board), traverse the DC hierarchy to its location to set policies and aggregate the monitored data back to a meaningful high level metric."

Data Center Group Resource Monitoring Framework

A typical data center hierarchy consists of compute nodes as rack mounted servers or aggregated as a subgroup of server blades with common chassis or SOC blades in a micro-server. A collection of nodes and node subgroups form either a physical group (rack) or span racks to form a logical group. A cloud workload scheduler working with a network zone will typically consist of such a collection of physical and logical groups, as shown in Figure 12.

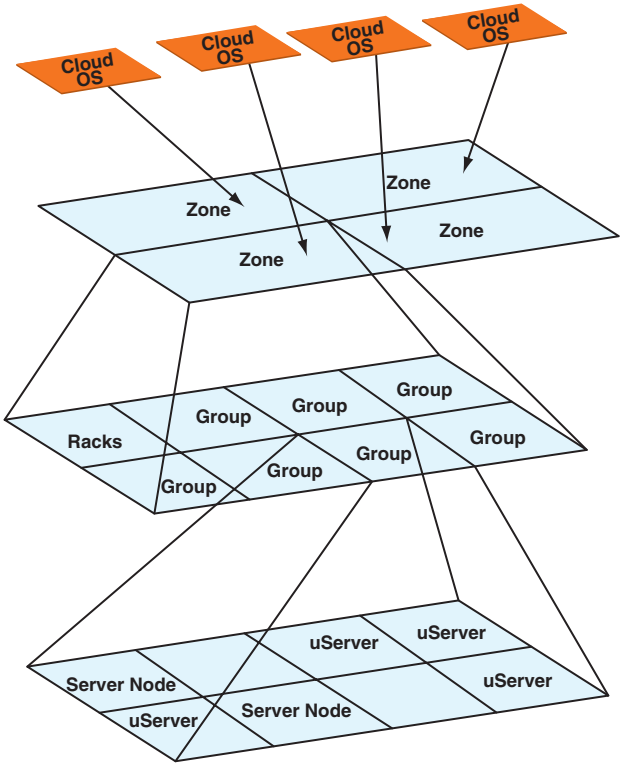


Figure 12: DC management hierarchy (Source: Intel Corporation, 2012)

"A scalable monitoring framework in a zone, group, node hierarchy would consist of monitoring at the node and the groups, thus extracting the required intelligence at each level."

A scalable monitoring framework in a zone, group, node hierarchy would consist of monitoring at the node and the groups, thus extracting the required intelligence at each level. At every level of hierarchy, the managed element would have specific ranks associated with it, which allows the zone level management function to get a quick snapshot of the status. For example, if the cloud OS scheduler requests a particular server with a particular QoS requirement, it would look at the availability rank and QoS rank for the zones it is managing (Figure 13). The zone's QoS rank will be derived using the ranks of the groups it is managing and that in turn from the nodes in the group. A higher rank could be termed favorable for this use case. This mechanism reduces the network span to address a system and also reduces the data set for search. Needless to say, data collection at the nodes and groups is required to be scalable and the data search distributed.

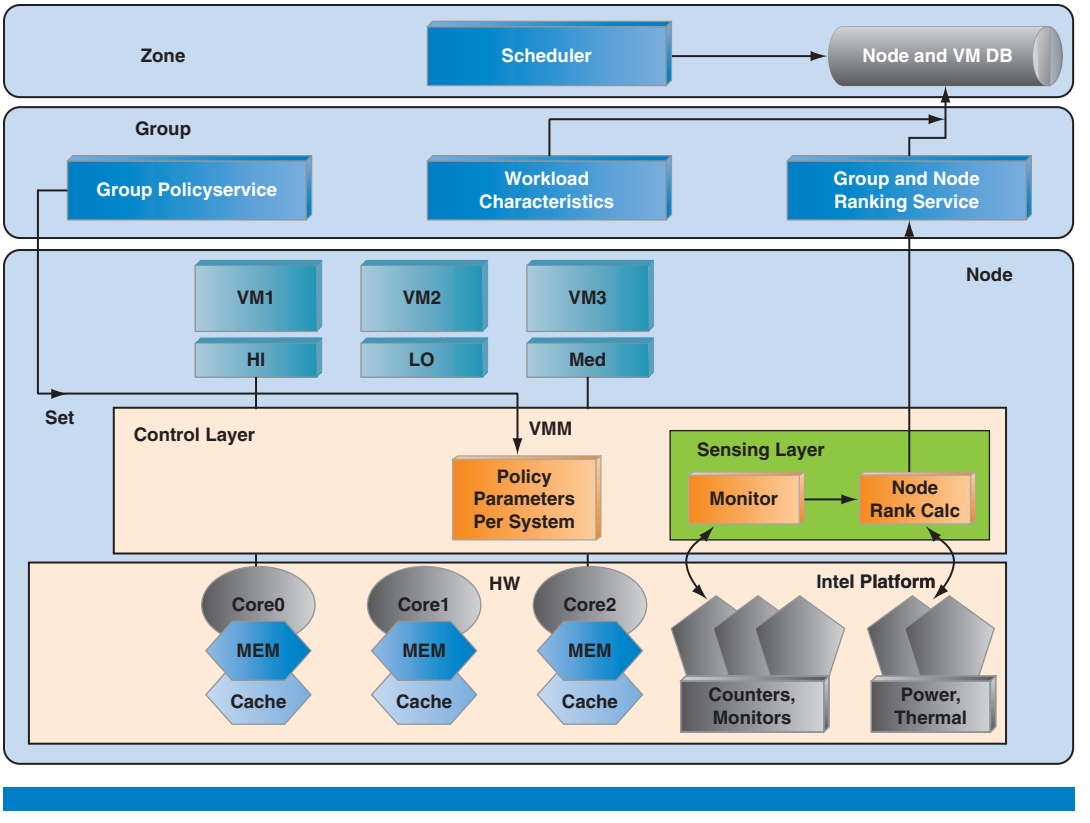


Figure 13: Monitoring and ranking
(Source: Intel Corporation, 2012)

Data Model at the Group

A hypertable[24] "tablet" for monitored data collected at the node and group will have data stored and collected in a format shown in Table 2.

Row Key	time	ID	IP address	Metrics	Rank
	T0	ComputeNode id =>"CN_UUID1", ServerGroupID => "SG_UUID1",	MCU_add = "10.255.255" BMC_Add = "10.255.255" VMM_Add = "10.255.255"	Usage => "100" MemBW => "80" Stalls => "40" Cycles => "100"	AvailabilityRank => "100" QOSRank => "200"
	T1	ComputeNode id => "CN_UUID2" ……	……	…..	…..

Table 2: Monitored Data Storage Format
(Source: Intel Corporation, 2012)

Workload Map

A significant amount of work has been conducted on analyzing workload characteristics, user behavior, and performance of workloads in single- and multisystem virtualized and nonvirtualized environments[28][25] (see Code 1). User behavior and phases of workload execution, whether submitted as

pre-known inputs or learned by correlation during execution have been discussed [26][27].

```
{“user session”: {
        “jobs”: [{job:1},{job:2} ],
        “Instruction Sequence”: [ ],
“time”: [ ],
“machine_instruction_type”:[{mem_read:1},{mem_write:2},{ARM:3}],
“Phases”:[{phase1:“mem_read”},{phase2:”mem_write”}]
}}
{ “UsagePattern”:{
        “CPUUsagePattern”: [ ],
        “MemoryUsagePattern”: [ ] ,
        “NetworkIOUsagePattern”: [ ],
        “DiskIOUsagePattern”: [ ]
}}
```

Code 1: An example workload map in JavaScript Object Notation (JSON) (Source: Intel Corporation, 2012)

“If workload runtimes and throughput are the cost functions to optimize, then a workload autonomics manager sets those parameters for providing an efficiency rank.”

“Node level control and throttling will also be needed where in usage of shared resources like power, cache, and bandwidth can be set as thresholds and throttled.”

Autonomic Control Mechanism for Workload Placement

Autonomic workload placement in the data center has often been done to optimize on cost, energy usage, or SLA[29][30]. The scalable and hierarchical monitoring framework can be used to perform optimal workload placement as shown in Figure 14. If workload runtimes and throughput are the cost functions to optimize, then a workload autonomics manager sets those parameters for providing an efficiency rank. The workload map and the placement policies are set at the zone, group, and the compute node. As workloads are placed, the efficiency ranks of the nodes are updated based on monitored data. The “control” block in this flow may attempt to adjust the workloads within a group by migrating to different systems, to maintain the efficiency rank of the group. Node level control and throttling will also be needed where in usage of shared resources like power, cache, and bandwidth can be set as thresholds and throttled to maintain a node level efficiency rank, thus reducing the cost of migration.

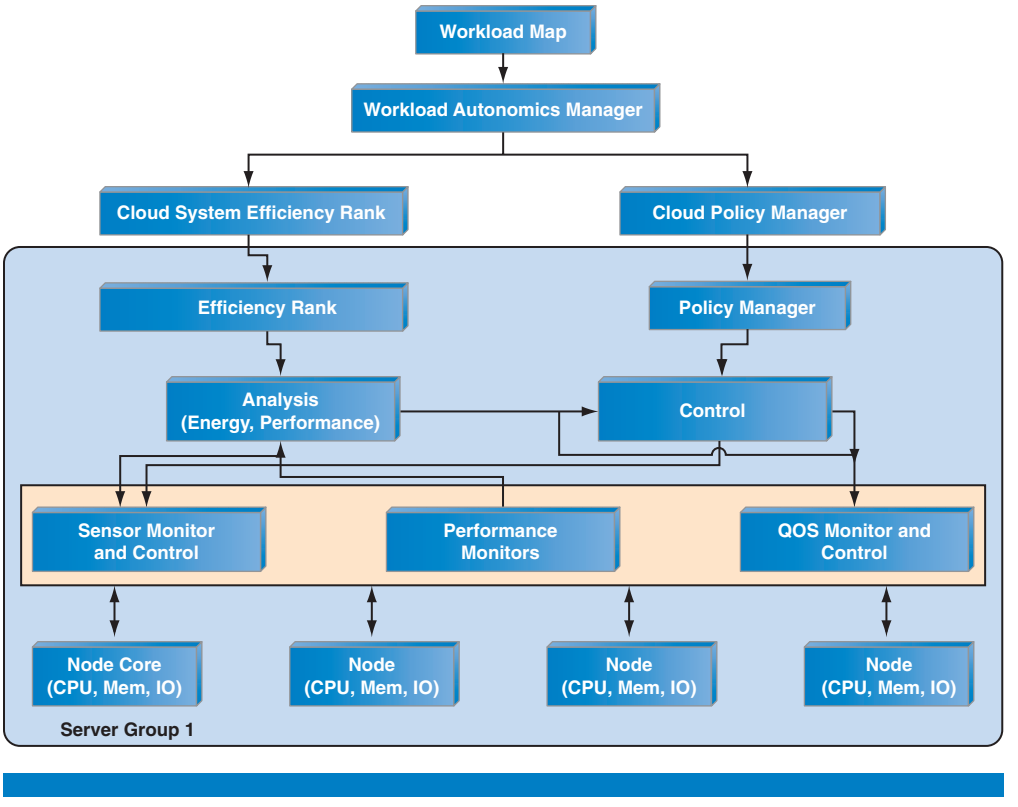


Figure 14: Design of autonomic workload scheduling
(Source: Intel Corporation, 2012)

Data Center Trends and Technologies

To take full advantage of platform features requires an understanding of future data center needs and possible interactions between the IT equipment and the data center. Some of these interactions are easily understood and managed while others require a high degree of communication and real-time optimization between the ITE and data center. A holistic view will become the norm when evaluating data center efficiency.

"Platform features requires an understanding of future data center needs and possible interactions between the IT equipment and the data center."

Presently, the data center and the ITE are viewed separately and there is no agreed-upon way to measure overall efficiency. The data center can be evaluated using power usage effectiveness (PUE), which is simply the ratio of total energy entering the data center to the ITE load. The Green Grid has defined how PUE should be stated based on how the measurements are performed. PUE provides a good way to evaluate and compare infrastructure designs but does nothing to address the efficiency of the ITE (the "1" in the PUE equation). Separately, ITE compute efficiency metrics have been under evaluation by the governmental entities such as the US government's Energy Star to characterize the efficiency using both power and performance so that servers can be directly compared with each other.

"The data center can be evaluated using power usage effectiveness (PUE), which is simply the ratio of total energy entering the data center to the ITE load."

In the future, collaborative design between the data center and the ITE must occur. The data center and ITE must be fully aware of and communicative

"An integrated data center will have the software and algorithms using the data supplied by the ITE to optimally address the real-time workloads."

with each other in order for real-time optimization to occur. That collaboration will result in features and capabilities in the ITE that are available and usable when needed to optimally meet the data center's needs and vice versa. It is not enough to have the features. An integrated data center will have the software and algorithms using the data supplied by the ITE to optimally address the real-time workloads.

Some of the most important thermal interactions requiring data center/ITE collaboration and optimization are:

1. Server inlet temperature – Servers are designed to support specific environments based on the temperature delivered at the inlet to the server. When the data center is aware of inlet temperature (as delivered by ITE sensors), cooling adjustments can be made to the room to ensure compliance to the ITE specification and prevent reduced performance or shutdown. Alternatively, workload can be moved to other servers not under this type of stress.
2. Rack airflow demand – Airflow demanded by a server varies based on workload and thermal conditions. (ITE thermal management drives server fans to ensure compliance to component thermal specifications.) The data center must be able to satisfy that demand or cooling may be compromised. The result may be airflow recirculation in the data center leading to further increases in IT airflow demands to ensure adequate cooling to the server. Optimized, air-cooled data centers will enable delivery of precisely the airflow required to cool the ITE.
3. Rack exhaust temperature – The air exhausting from a rack can be both a safety and functional concern. If employees will be working in this space, temperatures can easily approach burn limits, or may simply be too hot for humans. Air movement can be increased to eliminate this concern. Also, some equipment including cables and switches have temperature limits that can be exceeded. Inclusion of exhaust temperature in the server's thermal management can alert the data centers of potential issues. More importantly, awareness at a rack or data center level can create opportunity for the data center to respond by improving flow to that rack or through better distribution of the workload.

"Awareness at a rack or data center level can create opportunity for the data center to respond by improving flow to that rack."

4. Thermally limited performance – When increased cooling within the server is inadequate to meet thermal requirements power management features such as throttling may engage to ensure thermal compliance. By knowing whether power management features are engaging and how often, determination can be made whether to redistribute workload to less-stressed servers to better meet the workload demands of the data center. In some cases it may be desirable to keep workload on thermally-stressed systems for overall power reduction. (Running many servers under lower load may be much more power-consuming than running fewer servers under heavy loads.)

The most important power/performance interactions requiring data center/ITE optimization are:

1. Power supply capability – Power supplies are not typically designed to support simultaneous worst-case consumption on all components. When workloads approach the limits of the power supply to support them the data center should be aware so that workload distribution can take place assuming that is the desired response by the data center operator.
2. Power consumption – Similar to power supply budget, the capability for delivering power in the rack or data center can be limited. The ability to know power consumption at a rack level enables the data center operator to redistribute workload prior to exceeding the capability for the data center to deliver the power for that workload.
3. Power-thermal-aware scheduling – The holy grail of power/performance/thermal management is the capability for scheduling workloads based on awareness of how to complete the workload while consuming the least power in the required time. Thermal, power, and performance characteristics all weigh in the algorithms required to determine this. Without the previously described sensor capability power-thermal-aware scheduling would be impossible. Each power or thermal management feature in concert with the sensors that support them plays its part in achieving an optimal data center–ITE capability. The data center characteristics must be combined with the ITE characteristics to be able to begin the process of optimization. The capability for scheduling, adjusting, or moving workloads based upon the power and thermal capability of the data center will distinguish future data centers from present implementations where there is little or no awareness between the ITE and the data center.

Conclusion

High power and density pose significant cooling challenges for system design as well as for the facility housing the equipment. Designing a cooling solution to manage temperature of these high power chips in a server is critical to reliable performance and life of the equipment. This requires a well-designed autonomous thermal management implementation that can enable minimal thermal guard bands and the flexibility to configure the platform for performance and power reduction. Dynamic thermal management presents a fast growing approach that couples thermal management and explicit management of energy consumed to optimize energy efficiency of the chip. The challenge in facilitating a holistic approach requires defining thermal and power telemetry with control mechanisms that are ubiquitous and hierarchical within a data center. In this article we described the system approach to building a standardized *Memory* power and thermal management infrastructure. We described the RAPL methodology that enforces power limits over a time widow, while reducing performance impact in highly dynamic and transient data center workloads. RAPL scheme dynamically determines the maximum available energy budget for different memory components using currently applied power limits, recent workload behavior, and the measured/calculated power

"High power and density pose significant cooling challenges for system design as well as for the facility housing the equipment."

"Dynamic thermal management presents a fast growing approach that couples thermal management and explicit management of energy consumed to optimize energy efficiency of the chip."

consumption. We also presented Server Dynamic CLTM architecture that integrates thermal management within the processor and integrated memory controller (IMC). We also reiterated the need for interface standardization through standard configuration architecture. It allows standardized discovery, organization, and consistency of layout that result in consistent implementation at feature level while reducing platform software costs and enabling OS support. The standardized approach also includes the ability to reduce the software polling overhead by using standard events and interrupts. Event processing is accountable for processing an event cloud in an effort to establish a meaningful pattern, sequence of events, or a situation. It employs heuristics that relate temporal properties of events,correlation between events, event-driven processes, and so on. We illustrated the management models to drive the decisions for optimal thermal management in a platform in the presence of acoustics, environmental standards, power, and performance targets.

"The standardized approach also includes the ability to reduce the software polling overhead by using standard events and interrupts."

References

[1] J. G. Koomey, "Estimating Total Power Consumption by Servers in the U.S. and the World"; http://enterprise.amd.com/Downloads/svrpwrusecompletefinal.pdf.

[2] L. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, Jan 2007.

[3] U.S. EPA, "Report to congress on server and data center energy efficiency," *Tech. Report*, Aug. 2007.

[4] Intel® Core™ i7 Processor Family for the LGA2011-0 Socket Thermal / Mechanical Specification and Design Guide. Document number 326199-001.

[5] ACPI Specification Revision 5.0, http://.www.acpi.info/spec

[6] DCMI – Data Center Manageability Interface Specification v1.0, Revision 1.0, May 1, 2008, http://www.intel.com/go/dcmi

[7] Shah, A. J., Carey, V. P., Bash, C. E., Patel, C. D., 2003 (submitted), "Exergy Analysis of Data Center Thermal Management Systems," *IMECE 2003–42527, 2003 International Mechanical Engineering Congress and Exposition*, Washington, DC.

[8] R. Sharma, C. E. Bash, C. D. Patel, R. S. Friedrich, and J. Chase, "Balance of power: Dynamic thermal management for internet data centers," *Hewlett-Packard Laboratories Technical Report*: HPL-2003-5.

[9] Patel, C. D., Sharma, R. K, Bash, C. E., Beitelmal, A, "Thermal Considerations in Cooling Large Scale High Compute Density Data Centers," *ITherm 2002 – Eighth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, May 2002, San Diego, California.

[10] N. Ahuja, G. Govindaraju, T. Traush, C. Rego, S. Ahuja, B. Carn, "Energy Savings through High Ambient Data Center Operations," DTTC, 2010.

[11] R. Khanna, M. Kumar, K. Li, J. Tang, C. Le, "Memory Containerization for Dynamic Power Optimization," DTTC, 2007.

[12] "Data Center Energy Characterization Study," Pacific Gas and Electric, California, USA, Feb 2001.

[13] R. Khanna, R. Steinbrecher, F. Lopez, K. Cheng, C. Lel, "Dynamic Closed Loop Memory Throttling to Optimize Power and Performance," *DTTC*, 2007.

[14] Rahul Khanna, Mohan J Kumar, Kevin Y Li, James Tang, Christian Le, "Memory Containerization for Dynamic Power Optimization," *DTTC*, 2007.

[15] H. David, E. Gorbatov, U. Hannebute, R. Khanna, C. Le, "RAPL: Memory Power Estimation and Capping," *ISLPD*, 2010.

[16] J. Lin, H. Zheng, Z. Zhu, E. Gorbatov, H. David, and Z. Zhang, "Software thermal management of DRAM memory for multicore systems," *In Proc. of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 337–348, 2008.

[17] Intel, "Intel Intelligent Power Node Manager 2.0 External Interface Specification Using IPMI," CDI #434090, 2010.

[18] Intel, "Intel Dynamic Power Performance Management (DPPM)," CDI #405840, 2008.

[19] PCIe Base 3.0 Specification, http://www.pcisig.comspecificationspciexpressbase3

[20] http://www.lbl.gov/

[21] http://thegreengrid.org

[22] DC Robinson and Morris Sloman, "Domains: a new approach to distributed system management," Proceedings, Workshop on the Future, Trends of Distributed Computing Systems in the 1990s, 1988., pages 154–163. ieeexplore.ieee.org, 14–16 Sep 1988.

[23] Hanson, James E.; Whalley, Ian; Steinder, Malgorzata; Kephart, Jeffrey O., "Multi-Aspect Hardware Management in Enterprise Server Consolidation," IEEE, 2010.

[24] Dionysios Logothetis, Kenneth Yocum, "*Data Indexing for Stateful, Large-scale Data Processing*," 5th international workshop on networking meets databases (NetDB 2009). SIGOPS Oper. Syst. Rev. 43, 4 (Jan. 2010)

[25] Mohamed A. El-Refaey; Dr. Mohamed Abu Rizkaa, "Virtual Systems Workload Characterization," 18th IEEE International Workshops on Enabling Technologies, 2009.

[26] Daniel Gmach; Jerry Rolia and Ludmila Cherkasova; Alfons Kemper, "Workload Analysis and Demand Prediction of Enterprise Data Center Applications, "In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization* (IISWC '07).

[27] Helmut Hlavacs, Ewald Hotop, Gabriele Kotsis; "Workload Generation by Modeling User Behavior," http://www.bisante.org

[28] Maria Calzarossa, Giuseppe Serazzi, "Workload Characterization–A Survey," Italian Research Council C.N.R Progetto Finalizzato SistemiInformatici e Calcolo Parallelo

[29] Josep Ll. Berral, Ricard Gavaldà, Jordi Torres, "Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning," Universitat Politècnica de Catalunya and Barcelona Supercomputing Center

[30] Norman W. Paton, Marcelo A. T. de Aragão, Kevin Lee, Alvaro A. A. Fernandes, Rizos Sakellariou, "Optimizing Utility in Cloud Computing through Autonomic Workload Execution," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2009.

[31] R. Khanna, M. Kumar. *A Vision for Platform Autonomy*. Intel Press, 2011. http://intel.com/intelpress.

Author Biographies

Christian Le is a server power and thermal architect in Intel's Data Center and Connected Systems Group. He has spent 16 years designing system thermal and power management solutions. His current focus is on data center power optimization and platform autonomics research.

Robin Steinbrecher is a server thermal architect in Intel's Data Center and Connected Systems Group. He is responsible for cooling architecture for server products including thermal management, cooling capability, and power/thermal optimization. He has developed silicon- and system-based methods enabling optimal thermal control in servers. Robin has over twenty years of experience in electronics cooling technologies at Intel and IBM, and now focuses on integration of these technologies in data center applications.

Rahul Khanna is a platform architect at Intel Corporation involved in development of energy efficient algorithms. Over the past 17 years he has worked on server system software technologies including platform automation, power/thermal optimization techniques, reliability, optimization, and predictive methodologies. He has authored several technical papers and book chapters in the areas related to energy optimization, platform

wireless interconnects, sensor networks, interconnect reliability, predictive modeling, motion estimation, and security, and holds 27 patents. He is also the co-inventor of the Intel IBIST methodology for high-speed interconnect testing. His research interests include machine learning based power/thermal optimization algorithms, narrow-channel high-speed wireless interconnects, and information retrieval in dense sensor networks. Rahul is member of IEEE and the recipient of three Intel Achievement Awards for his contributions in areas related to advancements of platform technologies. He is the author of the book *A Vision for Platform Autonomy: Robust Frameworks for Systems*. Rahul Khanna can be reached at rahul.khanna@intel.com

Mrittika Ganguli is a Data Center Management Software Architect at Intel's Data Center and Connected Systems Group. She has over 16 years of experience in software development, management, and architecture roles. Her technical strengths and contributions are in server hardware management, system software, and energy management software.

FUZZY LOGIC: ADAPTIVE FAN SPEED CONTROL METHODOLOGY

Contributor

Rafael de la Guardia
Intel Labs, Guadalajara

This article studies an adaptive fan speed control technology to deliver scalable acoustic control that is integrated in platforms. The technology consists of two critical elements: algorithms that automatically adapt to system configuration and loading, and direct coupling between acoustic fan speed control techniques and power management.

The automatic tuning eliminates the need for manual tuning while reducing the guard bands that are introduced by current static algorithms to account for system-to-system variations. The automatic tuning capability combined with the auto-discoverable thermal management capability enables a scalable solution for optimal cooling. The objective of the fan controller is then to maintain a positive headroom for each device at all times.

Introduction

"A fan controller integrated in the platform to set the fans to turn at low speeds when thermal conditions allow."

Modern computer platforms can have several fans as part of their cooling solutions. Cooling requirements in a platform vary continuously with time. Hence it is advantageous to have a fan controller integrated in the platform to set the fans to turn at low speeds when thermal conditions allow. The controller can use either a feed-forward (FF) or feedback (FB) scheme to generate a pulse width modulated (PWM) voltage signal to set the speed of each fan. Typical FF controllers[1] apply linear interpolation between programmable low and high fan speed limits to compute the PWM level as a function of the temperature input. FB controllers[2] work on a thermal error signal defined as the difference between the measured temperature input and a set point temperature. The controller attempts to drive the thermal error to zero by adjusting the fan speed, for example, using a PID control law. The increased complexity of FB controllers relative to traditional FF controllers is justified because they can have a significant performance advantage in terms or reducing thermal guard bands and providing a smoother response. Attaining this benefit using traditional FB control methods is complicated in practice because:

"Conservative tuning of the parameters and testing by the system integrator are required to ensure reliable performance."

- The thermal relationship between the fan speed input and the temperature sensor output is nonlinear
- The fans themselves feature static nonlinearities; for example, the speed is bounded by upper and lower limits
- The response of the fans is slow relative to the thermal error dynamics. The result is that conservative tuning of the parameters and testing by the system integrator are required to ensure reliable performance.

In the case of servers in particular, the goal of fan speed control is to improve performance and acoustics while making the overall systems more power efficient without changing thermal requirements, altering server system reliability, or negatively impacting performance[3]. From the point of view of the thermal solution, server systems consist of cooling zones, at least some of which may contain one or more cooling fans. Most modern dual- and multiprocessor servers have from three to five fan domains controlled by independent fan speed signals. During normal operation, as opposed to during failure situations, fan speeds are driven by the (minimum) thermal margin in each domain. Hence, the control system monitors ambient and/or component temperatures and applies algorithms to reduce thermal margins without increasing thermal risk. A combination of lookup tables and feedback control algorithms are typically used[4].

"Modern dual- and multiprocessor servers have from three to five fan domains controlled by independent fan speed signals."

"The control system monitors ambient and/or component temperatures and applies algorithms to reduce thermal margins without increasing thermal risk."

In this article, we propose a solution to enhance the energy efficiency of servers by reclaiming the thermal margins of the fans. We present a new fan speed control system which manages the speed of the fans proactively by monitoring both the usage indicators and the temperatures of the components. It employs thermal models based on fuzzy logic to minimize the energy consumed by the fans without violating the thermal constraints of the system. It achieves this at equal performance and lower acoustic noise. In addition, the system is adaptive and self-tuning, thereby eliminating the need for cumbersome manual tuning. Thus, the cooling solution is automatically optimized for each individual server system. The remainder of this article is organized as follows. In the section "Adaptive Fuzzy Models" we describe a type of fuzzy systems called Takagi-Sugeno that can be used to model the thermal interactions in a server system. In the section "Model-based Predictive Control" we describe a model-predictive control technique that uses a fuzzy thermal model to minimize the total power in a system while maintaining positive thermal headroom. In the section "Self-Tuning, Adaptive Fan Speed Control" we present some simulation results that demonstrate the benefit of the system. The "Conclusions" section makes some final observations.

Adaptive Fuzzy Models

In this section we describe a type of fuzzy system that can be used to create and update models which describe the complex, possibly nonlinear relationships between components found inside computer systems. These models are data-driven in the sense that they can be fully designed, extracted, or learned from data, which can be gathered from different sources: raw data from measurements; context information, such as OS hints, applications, and user input and environment; structured sources, such as technical specifications; and streams, typically from an on-line processing context (sample- or block-wise). Several methodologies can be used for model training, such as statistical approaches, machine learning, and iterative least-squares techniques. The result of training is the determination of optimal

parameter settings for the system. An important class of data-driven models is comprised of evolving systems, which are automatically adapted, extended and evolved dynamically on the fly based on new incoming data samples[5]. Compared to adaptive models, which include mechanisms to update some model parameters, evolving systems can also be extended based on data; that is, they can generate new structural components as needed in order to improve accuracy. Hence, evolving models are a key element to enable self-learning computer systems and machines.

"Takagi-Sugeno (TS) fuzzy systems are widely used to build adaptive fuzzy systems."

"Like other types of fuzzy systems, TS systems include a set of rules of the general form IF antecedent *THEN* consequent*."*

Takagi-Sugeno Fuzzy Systems

Takagi-Sugeno (TS) fuzzy systems are widely used to build adaptive fuzzy systems. Figure 1 presents the main elements of a TS system. Like other types of fuzzy systems, TS systems include a set of rules of the general form IF *antecedent* THEN *consequent*. The characteristic feature of TS systems is the linear consequent functions, which are combined by the nonlinear fuzzy sets and T-norm operators in the antecedent part of the rules to form a smooth nonlinear model[5].

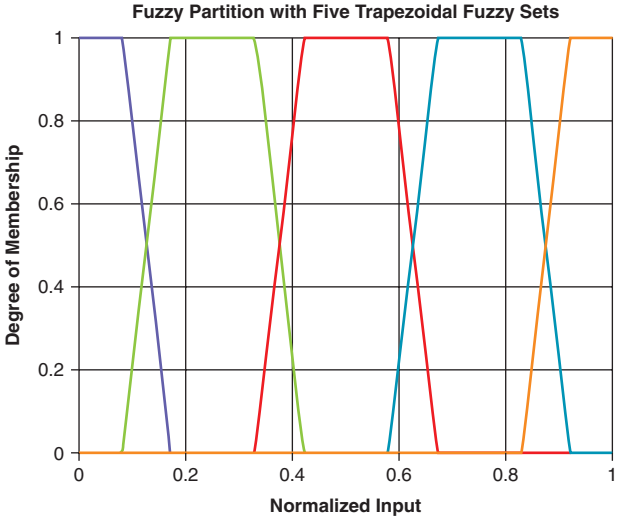
- The *i*-th rule of a TS system is of the form
  R*i* : IF z_1 is Z_{i1} AND . . . AND z_p is Z_{ip} THEN
  $$y_i = \varphi_{i0} + \sum_{j=1}^{P} \varphi_{ij} z_j$$
- Rules are combined by fuzzy inference
  $$\hat{f}(\mathbf{z}) = \hat{y} = \sum_{i=1}^{c} \bar{w}_i(\mathbf{z}) y_i$$
- Rules' membership degree obtained by applying t-norm to antecedent part
  $$w_i(\mathbf{z}) = \top_{j=1}^{p} Z_{ij}(z_j)$$
- Rule weights are normalized
  $$\bar{w}_i(\mathbf{z}) = \frac{w_i(\mathbf{z})}{\sum_{j=1}^{C} w_j(\mathbf{z})}$$

Figure 1: Elements of a Takagi-Sugeno fuzzy system (Source: Intel Corporation, 2012)

Figure 2 shows two of the most common types of fuzzy sets used in the construction of TS systems. Trapezoidal functions are simple to compute and interpret but are not steady differentiable and may not cover the input space sufficiently in the case of data-driven systems. On the other hand, Gaussian functions have infinite support and are steady differentiable but their interpretability is weaker.

- Trapezoidal fuzzy sets

$$Z_{ij}(z_j) = \begin{cases} \frac{z_j - a_{ij}}{b_{ij} - a_{ij}} & \text{if } a_{ij} < z_j < b_{ij} \\ 1 & \text{if } a_{ij} < z_j < b_{ij} \\ \frac{d_{ij} - z_j}{d_{ij} - c_{ij}} & \text{if } a_{ij} < z_j < b_{ij} \\ 0 & \text{otherwise} \end{cases}$$



- Gaussian fuzzy sets

$$Z_{ij}(z_j) = e^{\frac{-(z_j - c_{ij})^2}{2\sigma_{ij}^2}}$$

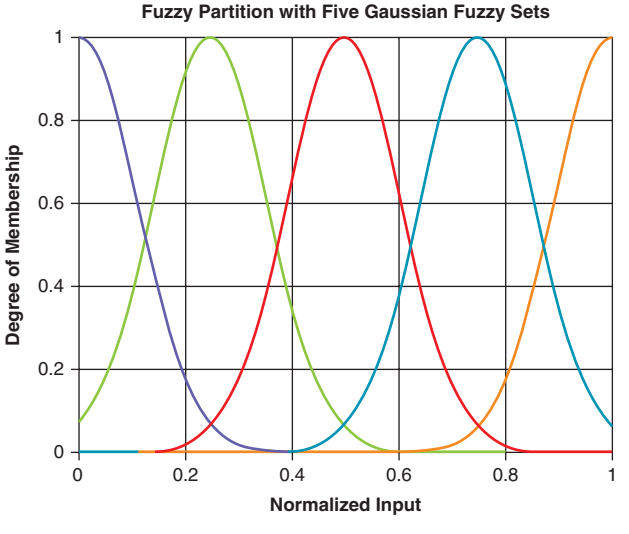


Figure 2: Trapezoidal and Gaussian fuzzy sets (Source: Intel Corporation, 2012)

To illustrate how to use a TS system we present a simple example. Figure 3 shows the rules and fuzzy sets for a simple system with two fans and one temperature sensor. The inputs to the system are the scheduling variables z_1 and z_2 that represent the speed setting for each fan. The inputs are normalized

"The inputs to the system are the scheduling variables z_1 and z_2 that represent the speed setting for each fan."

so that they take values in the range between 0 (minimum speed) and 1 (maximum speed). The input space for each fan speed is partitioned into two fuzzy values, Slow1 (respectively Slow2) and Fast1 (respectively Fast2).

R1 : IF z_1 is Slow1 AND z_2 is Slow2 THEN $y_1 = \varphi_{10}$
R2 : IF z_1 is Slow1 AND z_2 is Fast2 THEN $y_2 = \varphi_{20}$
R3 : IF z_1 is Fast1 AND z_2 is Slow2 THEN $y_3 = \varphi_{30}$
R4 : IF z_1 is Fast1 AND z_2 is Fast2 THEN $y_4 = \varphi_{40}$

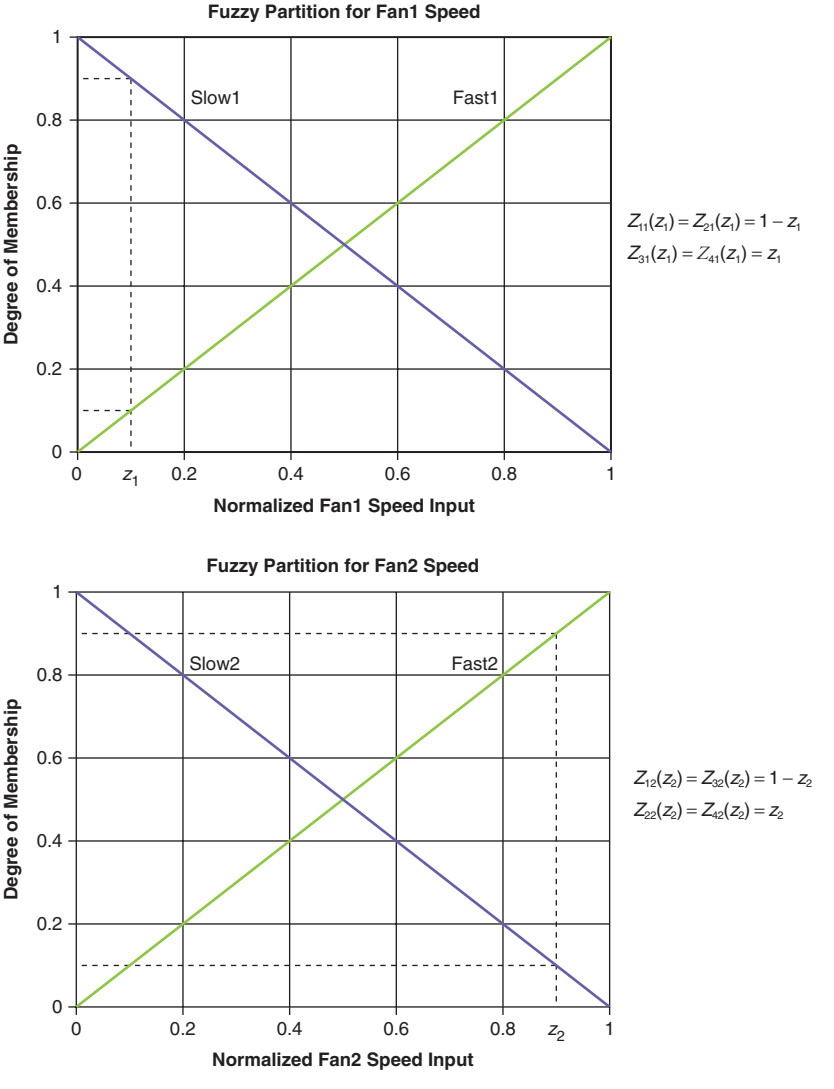


Figure 3: Elements of a Takagi-Sugeno fuzzy system representing a thermal model
(Source: Intel Corporation, 2012)

"For simplicity, the consequents for the rules include only the singleton consequent parameter."

For simplicity, the consequents for the rules include only the singleton consequent parameter, φ_{i0}. Using multiplication in place of the T-norm for the conjunction operator, the weight of each rule can be computed as follows.

$$w_1 = Z_{11}(z_1)Z_{12}(z_2) = (1 - z_1)(1 - z_2) \quad (1)$$

$$w_2 = Z_{21}(z_1)Z_{22}(z_2) = (1 - z_1)z_2 \quad (2)$$

$$w_3 = Z_{31}(z_1)Z_{32}(z_2) = z_1(1 - z_2) \quad (3)$$

$$w_4 = Z_{41}(z_1)Z_{42}(z_2) = z_1 z_2 \quad (4)$$

For example, suppose that the normalized speeds are $z_1 = 0.1$ and $z_2 = 0.9$, as shown in the figure. Then, using equations 1–4 the value of the rule weights correspond to $w_1 = 0.09$, $w_2 = 0.81$, $w_3 = 0.01$, and $w_4 = 0.09$. Therefore, the output of the fuzzy model, which in this case corresponds to a predicted temperature, is given by

$$\hat{f}(z_1, z_2) = \hat{y} = 0.09\ \varphi_{10} + 0.81\ \varphi_{20} + 0.01\ \varphi_{30} + 0.01\ \varphi_{40} \quad (5)$$

We'll have more to say about the parameters φ_{i0} in the next section.

Model-based Predictive Control

The fan speed control problem can be divided in two tasks. The first task is to construct a Takagi-Sugeno fuzzy system that can be learned from data, with fan speeds (or fan voltage or PWM) as inputs and device temperatures as outputs. The second task is to design an optimum controller that will exploit the predictive capabilities of the TS system to drive the fan speeds toward optimal settings that will minimize a cost function while limiting the temperatures in the system below the desired limits. The two tasks are described in detail in the following two sections.

Adaptive Fuzzy Thermal Model

A thermal model describes interactions between active components, like the CPU, memory, and fans. Every active component in a server system may have a thermal/power relationship with every other active component; however for the geometric and time scales involved in fan management, most of these relationships are fairly weak and can be safely ignored. Identifying, for each component, the key thermal relationships with the rest of the system and constructing accurate thermal models are two of the most difficult challenges to overcome for deploying effective fan management solutions.

"A thermal model describes interactions between active components, like the CPU, memory, and fans."

Before going into the details, the main elements of the system can be summarized as follows. When the fan speeds change, and therefore the airflow inside a computer system changes, thermal relationships change in a nonlinear way. Capturing these nonlinear interactions is necessary for an effective and robust fan speed control system. The output of a fuzzy thermal model is a predicted device or zone temperature. The past values of the output become inputs to the thermal model, together with the past and present values of the power inputs. These inputs provide an indication of the active power consumed when some component is used. The parameters of the system can be self-tuned on the fly using a least-squares parameter adaptation algorithm. Using this fuzzy model, critical temperatures in the system are predicted ahead of time, enabling proactive control versus a purely reactive approach.

"When the fan speeds change, and therefore the airflow inside a computer system changes, thermal relationships change in a nonlinear way."

Using the fuzzy thermal model, the temperature at time $t+1$ predicted at time t for a particular sensor is given by

$$\hat{y} = \bar{w}_1 y_1 + \bar{w}_2 y_2 + \cdots + \bar{w}_C y_C \tag{6}$$

From Figure 1, Equation 6 corresponds to the output of a TS system with C rules. For a system with p fans we have the following formula for the membership degree of the i-th rule.

$$w_i = \prod_{j=1}^{p} Z_{ij}(z_j) \tag{7}$$

The symbol $\prod$ indicates multiplication over the corresponding fuzzy sets in the antecedent part of the rules and the scheduling variables, z_j, represent the fan inputs applied at time $t-d$. The delay d represents the time delay between an applied fan input and the resulting thermal response of the system. The consequent of the i-th rule is given by

$$y_i = \varphi_{i0} \tag{8}$$

Where, as in the example given in the section "Takagi-Sugeno Fuzzy Systems," only the singleton consequent parameter was kept. Substituting equation (8) in equation (6) and dropping the sub-index 0 from the singletons, the predicted temperature can be calculated as follows.

$$\hat{y} = \bar{w}_1 \varphi_1 + \bar{w}_2 \varphi_2 + \cdots + \bar{w}_C \varphi_C \tag{9}$$

To understand the meaning of the singletons, refer once again to Figure 3 and consider the case when the fan input z_1 is 0 and the fan input z_2 is also 0. In this case, the predicted temperature would be exactly $\hat{y} = \varphi_1$. Therefore, φ_1 corresponds to the predicted temperature when the applied fan speed inputs are Slow1 and Slow2, respectively. In general, the singletons correspond to local solutions of the nonlinear TS system for a particular combination of fan inputs, which we refer to as the scheduling variables. This insight leads to a method to determine the value of the singletons for a particular system, for consider what happens when the fan speeds are constant. It is well known that when this is the case, the future temperature of an electronic component can be estimated accurately from a linear function of the applied power (see for example Huang et al.[8]).

"In general, the singletons correspond to local solutions of the nonlinear TS system for a particular combination of fan inputs."

$$\varphi_i(t+1) = -\sum_{k=1}^{n_a-1} a_k y^{t+1-k} + \sum_{k=0}^{n_b-1} b_k u^{t+1-k} \tag{10}$$

Where φ_i is the temperature at time t+1 predicted at time t when the fan inputs correspond to the antecedent of the i-th rule. The first term on the right hand side is a regression of the observed temperature, y^{t+1-k}, up to time t. The second term is a moving average of the observed power, u^{t+1-k}, up to time $t+1$. In vector form, equation 10 can be rewritten as

$$\varphi_i(t+1) = \theta_i^{\mathrm{T}} \Psi(t) \tag{11}$$

The superindex T on the right side of the equality is the vector transpose operator. The parameters $a_1 \cdots a_{na-1}$ and $b_0 \cdots b_{nb-1}$ are collected in column vector θ_i and the recent temperature and power measurements are collected in column vector Ψ. Combining equations (9) and (11), the predicted temperature from the fuzzy thermal model is given by

$$\hat{y}(t+1) = \left(\theta_1^{\mathrm{T}}, \theta_2^{\mathrm{T}} \ldots, \theta_C^{\mathrm{T}}\right) (\bar{w}_1 \Psi^{\mathrm{T}}(t), \bar{w}_2 \Psi^{\mathrm{T}}(t) \ldots, \bar{w}_C \Psi^{\mathrm{T}}(t))^{\mathrm{T}} = \theta^{\mathrm{T}} \Phi(t) \tag{12}$$

The parameters in equation 12 can be identified adaptively using a least squares parameter adaptation algorithm. The book by Lughofer[5] is a good reference to learn more about such algorithms and their particular application to fuzzy systems. We'll consider the identification problem in the section "Self-Tuning, Adaptive Fan Speed Control."

Optimum Fan Speed Control

Model predictive control (MPC) is a general approach to nonlinear optimal control[6]. Applying MPC to fan speed control, the future output $y(t+T)$ of the system is estimated at time t using the fuzzy thermal model such that the predicted output $\hat{y}(t+T)$ is a function of the current observation vector $\psi(t)$ and of the fan speed $z(t+k)$, where $0 \le k < T$ and $z(t+k) = z(t+K-1)$ for $K \le k < T$. Hence the fan speed varies only within the control horizon K.

The control law is obtained by minimizing an objective function. This minimization problem yields the optimum control sequence $z^* = \{ z^*(0), z^*(1) \cdots z^*(K-1)\}$. The control applied at instant t corresponds to $z(t) = z^*(0)$. All that remains to be done at this point is define the optimization problem to be solved in the MPC scheme. Since our interest in fan speed control is using it to minimize system power, the cost function should include the power consumed by the fans in cooling the system and the leakage power that is a function of the electronic components' temperature. Our cost function is then defined as

"The control law is obtained by minimizing an objective function. This minimization problem yields the optimum control sequence."

"The cost function should include the power consumed by the fans in cooling the system and the leakage power."

$$L = \sum_{k=1}^{k=T} P_a(\mathbf{z}(k)) + P_s(y(k)) \tag{13}$$

The first term on the right side of the equality, P_a, is the power consumed by the fans, which is a function of the fan inputs only. Therefore, for a given set of fan inputs, z, we can measure the power consumed by the fans and associate the value obtained with one of the rules of the fuzzy thermal model. In other words, each rule in the fuzzy thermal model has a cost associated with it that represents the power needed to drive the fans at the corresponding input's levels specified in the rule's antecedent. Consider now the second term on the right side of the equality in equation 13. Since leakage is a function of temperature, we can use equation 8 and associate to each rule in the fuzzy thermal model a cost term proportional to the estimated leakage power. The total cost of each rule is therefore

$$L_i = P_a(\mathbf{z}_i) + P_s(\varphi_i) \tag{14}$$

The optimization problem associated with MPC can now be stated as follows:

$$\text{Minimize} \quad L = \sum_{k=1}^{k=T} \sum_{i=1}^{C} \bar{w}_i(\mathbf{z}(k-d)) L_i(k) \tag{15}$$

$$\text{such that} \quad \sum_{i=1}^{C} \bar{w}_i(\mathbf{z}(k)) = 1, \quad 0 \le k < T \tag{16}$$

$$\text{and} \quad \sum_{i=1}^{C} \bar{w}_i(\mathbf{z}(k-d)) y_i(k) \le y_{\lim}, \quad 0 \le k < T \tag{17}$$

Equation 15 represents the total system power, including cooling power and leakage. As mentioned in the previous section, the delay d represents the time delay between an applied fan input and the resulting thermal response of the

system. Equation 16 is the normalization condition for the fuzzy weights. Equation 17 represents the condition that the resulting temperature must be within the thermal limits of the component. Notice that the optimization problem defined by equations 15 through 17 can be solved using a standard linear programming solver such as the simplex [7]. The solution of the optimization problem is the set of fuzzy weights $\{\bar{w}_1^*, \bar{w}_2^*, \ldots \bar{w}_C^*\}$ that should be applied to the rules of the fuzzy thermal model. The final step is thus to determine the optimal fan speeds as follows:

$$z_j(t) = \bar{w}_1^* Z_{1j} + \bar{w}_2^* Z_{2j} + \cdots + \bar{w}_C^* Z_{Cj} \tag{18}$$

Self-Tuning, Adaptive Fan Speed Control

In this section, we use an example to illustrate the implementation aspects of a self-tuning adaptive fan speed control system. Figure 4 shows a thermal model of a CPU with cooling solution.

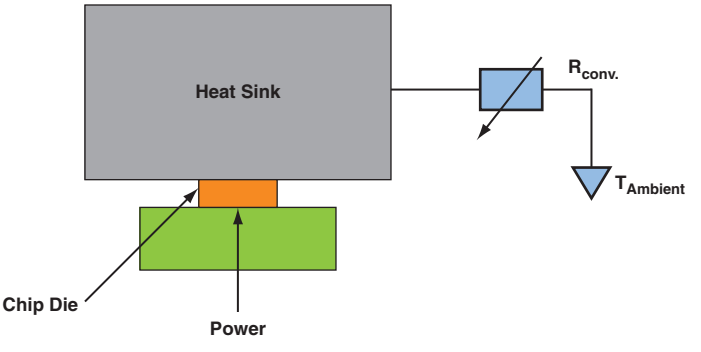


Figure 4: Thermal model of a CPU with cooling solution (Source: Intel Corporation, 2012)

The model can be represented as a thermal network in a similar way as is done for example in HotSpot[8]. Figure 5 shows a thermal network model representation of the CPU system depicted in Figure 4.

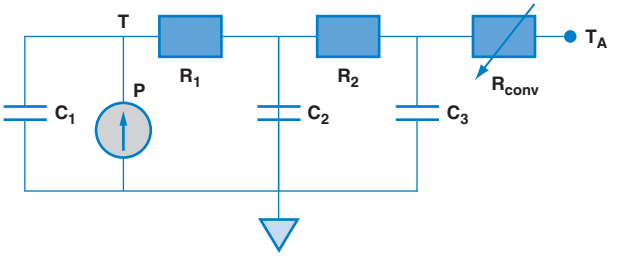


Figure 5: Thermal network model (Source: Intel Corporation, 2012)

"The thermal network is a simplified way to represent the dynamic thermal behavior of the system."

The thermal network is a simplified way to represent the dynamic thermal behavior of the system in terms of equivalent thermal resistances and capacitances. In

the thermal-electrical analogy, temperatures correspond to voltages and power corresponds to current. The CPU power includes the power loss due to leakage.

$$P_s(\varphi_i) = F_s \times (M\varphi_i + N) \quad (19)$$

F_s is the leakage current. M and N are parameters obtained by curve fitting in the piece-wise linear model[9].

The thermal model has a variable thermal resistance, R_{conv}, to represent the effect of the fan on the system. In our model, R_{conv} is defined as an inverse exponential function of fan speed.

$$R_{conv} = a\omega(z)^{-\alpha} \quad (20)$$

The function $\omega(z)$ in equation 20 corresponds to the CPU fan speed in RPM that results from applying the fan input z. The parameters a and α depend on the fan and heat sink characteristics. Fan power is proportional to the cube of the speed in RPM, so if on average we change the fan speed from some reference value RPM1 to a new value RPM2, we have from the fan law[10] that

"Fan power is proportional to the cube of the speed in RPM."

$$RPM1/RPM2 = (Power1/Power2)^3 \quad (21)$$

Equations 19–21 plus the set of differential equations that can be derived from the thermal network model of Figure 5 by applying the thermal-electrical analogy describe the dynamics of the CPU thermal model. This model was used to generate data to train an adaptive fuzzy thermal model based on the scheme described in the section "Model-based Predictive Control." Table 1 shows the parameters of the model.

Parameter	Value
Thermal resistances & capacitances (Fig. 5)	$R_1 = 0.305$, $R_2 = 0.122$, $C_1 = 1.141$, $C_2 = 19.45$, $C_3 = 30.71$
Fan equation (Eq. 20)	$a = 9000$, $\alpha = 1.20$
Ambient temperature	$T_A = 22°C$
Number of fan inputs	$p = 1$
Number of rules	$C = 3$
Control delay	$d = 1$
MPC control horizon	$K = 1$, $T = 1$
Fan power (per rule)	$P_a = \{1.3, 2.4, 14.0\}$
Leakage power (Eq. 19)	$F_s = 1$, $M = 0.22$, $N = -4.5$
Fuzzy sets	Slow = (0, 0.025), Medium = (0.175, 0.057), Fast = (1, 0.225)
Local models (Eq. 10)	$n_a = 12$, $n_b = 12$

Table 1: Model parameters
(Source: Intel Corporation, 2012)

The steps needed to generate the model from data are explained next. The first step to generate a fuzzy thermal model for the system of Figure 4 is to partition the input space using fuzzy sets. In this example we use Gaussian fuzzy sets, with the input, z, corresponding to the normalized voltage or PWM input applied to the fan. The

"The first step to generate a fuzzy thermal model … is to partition the input space using fuzzy sets."

fuzzy inputs can therefore be assigned linguistic values such as Slow, Medium, and Fast. Hence, the following three rules for the fuzzy system are obtained.

IF $z(t-d)$ is Slow THEN $y_1(t+1) = \varphi_1(t+1)$ (22)

IF $z(t-d)$ is Medium THEN $y_2(t+1) = \varphi_2(t+1)$ (23)

IF $z(t-d)$ is Fast THEN $y_3(t+1) = \varphi_3(t+1)$ (24)

Recall that the output of the fuzzy model corresponds to the predicted temperature that is obtained, via equation 9, by combining the output from the local models. Depending on the nonlinearity, a large number of local models may be necessary. While there are no general guidelines on how to choose the number, position, and shape of the fuzzy sets, in the present example the two extremes of the fan input range represent good choices for the Slow and Fast sets. The fuzzy set Medium can be placed initially at an arbitrary location between the first two. If necessary to improve the accuracy of the fuzzy model, it can be updated and/or extended adaptively at a later stage.

"Identification, or training, can be done for both the antecedent and consequent parts of the fuzzy rules."

In the case of a single input system, one local model can be associated with each fuzzy set used in partitioning the input space. Each of these local models needs to be identified adaptively. Identification, or training, can be done for both the antecedent and consequent parts of the fuzzy rules. In general, local consequent training is more robust than global training because of the inherent regularization. For the present example, an affine projection algorithm[11] was used for local model identification of the consequent parts. A standard gradient descent least-squares algorithm was used to optimize the parameters of the Medium fuzzy set, while the Slow and Fast sets remained fixed. Figure 6 shows the resulting fuzzy sets.

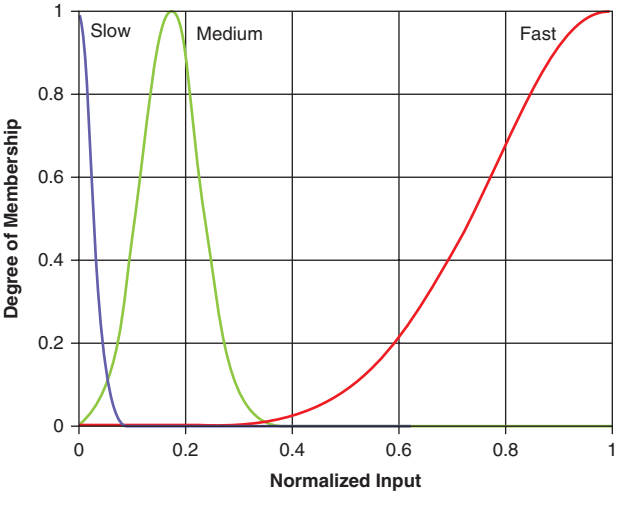


Figure 6: Partition of the fan input space using Gaussian fuzzy sets
(Source: Intel Corporation, 2012)

"A near perfect sequence of odd length was used to generate a power trace for local model training."

A near perfect sequence of odd length was used to generate a power trace for local model training[12]. The linear model parameters (equation 11) were identified adaptively using the following update equation[11].

$$\theta_i(t) = \theta_i(t-1) + \Psi(\Psi^T\Psi + r\,\mathbf{I})^{-1}\mathbf{E} \quad (25)$$

Where E is a $n_e \times 1$ error vector defined as

$$\mathbf{E} = (e(t), e(t-1), \ldots e(t-n_e+1))^T \quad (26)$$

Matrix Ψ is similarly defined (abusing notation) using the n_e most recent values of the measurements vector $\Psi(t)$ from equation 11. For the present work $n_e = 2$ was used. Matrix I is the $n_e \times n_e$ identity matrix and r is a regularization parameter (in our case, $r = 1e-8$).

After consequent and antecedent training were done, a different pseudo-noise sequence was applied to test the fuzzy model as shown in Figure 7.

"After consequent and antecedent training were done, a different pseudo-noise sequence was applied to test the fuzzy model."

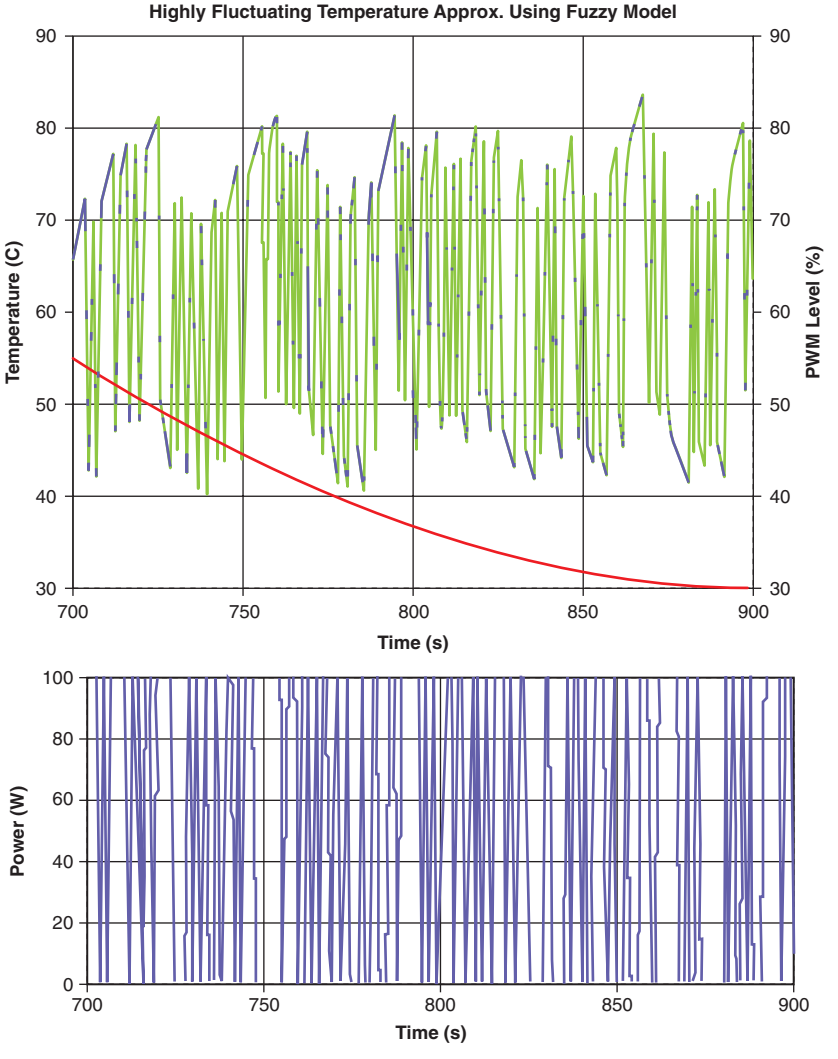


Figure 7: Simulation results using thermal network model and fuzzy thermal model. Top: temperature from network model (blue line), predicted temperature from trained fuzzy model (green line) and fan input (red line). Bottom: CPU power input applied to thermal network model. (Source: Intel Corporation, 2012)

"The fuzzy thermal model is used to predict the temperature d + 1 seconds ahead of the current time."

"If necessary, the fuzzy thermal model can be updated or extended at any time."

Having trained a fuzzy thermal model of the system, the next step is to calculate the total cost of each rule using equations 14, 19, and 21. This is the last preliminary step before the optimum fan speed control strategy can be deployed to control fan speed. During operation in a system, the flow of the adaptive fan speed control system is as follows. Once a second, the fuzzy thermal model is used to predict the temperature $d+1$ seconds ahead of the current time, using CPU power and temperature data sampled every a second. Based on these estimates, the linear optimization problem defined by equations 15–18 is solved to determine the fan speed level that is applied in the next second. If necessary, the fuzzy thermal model can be updated or extended at any time to cope for example with hardware configuration changes or aging of the components.

Figure 8 shows simulation results that illustrate the performance of an adaptive fan speed control system compared to a standard PID (proportional, integral,

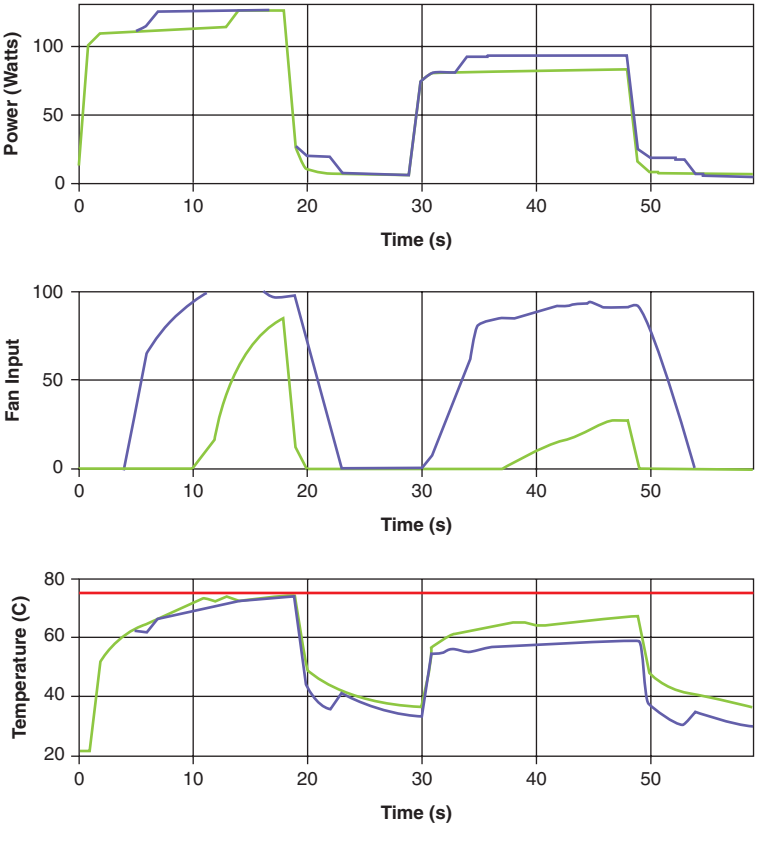


Figure 8: Simulation results using PID (blue line) and adaptive (green line) fan speed control. Top: total power is the sum of the active CPU power, leakage, and cooling power to drive the fan. Middle: fan input. Bottom: CPU temperature. The red lined represents the $T_{control}$ setting. Fans should be at 100 percent whenever the CPU temperature is above $T_{control}$.
(Source: Intel Corporation, 2012)

derivative) control. It can be observed that both methods successfully limit CPU temperature but the adaptive control manages in addition to significantly reduce the total power consumed by the system. Note that the parameters of the PID controller were manually tuned for this particular example to ensure that the thermal limit was not exceeded. On the other hand, no manual tuning was necessary for the adaptive system, as expected.

Conclusions

This article presented an adaptive system for fan speed control and power management. The system is self-tuning so thermal guard bands and system power can be minimized in every system. The system is data driven so it can be updated and extended automatically, enabling a scalable solution for optimal cooling. The fan controller maintains a positive headroom for each device at all times while minimizing total system power.

"The system is self-tuning so thermal guard bands and system power can be minimized in every system."

"The fan controller maintains a positive headroom for each device at all times while minimizing total system power."

Acknowledgements

The author gratefully acknowledges the contributions to this project from Willem Beltman, Murli Tirumala, Daryl Nelson, Karthik Sankaranarayanan, and Christian Le. He also thanks David Gomez for reviewing the first draft for technical content and making valuable suggestions that were incorporated in the final version.

References

[1] Intel® Pentium® D Processor and Intel Pentium Processor Extreme Edition 840 Thermal and Mechanical Guidelines, (Intel, May 2005).

[2] T. Byquist and N. Weber, "Advanced Fan Speed Control Tuning Lab," (Intel Developer Forum 2005).

[3] K. Man and G. Chandrasekaran, "Server fan speed control for better performance and acoustics," 2006 Intel Development Forum, PTMS003

[4] Z. Wang, C. Bash, N. Tolia, M. Marwah, X. Zhu, and P. Ranganathan, "Optimal Fan Speed Control for Thermal Management of Servers," Proceedings of the ASME InterPACK 2009, June 2009.

[5] Lughofer, E. *Evolving Fuzzy Systems – methodologies, advanced concepts and applications*, Springer-Verlag, 2011.

[6] D. Q. Mayne, J. B. Rawlings, C.V. Rao and P. Scokaert, "Constrained model predictive control: stability and optimality," Automatica 36, 789–814 (2000).

[7] Luenberger, D. G. and Ye, Y. Linear and nonlinear programming, 3rd edition, Springer-Verlag, 2008.

[8] Huang, W., Sankaranarayanan, K., Skadron, K., Ribando, R. J., and Stan, M. R. Accurate, pre-RTL temperature-aware processor design using a parameterized, geometric thermal model. IEEE Transactions on Computers, Vol. 57, No. 9, 1277–1288, 2008.

[9] Liu, Y., Dick, R. P., Shang, L. and Yang, H. Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy, DATE07, 2007.

[10] Smith, N. High Efficiency Electronic Cooling Fans. SEMI-THERM, 2009.

[11] Sayed, A. H. *Adaptive Filters*, John Wiley & Sons, Inc., 2008.

[12] Antweiler, C. and Antweiler, M. System Identification with Perfect Sequences Based on the NLMS Algorithm, International Journal of Electronics and Communications, vol. 3, May 1995, pp. 129–134.

Author Biography

Rafael de la Guardia is a senior hardware engineer in Intel Labs. Rafael joined Intel in 2005, working in the area of adaptive systems. He received a doctorate in engineering from the National Autonomous University of México (UNAM). His email is Rafael.de.la.guardia@intel.com.

A NOVEL CONTROL DESIGN APPROACH FOR SERVER SUBSYSTEMS: THE CONCEPT OF ACTIVE DISTURBANCE REJECTION AND A CASE STUDY

Contributors

John Ping
Data Center and Connected Systems Group, Intel Corporation

Zhiqiang Gao
Center for Advanced Control Technologies, Cleveland State University

Rahul Khanna
Software and Services Group, Intel Corporation

The fast data center growth and cloud computing implementations drive the demands for a higher server system power efficiency to reduce data center energy cost. In this article, a novel control strategy is explored for power optimization to key components in a server system, using the voltage regulator (VR) control as an illustrative example. The new approach is based on the unique active disturbance rejection control (ADRC) principle, which actively estimates, and compensates for, disturbances to the system caused by dynamic load changes rather than passively reacting to them as most existing methods do. Hence the controller is inherently efficient in rejecting the disturbances in real time. Without any hardware changes, this methodology leads to substantial power saving in a highly dynamic load environment in a simulation study.

Introduction

The US data center industry is in the midst of a major growth period stimulated by increasing demand for data processing and storage[1][2]. Financial services, Internet communication and entertainment, media, and global commerce all drive fast growth of the data center, along with a significant increase in energy consumption and its associated cost from the server system and data center infrastructure. The server system power efficiency becomes a frontline issue in server architecture, design, and research[3][4][5].

An Intel server system is shown in Figure 1. Under the hood of a modern server, we see many subsystems or circuits that are separately controlled. The server subsystem system controls can be characterized in several categories: voltage regulator control, power energy control and optimization, and thermal management and control. At the OS level, the control issues could be workload control, performance optimization, and so on. Each of these subsystems is quite different in its dynamics, but they all seek better control means to improve efficiency, robustness, smartness, and yet, at the same time, retain ease of use and intuitiveness. The improvement of control methodology or strategy in each subsystem in the server could result in a major improvement of the overall server system in terms of power efficiency, performance, and adaptation.

"The improvement of control methodology or strategy in each subsystem in the server could result in a major improvement of the overall server system in terms of power efficiency, performance, and adaptation."

Undoubtedly automatic control systems play a crucial role in server systems and yet their design and tuning have not been the focus of our work until recently. Our default solution for many years has been the conventional proportional-integral-derivative (PID) controller that dates back to early 1900s[6][7][16]. It is still widely used in server subsystems today due primarily to

Figure 1: Intel server system
(Source: Intel Corporation, 2012)

its simplicity and our familiarity with it. But perhaps we can no longer ignore its intrinsic shortcomings, including but not limited to the following:

- It is mostly tuned by trial-and-error, leaving much room for systematic improvement.
- It has limited ability to reject disturbances, such as load changes and process dynamics variations, which is the primary function in any control systems and this imposes unnecessary constraints on server systems.
- It regulates the system by reacting to the deviations in the process variables, such as voltage and temperature, from their desired values, also known as setpoints, wasting energy in the process, especially during high dynamic load change in server operations.

It is our belief that to overcome such shortcomings we must make a fundamental change in how we approach the problem for server subsystem control: instead of passively reacting to disturbances, we propose an active disturbance rejection (ADR) paradigm where the disturbance information is gathered and used preemptively in limiting the disturbance impact on the system. That is, we propose a method that will help eliminate the deviation before it appears, therefore saving the energy that would be otherwise needed in correcting the deviation.

Such a design principle has been discussed in depth before[8][9][10]. The key in general is to find a way of getting ahead of the curve in mitigating set-point deviation, as opposed to always playing catch-up like PID does most of time. The focus of this article is to creatively adapt the ADR principle to server problems, utilizing all our relevant knowledge of server dynamics. The key to the solution is how we obtain the disturbance information and fully taking advantage of it in helping the controller to get ahead in mitigating disturbance effects.

"Active disturbance rejection (ADR): the disturbance information is gathered and used preemptively in limiting the disturbance impact on the system."

"Eliminate the deviation before it appears."

This new way of thinking about server control problems is rooted in our understanding of a control system's primary task as that of disturbance rejection, upon which system performance is evaluated. Acting on the source of the deviation, that is, disturbance, as opposed to deviation itself, gives us the advantage of getting ahead, of treating the cause, not the symptom. As will be demonstrated, this has a profound impact on future energy saving in the server market.

In this article, we use a typical CPU VR control subsystem as an example to apply the ADR principles. The server in a data center normally runs in a high dynamic workload environment with the various tasks running above the operating system making the CPU current changes drastically in real time; this makes it a tough disturbance to deal with for the VR controller. From the perspective of efficiency, however, any improvement in the VR controller in handling each single load change will add up to potentially significant energy savings in a highly dynamic environment with big swings in load current. It is in this environment that we'll design and validate the ADR methodology to actively reject the disturbance in the CPU VR system and compare the performance and energy consumption with respect to step load changes, as it is compared to the standard PI controller currently used; with a given average dynamic load fluctuation, we derive the energy saving over a period of time.

The article is organized as follows: the following section, "Background: What Is the Control Anyway?" describes the related work and background of the control algorithms. The next section, "Active Disturbance Rejection," introduces the ADRC algorithm. Next, the section "Active Disturbance Rejection in a Server VR Subsystem" describes the ADRC control method on a server VR subsystem. "Comparing ADRC to Existing Solutions" provides an analysis of the result and makes the comparison between the PID and ADRC in terms of control performance and power efficiency. This is followed by a summary of the article.

"Today, automation has been built into the very fabric of modern society, from massive production lines of consumer goods to individual homes and personal electronic devices."

"Over a period of two centuries, control technology has emerged as a crucial centerpiece in all engineered systems, …"

Background: What Is Control Anyway?

Since not all server design engineers are well versed in the concepts and terminology of controls, we start with this basic question. Automatic control is a technology that has played a crucial role in industry ever since the era of the steam engine and the industrial revolution in the 18th century. Today, automation has been built into the very fabric of modern society, from massive production lines of consumer goods to individual homes and personal electronic devices. From the vantage point of control engineers, everything is a part of a process, or system, within which all variables are in some way interdependent to each other. The objective of control system design is to make such dependency, in a particular case, meet a predetermined goal or set of criteria. Over a period of two centuries, control technology has emerged as a crucial centerpiece in all engineered systems, simply because all such systems have a goal to reach, a need to satisfy, and the resources to reach the goal. To satisfy the need is what we call the act of control.

The act of control can be divided into two phases: collecting information and acting on it. Using the CPU VR control as an example, the goal is to provide a constant voltage supply for CPU to function. The information that can be collected are values of process variables such as voltage and current at various points in the circuit. Such information is used by the controller to adjust the amount of power supplied to the CPU—not too much, not too little, just right! That is, in a perfect CPU VR system, the power supplied to the CPU is exactly what it needs, resulting in a voltage supply that is kept at a constant 3.3 volts, despite huge, unpredictable swings in load current.

Perfect control, of course, doesn't exist in the real world. For instance, when we turn on a washing machine at night, the light may dim momentarily, indicating a voltage dip when the load current suddenly increases. The same thing happens in the VR control system: when the load currently unexpectedly increases, the voltage dips, the extent of which shows the ability of the "disturbance rejection" of the controller, a primary criterion and a central task in control design.

Curiously, little has changed since the beginning of the modern era in how we perceive and solve the disturbance rejection problem in control: we wait, we see, and we react to the deviation in the process variable from its desired value, or setpoint, the deviation caused by disturbances. Much progress has been made in all aspects of control engineering, techniques, hardware, and software, and so on, but this reactive paradigm has endured over two hundred years, crystallized in the dominant industrial control technology known as PID[6], a technology defined by how it react in three ways to the setpoint deviation, that is, tracking error, proportional, integral, and derivative, as shown in equation 1.

"Much progress has been made in all aspects of control engineering, ... but this reactive paradigm has endured over two hundred years."

$$u = K_p e + K_I \int e\,dt + K_D \dot{e} \tag{1}$$

where u is the control signal, e is the error between the process output and its desired value, and $\{K_p, K_I, K_D\}$ are controller gains. Over 95 percent of industrial controllers are of this type[6], an alternative to which is discussed below.

Active Disturbance Rejection

Emerging after World War II as a distinct engineering discipline, automatic control has been synonymous with feedback largely thanks to Norbert Wiener's brainchild of Cybernetics[17]. Wiener calls it "control by informative feedback," which means that "when we desire a motion to follow a given pattern the difference between this pattern and the actually performed motion is used as the new input to cause the part regulated to move in such a way as to bring its motion closer to that given by the pattern." In other words, the control mechanism first sees the deviation and then acts on it in order to reduce it. Such conception by Wiener influenced generations of control scientists and engineers and dominated the field ever since the publication of his book in 1948. Many, if not most, control textbooks have the word *feedback* in the

title. The renowned historian of control engineering, Otto Mayr, goes as far as saying that "this field is essentially based upon a single idea, that of the feedback loop" and there was never a serious debate or reflection on it, or was there?

The success, as well as the occasional problems of oscillation, of steam engines some 200 years ago attracted the attention of many scholars, and engineers. Among them, Jean-Victor Poncelet, a prominent French scholar and engineer, in the early 1800s conceived of a very different idea of control: measure the load disturbance to the engine and cancel it out with the adjustment of steam flow *before* the engine speed is affected[9]. Some 100 years later, Russian scholars revived Poncelet's idea and developed an entirely different theory and practice of automatic control that is called "combined system" or dual channel, where disturbances are measured and the information is used to make a much more effective control system[18][19]. A few scholars and engineers in England and the United States also discovered the benefits of adding the so called "feed forward" element to the control system, as shown in nature and manmade systems' control systems alike[6][20].

The Origin of ADRC

Conceived by Jingqing Han in the mid-1990s, Active Disturbance Rejection Control (ADRC) is in the same vein of the invariance principle of the Soviet scholars a few decades earlier, exposed to him when he was a budding graduate student in Moscow. By late 1980s, Han, well established as one of the top control theorists in China, openly challenged the modern control paradigm in the vein of Kalman Filter and mathematical control science, predicated on accurate model of the reality[21]. Han believed that such conception of the problem and presumption in its solution could be called a theory of mathematical models, but not of controls. Han believed that the Soviet scholar got it right: control systems are about disturbances; in particular, they are about how one strives to make the controlled variables, or process outputs, invariant under the assault of disturbances ubiquitously internal and external.

"PID had dominated industrial controls for decades and no serious researchers could ignored the reality any longer and avoid the question"why?" "

"If there is no uncertainty in the system, the control, or the environment, feedback control is largely unnecessary" – Roger Brockett.

The background of Han's 1989 paper[21] is that PID had dominated industrial controls for decades and no serious researchers could ignored the reality any longer and avoid the question "why?" If there was competition in engineering practice between PID and its users against the vast edifice of modern control theory and its creators and builders in academia, PID would have won hands down and everyone knows that. What is not so clear was the reason behind such a big divide between how control is practiced and how it is researched and taught. It took a scholar of the highest caliber to pinpoint the cause: our reliance on mathematical models and a misconception of what control engineering really is.

What a mathematical model represents is the known dynamics of the process being controlled; but the real task of control, the reason to have a control system in the first place, is to deal with the unknowns and unpredictables, also known as disturbances. Renowned control theorist Roger Brockett once said "If there is no uncertainty in the system, the control, or the environment, feedback control

is largely unnecessary."[22]. But modern control theory largely proceeded along the lines of the following: given the mathematical model, design a control law to achieve some measure of optimality, which is a valid question in itself but not necessarily the *only* control problem out there. Hence the theory/practice dichotomy and the eighty-year dominance of PID. The question was "What can we do about it?" The answer, according to Han, was ADRC.

From 1989 to the time of his passing in early 2008, Han dedicated the last two decades of his life to an alternative to PID and he came up with much more than just a replacement algorithm. ADRC, according to Han, "inherits from PID the quality that makes it such a success: the error-driven, rather than model-based, control law; it takes from modern control theory its best offering: the state observer; it embraces the power of nonlinear feedback and puts it to full use; it is a useful digital control technology developed out of an experimental platform rooted in computer simulations."

"A viable control law cannot be model driven."

In other words, Han concluded that a viable control law cannot be model driven. The success of PID demonstrates the effectiveness and practicality of the error-driven control paradigm. At the same, being a theorist he recognized that the vast research in modern control brought us its crown jewel, the state observer, which can be creatively used to extract the disturbance information from the already available input-output data. ADRC "actively" uses this information to cancel the disturbance out whenever possible, *before* it does any damage, in direct contrast to PID, which only passively reacts to the changes produced by the disturbances after it runs its course through the process.

Another barrier broken through via ADRC is the linear-nonlinear divide in control theory. Instead of the linearizing the nonlinear dynamics so that they can fit into the well-developed linear system theory, Han demonstrated in the ADRC framework that one could purposely add nonlinearity into the PID structure to make it more effective. This and other discoveries are only made possible because the computer simulation provided us with a platform where control research could be done experimentally, like other physical sciences, instead of as a branch of mathematics. Han emphasized that it is through experimental research ADRC was *discovered*, as opposed to *derived*.

"ADRC was discovered*, as opposed to* derived*."*

In summary, ADRC can be viewed as a distinctly different conception of what control is; as a way of conducting an experimental science; and finally as a new control system platform, absorbing the error-driven mentality of PID but adding to it a proactive disturbance rejection facility that makes control truly "active."

Illustration of Active Disturbance Rejection for a Second Order Plant

The conception and the methodology of ADRC obviously is quite general and fundamental, applicable to most control systems across disciplinary boundaries, so much so that any concrete application of it would come with it limitations pertaining only to that application, which is sometimes mistaken

for limitations in general. With this in mind, we introduce a second order, nonlinear, uncertain, and time-varying process and demonstrate how the problem can be reformulated with the guidance of ADRC principles.

Although the ADRC method is applicable, in general, to *n*th order, nonlinear, time-varying, multi-input and multi-output systems (MIMO), for the sake of simplicity, its basic concept is illustrated here using the second-order motion control problem in equation 2.

$$\ddot{y} = p(y,\ \dot{y},\ w,\ u,\ t) \tag{2}$$

of which

$$\ddot{y} = u \tag{3}$$

is an idealization corresponding to Newton's law of motion $f = ma$. Between the totally unknown system of equation 2 and the idealized motion of equation 3, the actual motion system can be described as

$$\ddot{y} = f(y,\ \dot{y},\ w,\ t) + bu \tag{4}$$

That is, $p(y, \dot{y}, w, u, t)$ can be meaningfully separated as

$$p(y,\ \dot{y},\ w,\ u,\ t) \approx f(y,\ \dot{y},\ w,\ t) + bu \tag{5}$$

Adopting a disturbance rejection framework, the motion process in equation 2 can be seen as a nominal, double integral, plant in equation 3 scaled by b and perturbed by $f(y, \dot{y}, w, t)$. That is, $p(\dot{y}, y, w, t)$ is the generalized disturbance and the focus of the control design.

Contrary to all existing conventions, Han proposed that $f(y, \dot{y}, w, t)$ as an analytical expression perhaps is not required or even necessary for the purpose of control design. Instead, what is needed is its value estimated in real time. Specifically, let $\hat{f}$ be the estimate of $f(y, \dot{y}, w, t)$ at time t, then

$$u = (-\hat{f} + u_0)/b \tag{6}$$

reduces equation 1 to a simple double-integral plant

$$\ddot{y} \approx u_0 \tag{7}$$

which can be easily controlled.

"The control of a complex nonlinear, time-varying, and uncertain process in equation 2 is reduced to the simple problem in equation 7."

This demonstrates the central idea of active disturbance rejection: the control of a complex nonlinear, time-varying, and uncertain process in equation 2 is reduced to the simple problem in equation 7 by a direct and active estimation and rejection (cancellation) of the generalized disturbance, $f(y, \dot{y}, w, t)$. The key difference between this and all of the previous approaches is that no explicit analytical expression of $f(y, \dot{y}, w, t)$ is assumed here. The only thing required, as stated above, is the knowledge of the order of the system and the approximate value of b in equation 4. The bu term in equation 4 can even be viewed as a linear approximation, since the nonlinearity of the actuator can be seen as an external disturbance included in f.

Obviously, the success of ADRC is tied closely to the timely and accurate estimate of the disturbance. A simple estimation such as $\hat{f} = \hat{\ddot{y}} - u$ may very well be sufficient for all practical purposes, where $\hat{\ddot{y}}$ denotes an estimation of $\ddot{y}$.

"The success of ADRC is tied closely to the timly and accurate esitmation of the distrubance."

The Extended State Observer and the Control Law

There are also many observers proposed in the literature, including the unknown input observer, the disturbance observer, the perturbation observer, and the extended state observer (ESO). See, for example, a survey in Tian and Gao[9]. Most require a nominal mathematical model. A brief description of the ESO of equation 1 is described below. The readers are referred to Tian and Gao[14], Goa[10][11], and Sun and Gao[12] Zheng and Gao[13] for details, particularly for the digital implementation and generalization of the ESO in Ping and Gao[15].

"The ESO (extended state oberserver) is simplified its implementation and made the design transprent to engineers."

The ESO was originally proposed by J. Han[23]. It is made practical by the tuning method proposed by Goa[11], which simplified its implementation and made the design transparent to engineers. The main idea is to use an augmented state space model of equation 1 that includes f, short for $f(y, \dot{y}, w, t)$, as an additional state. In particular, let

$$x_1 = y,\ x_2 = \dot{y},\ \text{and}\ x_3 = f \tag{8}$$

The augmented state space form of equation 1 is

$$\begin{aligned} \dot{x} &= Ax + Bu + Eh \\ y &= Cx \end{aligned} \tag{9}$$

$$\text{with } A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},\ E = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Note that $x_3 = f$ is the augmented state and $h = \dot{f}$ is a part of the jerk; that is, the differentiation of the acceleration, of motion and is physically bounded. The state observer

$$\begin{aligned} \dot{z} &= Az + Bu + L(y - \hat{y}) \\ \hat{y} &= Cz \end{aligned} \tag{10}$$

with the observer gain $L = [l_1\ l_2\ l_3]^T$ selected appropriately, provides an estimate of the state of equation 9, $z_i \cong x_i$, i = 1, 2, 3. Most importantly, the third state of the observer, z_3, approximates f. The ESO in its original form employs nonlinear observer gains. Here, with the use of linear gains, this observer is denoted as the linear extended state observer (LESO). Moreover, to simplify the tuning process, the observer gains are parameterized as

$$L = [3\omega_o,\ \ 3\omega_o^2,\ \ \omega_o^3]^T \tag{11}$$

where the observer bandwidth, ω_o, is the only tuning parameter.

With a well-tuned observer, the observer state z_3 will closely track $x_3 = f(y, \dot{y}, w, t)$. The control law

$$u = (-z_3 + u_0)/b \tag{12}$$

then reduces equation 4 to equation 7, that is,

$$\ddot{y} = (f - z_3) + u_0 \approx u_0 \tag{13}$$

An example of such u_0 is the common linear proportional and derivative control law

$$u_0 = k_p(r - z_1) - k_d z_2 \tag{14}$$

where r is the set point. The controller tuning is further simplified with

$$k_d = 2\omega_c \quad \text{and} \quad k_p = \omega_c^2 \tag{15}$$

where ω_c is the closed-loop bandwidth[11]. Together, equations 10 through 15 are collectively denoted as the parameterized linear ADRC, or LADRC.

Active Disturbance Rejection in a Server VR Subsystem

"Apply ADRC to a Romley Server CPU PVTT power rail voltage regulation subsystem, and compare the simulation results."

In this section, we apply ADRC to a Romley Server CPU PVTT power rail voltage regulation subsystem, and compare the simulation result with traditional PID control in the next section.

Sandy Bridge CPU VTT Voltage Regulator

The Romley PVTT VR is designed to provide power to the VCCPPA, VCCPCA, VCCPDTTA pins of the Sandy Bridge processor. The VR switching regulator is a single phase synchronous buck converter as shown in Figure 2. It consists of two MOSFETs, one inductor, and one capacitor.

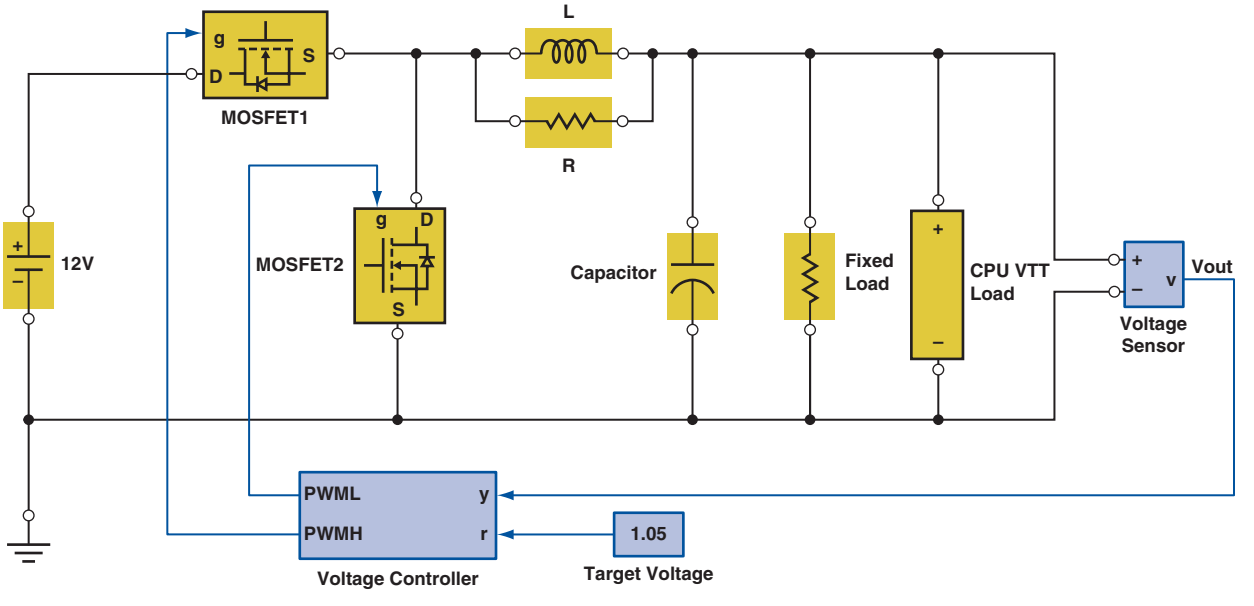


Figure 2: VTT VR circuit
(Source: Intel Corporation, 2012)

It converts the 12 V to 1.05 V Vout or 1.0 V Vout. It is capable of providing a maximum load of 20 A, the maximum step load size is 14 App, and the maximum step load slew rate is 20 A/µs. The frequency of the pulse-width modulation (PWM) is 500 kHz.

The MOSFETs are turned on and off to alternate between connecting the inductor to source voltage to store energy in the inductor and discharging the inductor into the load, and the capacitor smooths the ripple of voltage output from the inductor. The PWM control the MOSFETs open and close the time ratio to determine the output voltage level.

The control object of the controller is to deal with voltage deviation caused by the CPU VTT dynamic load changes and maintain the desired voltage level by adjusting the PWM duty ratio.

MATLAB Modeling of the Voltage Regulator

To be able to test ADRC in simulation, a MATLAB model is built to describe the CPU PVTT buck converter circuit. Based on the original circuit implemented in the Romley Rosecity Server Reference board, we created the model to describe the CPU PVTT VR circuit as shown in Figure 3.

"A MATLAB model is built to describe the CPU PVTT buck converter circuit."

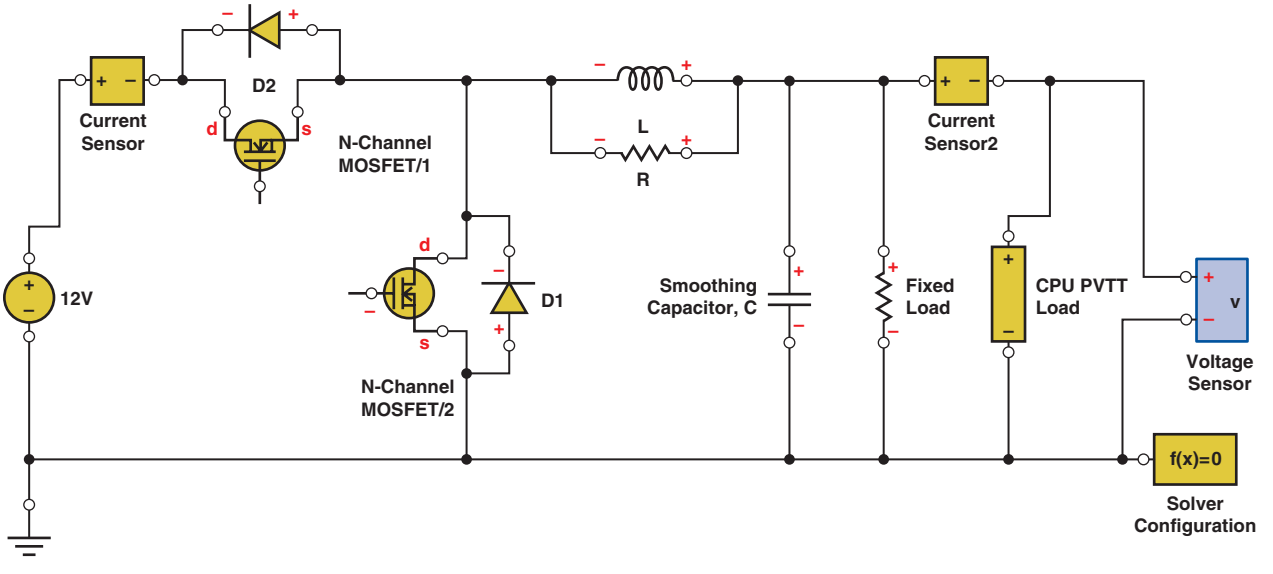


Figure 3: PVTT VR circuit modeling in MATLAB
(Source: Intel Corporation, 2012)

The CPU PVTT load connects to the output of the VR circuit to simulation CPU PVTT load changes. Current sensors are added to the input and output of the VR circuit to get the current reading in real time, and a voltage sensor is applied to the output side; thus the power data can be derived with product of the voltage and current.

Active Disturbance Control Design

As described earlier, the ADRC control law is given as follows:

$$\dot{z}(t) = Az(t) + Bu(t) + L(y(t) - \hat{y}(t))$$
$$\hat{y}(t) = Cz(t) \tag{16}$$

where

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \; B = \begin{bmatrix} 0 \\ b_0 \\ 0 \end{bmatrix}, \; L = \begin{bmatrix} 3\omega_0 \\ 3\omega_0^2 \\ \omega_0^3 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},$$

Here, ω_0 is the bandwidth of the observer. The control law is

$$u = \frac{\omega_c^2 (r - z_1) - 2\omega_c z_2 - z_3}{b_0} \tag{17}$$

where r is the set point and ω_c is the control bandwidth. ADRC has three design parameters, b_0, ω_0, and ω_c, which can be easily *tuned*[8][9][10][11].

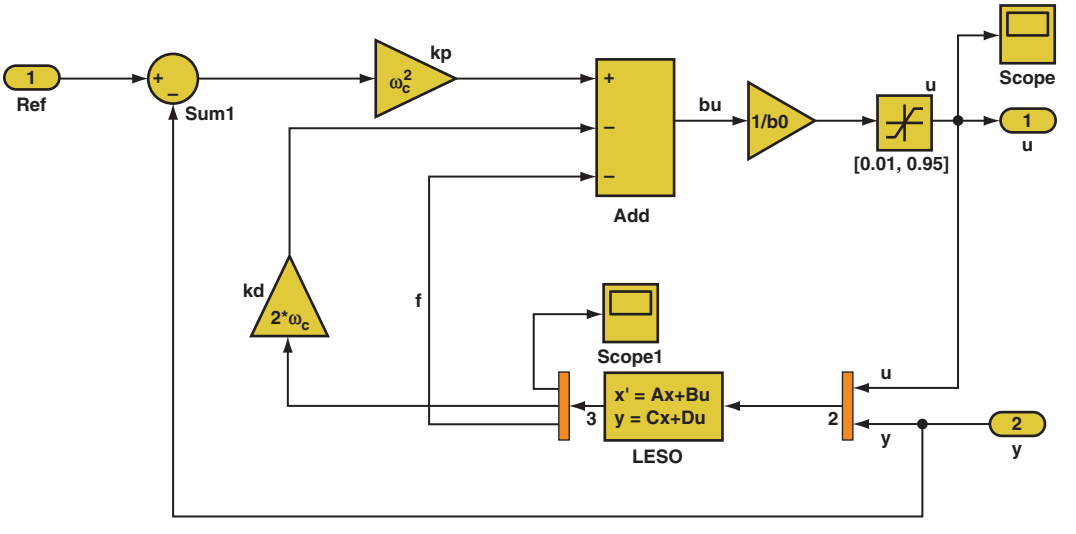


Figure 4: ADRC simulation block diagram in MATLAB
(Source: Cleveland State University, 2003, 2012)

"The model of the ADRC is built in MATLAB and connected with the CPU VTT VR model build from last section."

The model of the ADRC is built in MATLAB as shown in Figure 4, and when connected with the CPU VTT VR model built from last section, we get a fully controlled CPU VTT voltage regulator simulation model, which is shown in Figure 5. A cyclic step load resource to simulate the CPU VTT dynamic load changes is added to the input of the VTT VR model. The setup point to the controller is set to 1.05 V to the ADRC controller to regulator the voltage to 1.05 V.

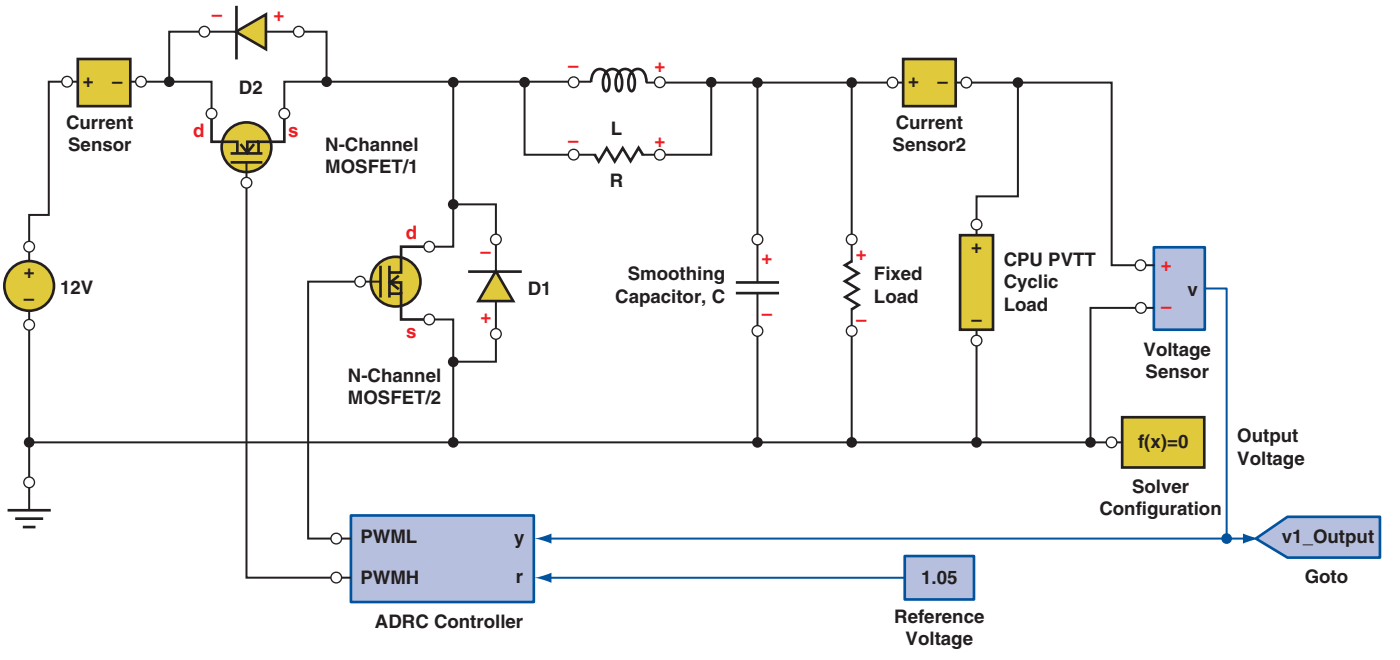


Figure 5: PVTT with ADRC controller modeling in MATLAB
(Source: Intel Corporation, 2012)

ADRC Simulation Result

The simulation result of the ADRC is shown in Figure 6. The top chart is the voltage output, and the lower chart is the simulated CPU VTT cyclic step load change between 0~3 A in the frequency of 200 Hz (for further testing it is an idea to use the maximum load step change as 0~15 A or 0~50 A). The rising curve at the beginning of the output voltage is the control system

"ADRC simulation result is shown with cyclic step load."

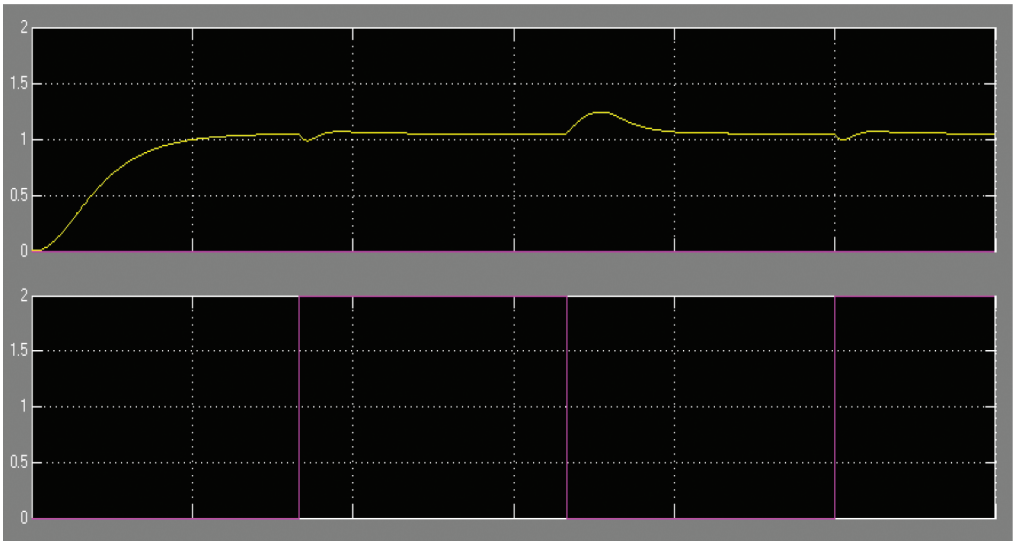


Figure 6: Simulation result of ADRC control
(Source: Intel Corporation, 2012)

transient response when the system starts. After the voltage reaches the desired voltage level at 1.05 V and is in steady state, the cyclic 0~3 A step loads are applied to the output of the regulator. From the simulation result, we can see that the ADRC can quickly correct the overshoot and undershoot caused by the dynamic step load change and quickly recover to the desired voltage without any oscillation. The control action is effective and efficient, thus resulting in a power saving by avoiding unnecessary control effort. In the next section, we compare the ADRC control with tradition PID control and show how much power it can save by ADRC with the same cyclic load over the certain period of time.

Comparing ADRC to Existing Solutions

"As the intrinsic characteristic of the ADRC, it generates more smooth control to the VR circuit and results in power savings."

In this section we compare the ADRC and PID to control the same VR circuit while adding in the same load changes. As the intrinsic characteristic of the ADRC, it generates more smooth control to the VR circuit and results in power savings. We will quantify the power savings based on the simulation comparison result.

Simulation Setup

A Simulink model is set up in MATLAB to compare the ADRC and PID as shown in Figure 7. Two identical VR circuit models we made in the last section are put into the comparison model, and the exact same CPU cyclic loads are applied to each VR circuit. The upper VR circuit model is connected with a PID controller; the lower VR controller is connected with an ADRC controller. To make a real-time comparison, the output voltage, output current, and the control signal from the controller output are fed into the simulation scope so that we can visualize the difference between these two control methodologies. Specifically, the VR input voltage and current are multiplied and have the integration over time to make the energy consumption comparison between these two control methods for the same VR circuit. In addition, the Integral of Absolute Errors (IAE) of the VR voltage output is calculated for each control method for comparison, the purpose of the extraction of IAE data is to make a common reference parameter to make a fair comparison. We make the above comparisons under the condition that the FAE with these two control methods are about the same.

"PID is well known as an empirical design with users assuming little knowledge of the plant dynamics. On the other hand, most design methods based on control theory, classical or modern, require detailed and accurate knowledge of plant dynamics in the form of a mathematical model."

Controller Tuning

In addition to performance, especially disturbance rejection ability, the comparison between controllers must include the ease of use, which consists of two aspects: 1) what does the user need to know to perform the controller design? And more importantly 2) how easy it is to adjust the controller parameters in order to meet different design specifications?

PID is well known as an empirical design with users assuming little knowledge of the plant dynamics. On the other hand, most design methods based on control theory, classical or modern, require detailed and accurate knowledge of plant dynamics in the form of a mathematical model. In practice, the

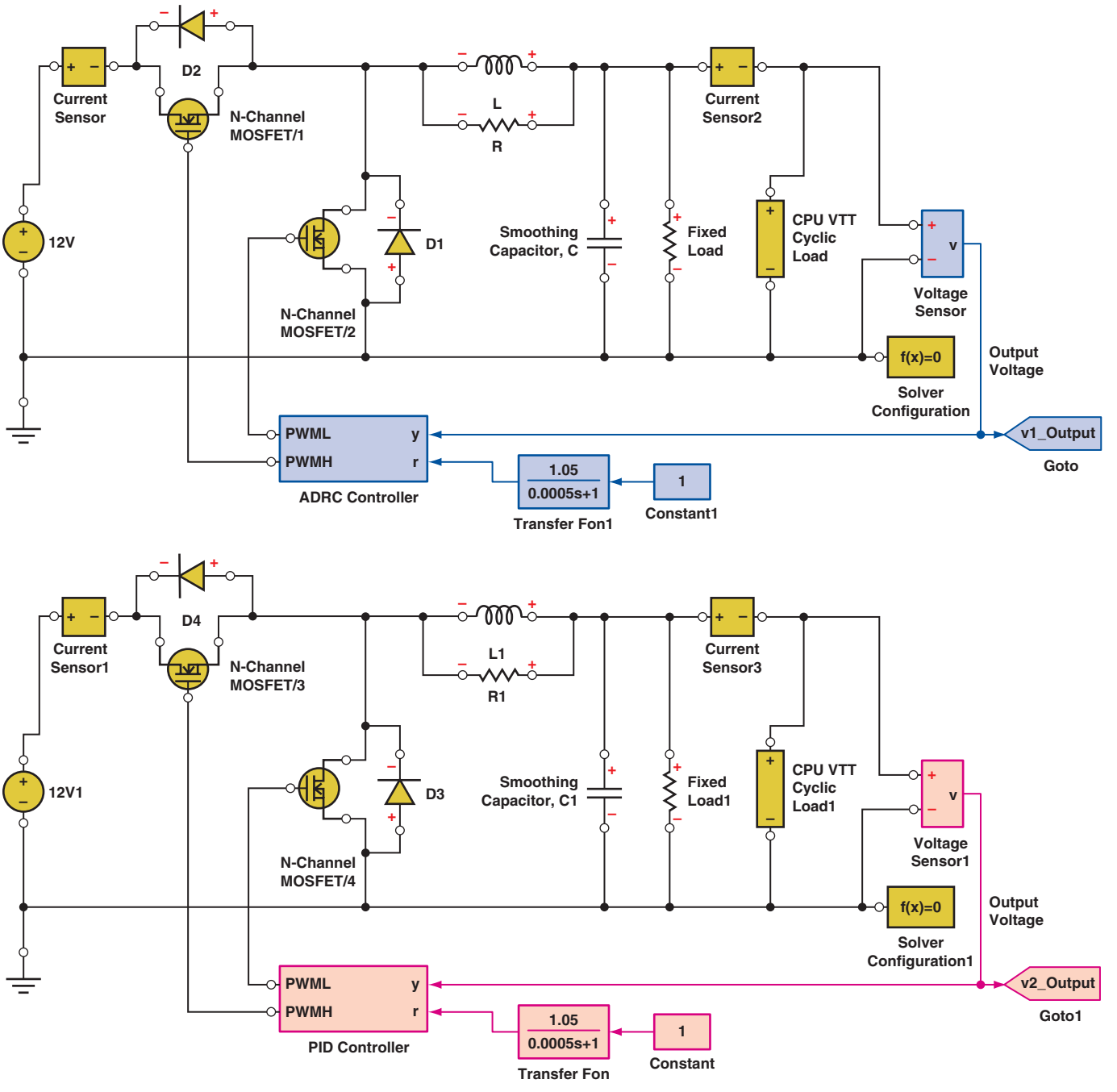


Figure 7: MATLAB Modeling to compare ADRC and PID control (Source: Intel Corporation, 2012)

PID controller is mostly tuned based on the user's experience and model-based controllers are tuned based on the identification or estimation of the parameters of the plant model.

ADRC design and tuning require a different mindset: it presumes that the users are familiar with the physics of the physical process but not necessarily its detailed dynamic relationship between the input and output. Based on such

knowledge the user chooses the order of the plant, n, to be used in design, which is not necessarily the actual order of the plant but, instead, is the order in which the controller will force the plant to behave. Once n is chosen, the users need to know, or acquire the information of, how the change in input u, *approximately* translates to the change in the nth derivative of the output y, as described in the parameter b in equation 9. Such information can be easily obtained as the initial rate of temperature change in a step response test for a thermal system.

"There are two key parameters in ADRC: the observer bandwidth and the controller bandwidth."

Once the order of the plant is selected and the parameter b is obtained, the tuning of ADRC is quite straightforward. Shown in equations 10 through 15, there are two key parameters in ADRC: the observer bandwidth and the controller bandwidth. All observer gains are functions of the former and all controller gains the latter. The observer bandwidth is in general several times higher than that of the controller, to ensure that the state estimation converges fast enough for the controller, although there are exceptions. Once the ratio of the two bandwidths is fixed, the only tuning parameter is the controller bandwidth, which is the measure of the aggressiveness of the control system.

With such single parameter tuning, practical optimality or tradeoff is easily obtained. It is obvious to the users that, increasing bandwidth from low to high, the tracking and disturbance rejection are improved, but at the costs of increased sensitivity to measurement noises, the larger amount of energy exerted, and the reduced stability margin. Seeing both sides, it will not be hard for the user to choose a compromise.

PID Tuning

In PID tuning, we strive for fairness in comparison. Since PID is usually tuned by experience in practice, in a time-consuming process, duplicating that in our simulation is challenging. Instead, we take advantage of the MATLAB embedded PID autotuning tool to get the optimal coefficient value of K_p, K_i, and K_d. The MATLAB PID autotuner is a tool capable of computing the parameters of a regulator connected to the VR circuit automatically, without major user interaction apart from initiating the operation. The autotuner avoids tuning a PID regulator manually, which is not consistent and may not be optimal. The basic steps of a tuning process of the autotuner may be summarized as follows:

1. Observing the process behavior, eventually stimulating it somehow and turning this knowledge into a description of the process behavior
2. Establishing the desired closed loop behavior on the basis of the obtained process description
3. Computing the PID controller parameters in order to achieve the desired closed loop behavior.

Comparison Results

The comparison simulation result is shown in Figure 8. The top chart is for output voltage of the VR circuit, the second chart from top is the cyclic load, which simulates the CPU load frequent changes applied to the VR. The third

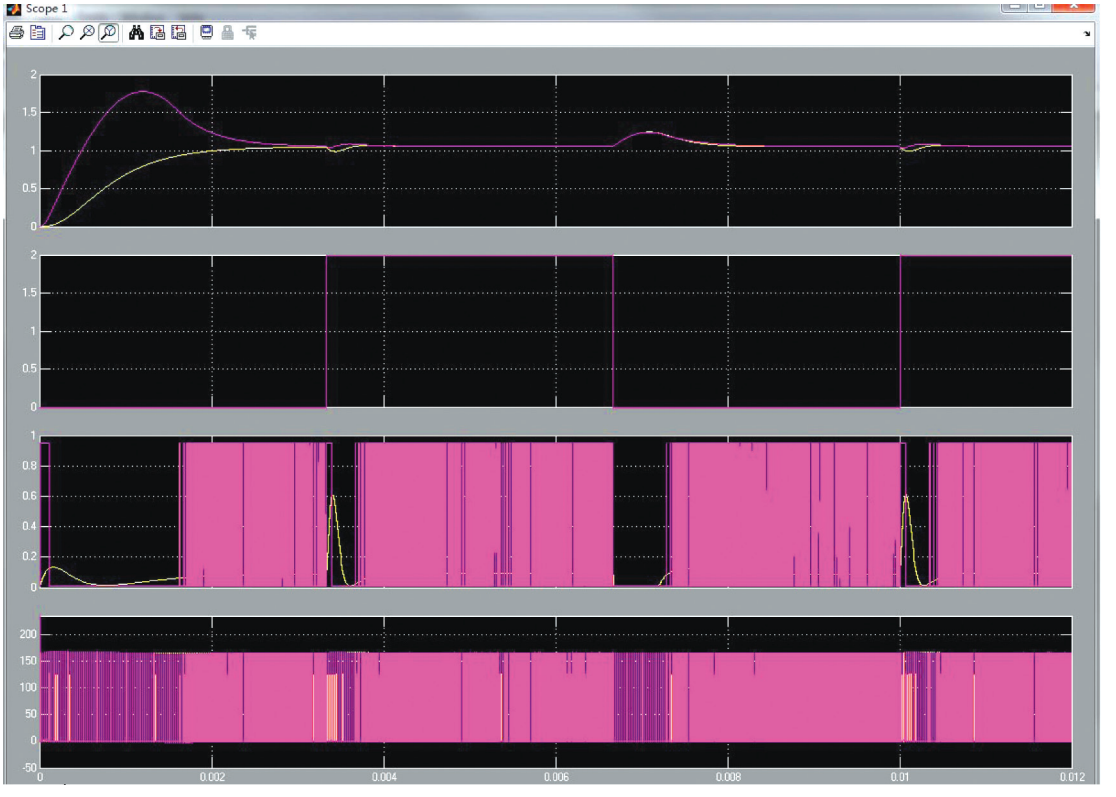


Figure 8: Comparison simulation result
(Source: Intel Corporation, 2012)

chart is the control signal output (PWM duty ration) from each controller. The bottom chart is the output current applied to the CPU. The purple line is PID, and the yellow line is ADRC.

Based on the comparison, we made the following observations:

1. For both ADRC and PID control, the output voltage all reach to the desired value 1.05 V after start transient and reach to steady state.
2. With the load step change, both ADRC and PID can correct the voltage back to 1.05 V with small overshoot or undershoot.
3. The major differences between ADRC and PID are the control signal output. The ADRC control is smooth and only acts when it is needed. PID does a busy control and it is very hard to maintain the output voltage at the same 1.05 V. Theoretically, the more efficient control will result in power savings, and we will look at how power saving ADRC can be provided quantitatively in the next step.

Energy Consumption Comparison between ADRC and PID

Figure 9 plots the integration of the input power to the VR circuit with both ADRC and PID control method; the integration of the power over time is the energy consumption. The energy consumption (yellow line)

"Compare the Integration of the power over time between ADRC and PID control method."

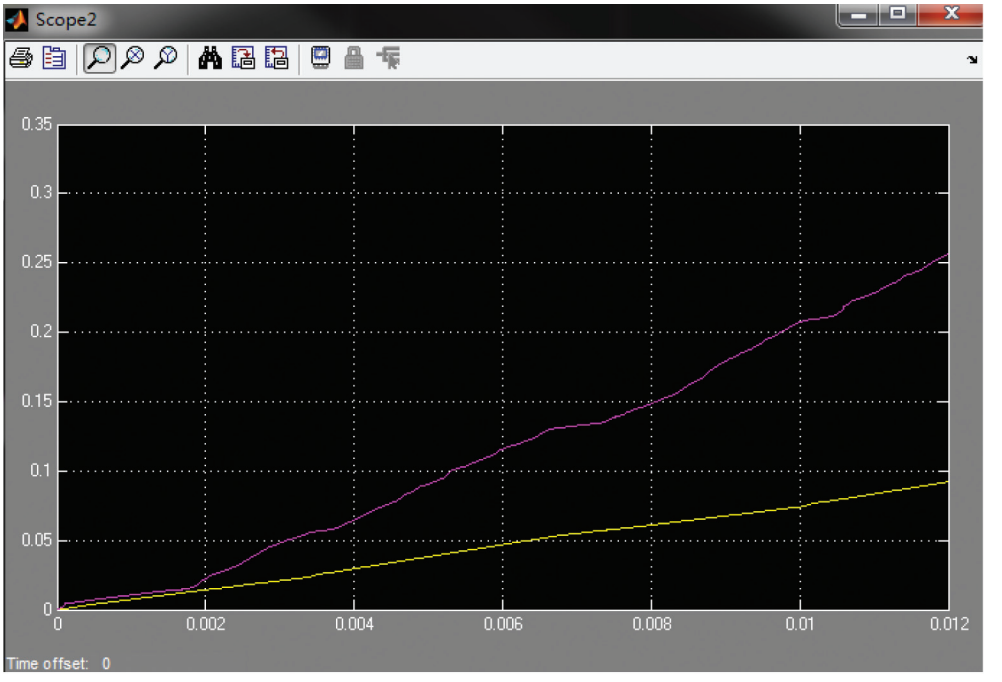


Figure 9: Energy consumption comparison between ADRC and PID (Source: Intel Corporation, 2012)

with ADRC control method (purple line) is obviously less than the energy consumption with the PID control method. With the time last, the gap of energy consumption between ADRC and PID is significant. The energy consumption is calculated between time 0.0025 seconds and 0.012 seconds. The reason to choose 0.0025 seconds as the start time is because at 0.0025 seconds it has reached steady state after transient for both ADRC and PID to make a fair comparison.

Table 1 shows the quantitative energy consumption different between the ADRC and PID while the output voltage IAE between the ADRC and PID are about the same.

	Energy Consumption (Watt X second) (input voltage 12 V)	**IAE**
ADRC	0.0919	3.3927e-04
PID	0.2358	3.3784e-04

Table 1: Energy Consumption Comparison between ADRC and PID (Source: Intel Corporation, 2012)

"ADRC control method save major power versus PID control method."

Based on the data shown in Table 1, ADRC saves about 68 percent energy versus the PID control method for this CPU VTT VR circuit.

Power Saving Estimation at System Level

The simulation timespan for the above data is 0.0095 second, so assuming the same cyclic load is applied to the VR, we can derive what the power consumption is in hours, days, and a year. Table 2 gives a comparison about the energy consumption for various time spans. In a year, only the ADRC in the single VTT VR controller will save about 131.4 kWh of energy for the server. If the same control methodology applied to each VR in the server, and in a data center, the energy and cost saving would be tremendous.

"Scale the energy saving to a year."

	1 hour	1 day	1 year	1 year energy saving per VR
ADRC	0.0097 kWh	0.23 kWh	83.95 kWh	131.4 kWh
PID	0.0248 kWh	0.59 kWh	215.35 kWh	

Table 2: Energy Saving For Various Timespans
(Source: Intel Corporation, 2012)

Summary

Design principles pertaining to control systems in server subsystems are examined in this article to distinguish two different paradigms: the reactive PID and active disturbance rejection. It is shown how the ADRC principle can be systematically applied to facilitate advanced control development for server subsystems. One class of such subsystems, the CPU VR control, is used to illustrate how the concept fits and how the corresponding control algorithm is developed and validated in simulation, with encouraging results. Much work is ahead to further test the concept in hardware implementation and in the expansion of the investigation into other Server subsystems.

"Design principles pertaining to control systems in server subsystems are examined in this article to distinguish two different paradigms: the reactive PID and active disturbance rejection."

Acknowledgments

The authors would like to thank Ms. Qinling Zheng for her assistance in simulation.

References

[1] U.S. EPA, "Report to congress on server and data center energy efficiency," Tech. Rep., Aug. 2007.

[2] J.G. Koomey, "Estimating Total Power Consumption by Servers in the U.S. and the World."

[3] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony, "The case for power management in web servers," Power Aware Computing, Jan 2002.

[4] Intel, "First the Tick, Now the Tock: Intel Microarchitecture (Nehalem)," 2009.

[5] L. Barroso and U. Hölzle, "The case for energy-proportional computing," IEEE Computer, Jan 2007.

[6] K. J. Astrom and T. Hagglund, "PID Control," The Control Handbook, W.S. Levine, Ed. p. 198, CRC Press 1996.

[7] N. Minorsky, "Directional Stability and Automatically Steered Bodies," J. Am. Soc. Nav. Eng., vol. 34, 1922, p. 280.

[8] Z. Gao "On Disturbance Rejection Paradigm in Control Engineering," Proceedings of the 2010 Chinese Control Conference, July 29–31, 2010, Beijing, China.

[9] G. Tian, Z. Gao, "From Poncelet's Invariance Principle to Active Disturbance Rejection," Proceedings of the 2009 American Control Conference, June 10–12, 2009, pp. 2451–2457.

[10] Z. Gao, "Active Disturbance Rejection Control: A Paradigm Shift in Feedback Control System Design," Proc. of the 2006 American Control Conference, Minneapolis, June 14–16, 2006, pp. 2399–2405.

[11] Z. Gao, "Scaling and Parameterization Based Controller Tuning," Proc. of the 2003 American Control Conference, June 2003, pp. 4989–4996.

[12] B. Sun and Z. Gao, "A DSP-Based Active Disturbance Rejection Control Design for a 1 kW H-Bridge DC-DC Power Converter," IEEE Transactions on Industrial Electronics. Volume 52, Issue 5, pp.1271–1277, Oct. 2005.

[13] Q. Zheng and Z. Gao, "On Practical Applications of Active Disturbance Rejection Control," Proceedings of the 2010 Chinese Control Conference, July 29–31, 2010 Beijing, China.

[14] G. Tian, Z. Gao, "Frequency Response Analysis of Active Disturbance Rejection Based Control System," Proceedings of the 2007 IEEE Multi-conference on Systems and Control, Singapore, October 1–3, 2007.

[15] Z. Ping and Z. Gao, "An FPGA-Based Digital Control and Communication Module for Space Power Management and Distribution Systems," Proceedings of the 2005 American Control Conference, Portland, June 8–10, 2005.

[16] Stuart Bennet, "The Past Of PID Controllers," Annual Reviews in Control 25 (2001) pp. 43–53

[17] Nobert Wiener, *Cybernetics*, Wiley, New York, 1948.

[18] G. Schipanov, "Theory and methods of designing automatic regulators," Automatika in Telemekhanika, vol. 4, no. 1, pp. 49–66, 1939.

[19] B. N. Petrov, "The invariance principle and the conditions for its application during calculation of linear and nonlinear systems," Proc. Of 1st IFAC Congress, Moscow, USSR, Butterworth. Ltd., London, England, vol. 1, pp. 117–126; 1961.

[20] William Ross Ashby, *An Introduction to Cybernetics*, Wiley, New York, 1956.

[21] J. Han, "Control theory: Model approach or control approach," Syst. Sci. Math., vol. 9, no. 4, pp. 328–335, 1989, (in Chinese).

[22] Roger Brockett, "New Issues in the Mathematics of Control," in *Mathematics Unlimited – 2001 and Beyond*, B. Engquist and W. Schmid Ed., pp. 189–220, Springer, 2001.

[23] J. Han, "A class of extended state observers for uncertain systems," *Control and Decision*, vol. 10, no. 1, pp. 85–88, 1995, (in Chinese).

Author Biographies

John Ping is a system architect in the Intel Data Center and Connected Systems Group. He has a Bachelor of Technology degree in electrical engineering from East China University of Science and Technology (Shanghai), and a master of science in electrical engineering and a doctor of engineering degree from Cleveland State University. His expertise and research interests span server system architecture, optimization of performance, power and energy in server nodes, racks and IP data centers. Another expertise and research area is advanced control algorithms. He can be reached at john.ping@intel.com

Zhiqiang Gao is associate professor of electrical engineering and director of the Center for Advanced Control Technologies at Cleveland State University. He received his PhD in electrical engineering from the University of Notre Dame in 1990. Employing an experimental science philosophy to research and a humanistic orientation to teaching, Dr. Gao and his team of researchers bring creative solutions to real world control problems and vitality of thinking to the young minds. He spent seventeen years developing ADRC from an obscure concept into a proven industrial control solution.

Rahul Khanna is a platform architect at Intel Corporation involved in development of energy efficient algorithms. Over the past 17 years he has worked on server system software technologies including platform automation, power/thermal optimization techniques, reliability, optimization, and predictive methodologies. He has authored several technical papers and book chapters in the areas related to energy optimization, platform wireless interconnects, sensor networks, interconnect reliability, predictive modeling, motion estimation, and security and holds 27 patents. He is also the co-inventor of the Intel IBIST methodology for high-speed interconnect testing. His research interests include machine learning based power/thermal optimization algorithms, narrow-channel high-speed wireless interconnects and information retrieval in dense sensor networks. Rahul is member of IEEE and the recipient of three Intel Achievement Awards for his contributions in areas related to advancements of platform technologies. He is the author of book *A Vision for Platform Autonomy: Robust Frameworks for Systems*. Rahul Khanna can be reached at rahul.khanna@intel.com

ASYMMETRICAL AND LOWER BOUNDED SUPPORT VECTOR REGRESSION FOR POWER PREDICTION

Contributors

Mel Stockman
American University of Beirut

Mariette Awad
American University of Beirut

Rahul Khanna
Intel Corporation

The successful design and evaluation of autonomous energy optimization techniques requires the availability of a ubiquitous and accurate set of measurement techniques that are cheap and easy to implement. We discuss an approach for mathematically estimating the wall power as well as the power of the principal functional units (like DRAM) in the server platforms without incurring the cost of hardware instrumentation. Support Vector Regression (SVR) has proven to be an effective tool in real value function estimation. In this paper we modify two loss functions, Vapnik's ε-insensitive loss function and an insensitive Huber loss function to be asymmetrical in order to limit underestimates. Our novel approach, asymmetrical support vector regression, provides accurate prediction while maintaining a low number of out of bounds misestimates. We test our approach on two different datasets by predicting the power for the next time interval and achieve accuracy rates of below 6 percent relative percentage error while keeping the number of boundary misestimates below 4 percent.

Introduction

Accurate power measurement at system and sub-component level allows a predictive analysis of energy consumption for an optimal efficiency of a platform. This helps in estimating the peak consumption and helps minimize system failures by allowing sufficient safety margins. Power metering can be established using an extensive network of instrumented power sensors (such as instrumented power supply units). While instrumentation solves the problem, it can be expensive to build and requires an extensive network of system interconnects with certain expectation of accuracy, bandwidth, and response time. High accuracy translates into high linearity of an analog sensor over a large range that can easily become expensive. Since sensors perform linearly only within a limited operating range, it makes them inaccurate in the regions of high or low currents. Further, environmental and electrical variations (temperature, humidity, environmental impurities, and electro-migration) and aging can cause inaccuracies over time, which will require a frequent recalibration of electrical components. At the same time, while meters can easily be built at a physical component level (CPU, memory, and power supply); they are rather difficult to build at the virtual level. For example, in a multi-partition system, instrumentation is rendered useless if all the partitions are powered by a single power supply unit with a single instance of power instrumentation. We discuss an approach to power metering by estimation using a set of observed variables that share linear or nonlinear correlation to the power consumption. This approach exploits the statistical relationship among potential variables and power consumption through predictive approximation.

"We discuss an approach to power metering by estimation using a set of observed variables that share linear or nonlinear correlation to the power consumption."

In many instances of approximation there is an uneven consequence of misprediction based on whether the error is above or below the target value. For instance, in power prediction, an incorrect low estimate may be of much more concern than an overestimate. Underpredicting could lead to insufficient cooling of data centers, inadequate UPS power supply, unavailable processor resources, needlessly powering down chip components, and so on. In the case of forest fire behavior prediction, a lower estimate of the threat could lead to additional property damage as well as loss of life due to a lack of adequate supply of personnel and equipment in response. In such circumstances, it is critically necessary to eliminate or strictly limit underestimating a function. It is preferable to relax accuracy constraints in order to decrease the likelihood that the estimation falls below certain bounds as necessitated by the application. In these cases, it is crucial to minimize misestimates on one side of a boundary even at the risk of reducing the accuracy of the entire estimation. It is necessary to restrict the loss function so that only a minimal number of under- or overestimates occur. This leads to an asymmetric loss function for training whereby a greater penalty is applied when the misestimate on the wrong side of the boundary.

"It is preferable to relax accuracy constraints in order to decrease the likelihood that the estimation falls below certain bounds as necessitated by the application."

Unlike other approaches, which predict power for the current time interval, we predict the power usage for the next time interval. This is preferable in cases where online configuration may take time.

The remainder of this article is organized as follows: The next section discusses prior research. This is followed by a brief overview of standard support vector regression (SVR). Next, we explain our asymmetrical approach (ALB-SVR). This is followed be an explanation of the six different data sets used. Then we present our experimental results and compare SVR and ALB-SVR accuracies, followed by our conclusion.

Prior Research

Although we are not aware of prior work specifically addressing our approach, we survey in this section some related work available in literature. Seok et al.[2] used an asymmetric ε-insensitive loss function in support vector quantile regression (SVQR) in an attempt to decrease the number of support vectors (SV). They altered the insensitivity according to the quantile and achieve a sparser model. Our work differs from theirs in that their aim was to decrease the number of SV while maintaining the same accuracy as a regular SVQR, while our approach specifically seeks to limit underestimates at the possible cost to accuracy. Asymmetrical loss functions are discussed by Schabe[3], who studies different loss functions for Bayes parameter estimation. Schabe compared a two-sided quadratic loss function to a quasi-quadratic s-loss function and showed that the modified version offers a smaller increase of loss and can be used in real world situations where overestimation and underestimation have different importance. Norstrom[4] studied Bayesian risk analysis and replaced the quadratic loss function with an asymmetric loss function to derive a general class of functions that approach infinity near the

origin to limit underestimates. Saketha Nath and Bhattacharyya[5] presented a maximum margin classifier that bounds misclassification for each class differently, thus allowing for different tolerances levels. Lee et al.[6] proposed a smoothing strategy to modify the typical SVR approach into a non-constrained problem, thereby only solving a system of linear equations rather than a convex quadratic program. Jeh-Nan Pan and Jianbiao Pan[7] compared three different loss functions for economic tolerance design: Taguchi's quadratic loss function, Inverted Normal Loss Function and Revised Inverted Normal Loss Function.

Support Vector Regression

"The usual approach trains using a symmetrical loss function, which equally penalizes both high and low misestimates."

SVR as proposed by Vapnik[1] has proven to be an effective tool in real value function estimation. The usual approach trains using a symmetrical loss function, which equally penalizes both high and low misestimates. Using Vapnik's ε-insensitive approach, a flexible tube of minimal radius is formed symmetrically around the estimated function such that the absolute values of errors less than a certain threshold ε are ignored both above and below the estimate. In this manner, points outside the tube are penalized but those within the tube, either above or below the function, receive no penalty.

One of the main advantages of SVR is that its computational complexity does not depend on the dimensionality of the input space. Additionally it has excellent generalization capability with high prediction accuracy[6].

ε-insensitive Loss Function

In Vapnik's ε-insensitive SVR[6], a real value y is predicted as:

$$y_i = w \cdot x_i + b \qquad (1)$$
$$\{x_i, y_i\} \quad i = 1 \ldots L, \qquad y_i \in \Re, x \in \Re^D$$

using a tube bounded by $\pm \varepsilon \forall_i$ as shown in Figure 1. The penalty function is characterized by only assigning a penalty if the predicted value y_i is more than ε away from the actual value t_i, (i.e. $|t_i - y_i| \geq \varepsilon$). Those data points that lie outside the ε-tube are given the same penalty whether they lie above (ξ^+) or below (ξ^-) the tube ($\xi^+ > 0$, $\xi^- > 0 \ \forall_i$):

$$t_i \leq y_i + \varepsilon + \xi^+ \qquad (2)$$

$$t_i \geq y_i - \varepsilon - \xi^- \qquad (3)$$

The accuracy of the estimation is then measured by the loss function $L_{\varepsilon SVR}(t, y)$ as shown in Figure 2:

$$L_{\varepsilon SVR}(t, y) = \begin{cases} 0 & if \ |t - y| \leq \varepsilon \\ |t - y| - \varepsilon & otherwise \end{cases} \qquad (4)$$

The empirical risk is:

$$R_{emp}(y) = \frac{1}{L} \sum_{i=1}^{L} L_{\varepsilon SVR}(t_i, y_i) \qquad (5)$$

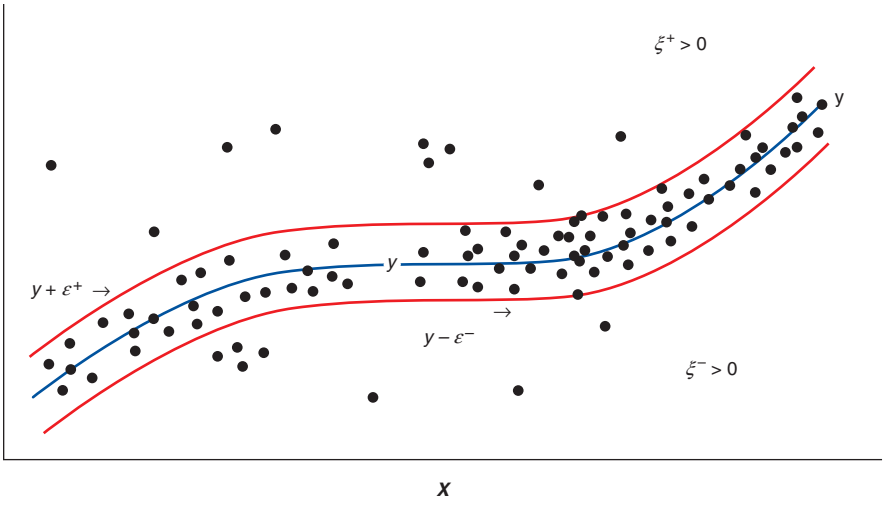


Figure 1: SVR with ε-insensitive tube
(Source: American University of Beirut, 2012)

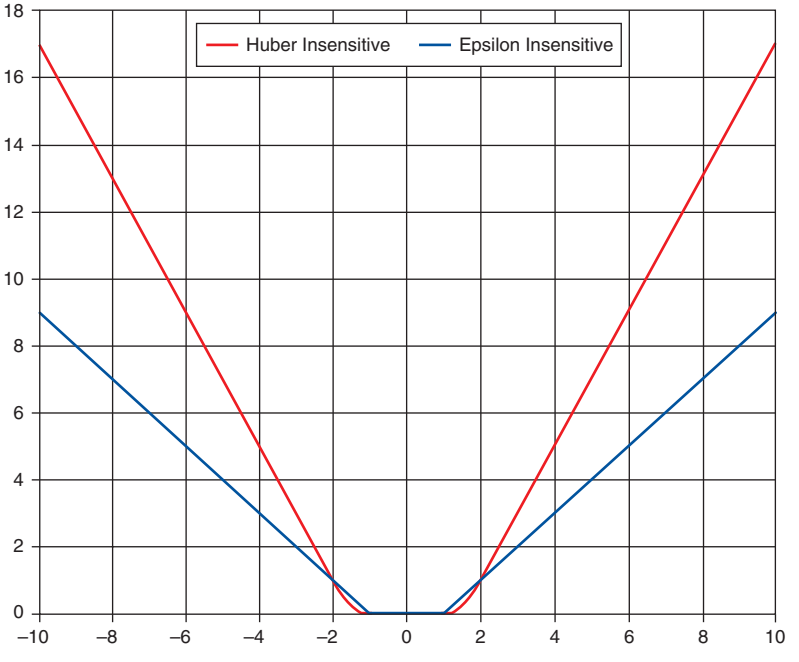


Figure 2: ε-insensitive and Insensitive Huber loss functions with $\varepsilon = 1$ $\partial = 2$
(Source: American University of Beirut, 2012)

leading to the SVR error function:

$$C\sum_{i=1}^{L}(\xi_i^+ + \xi_i^-) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{6}$$

which should be minimized subject to the constraints $\xi^+ \geq 0$, $\xi^- \geq 0\ \forall_i$ and equations 2 and 3. Support vectors (SVs) are those points that lie outside the ε-tube. The computational efficiency of the SVM lies in the fact that only the linear combination of the SVs is used for the solution so that large feature data sets do not affect the SVR[12].

"The computational efficiency of the SVM lies in the fact that only the linear combination of the SVs is used for the solution so that large feature data sets do not affect the SVR."

Huber Insensitive Loss Function

The Huber insensitive loss function, as proposed by Bo et al.[11] and shown in Figure 2, is similar to the ε-insensitive loss function; however it increases quadradically for small errors outside the epsilon bound but below a certain threshold $\partial > \varepsilon$ and then linearly beyond ∂, making it robust with regards to outliers.

$$L_{\varepsilon\partial HuberSVR}(t,y) = \begin{cases} 0 & if\ |t-y| \leq \varepsilon \\ (|t-y|-\varepsilon)^2 & if\ \varepsilon < |t-y| < \partial \\ (\partial-\varepsilon)(2|t-y|-\partial-\varepsilon) & if\ |t-y| \geq \partial \end{cases} \tag{7}$$

Asymmetrical and Lower Bound Support Vector Regression

Our approach, asymmetrical and lower bound support vector regression (ALB-SVR), modifies the SVR loss functions and corresponding error functions such that the epsilon tube is only above the function as shown in Figure 3 and Figure 4. The penalty parameter C is split into C^+ and C^- so that different penalties can be applied to the upper and lower mispredictions. We apply this technique to both the ε-insensitive and the Huber insensitive loss functions.

ε-insensitive ALB-SVR

For the ε-insensitive loss function equations 3, 4, and 6 are modified as follows:

$$t_i \geq y_i - \xi^- \tag{8}$$

$$L_{\varepsilon-ALB-SVR}(t,y) = \begin{cases} 0 & if\ 0 \leq (t-y) \leq \varepsilon \\ (t-y)-\varepsilon & if\ (t-y) > \varepsilon \\ (y-t) & otherwise \end{cases} \tag{9}$$

$$C^+\sum_{i=1}^{L}\xi_i^+ + C^-\sum_{i=1}^{L}\xi_i^- + \frac{1}{2}\|\mathbf{w}\|^2 \tag{10}$$

Introducing Lagrange multipliers: $\alpha_i^+ \geq 0$, $\alpha_i^- \geq 0$, $\mu_i^+ \geq 0$, $\mu_i^- \geq 0\ \forall_i$

$$L_p = C^+ \sum_{i=1}^{L} \xi_i^+ + C^- \sum_{i=1}^{L} \xi_i^- + \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{L}(\mu_i^+ \xi_i^+ + \mu_i^- \xi_i^-) - \sum_{i=1}^{L} \alpha_i^+ (\varepsilon + \xi_i^+ + y_i - t_i)$$
$$- \sum_{i=1}^{L} \alpha_i^- (\xi_i^- - y_i + t_i) \quad (11)$$

which leads to:

$$\frac{\delta L_p}{\delta \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-)\mathbf{x}_i \quad (12)$$

$$\frac{\delta L_p}{\delta b} = 0 \Rightarrow \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) = 0 \quad (13)$$

$$\frac{\delta L_p}{\delta \xi_i^+} = 0 \Rightarrow C^+ = \alpha_i^+ - \mu_i^+ \quad (14)$$

$$\frac{\delta L_p}{\delta \xi_i^-} = 0 \Rightarrow C^- = \alpha_i^- - \mu_i^- \quad (15)$$

Substituting equations 12 and 13 and maximizing L_D with respect to α_i^+ and α_i^- ($\alpha_i^+ \geq 0$, $\alpha_i^- \geq 0$ $\forall_i$) where:

$$L_D = \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) t_i - \varepsilon \sum_{i=1}^{L} \alpha_i^+ - \sum_{i=1}^{L} \alpha_i^- - \frac{1}{2} \sum_{i,j} (\alpha_i^+ - \alpha_i^-)(\alpha_i^+ - \alpha_i^-)\, \mathrm{x}_i\, \mathrm{x}_j \quad (16)$$

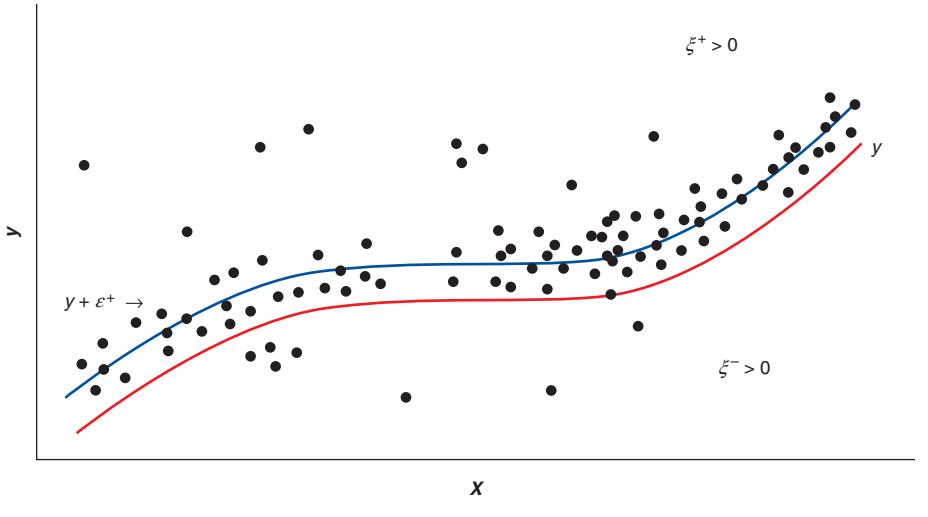


Figure 3: ALB-SVR with ε-insensitive tube
(Source: American University of Beirut, 2012)

Since $\mu_i^+ \geq 0$ and $\mu_i^- \geq 0$ and equations 14 and 15, therefore $\alpha_i^+ \leq C^+$ and $\alpha_i^- \leq C^-$. Thus we need to find

$$\max_{\alpha^+, \alpha^-} \begin{bmatrix} \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) t_i - \varepsilon \sum_{i=1}^{L} \alpha_i^+ - \sum_{i=1}^{L} \alpha_i^- \\ -\frac{1}{2} \sum_{i,j} (\alpha_i^+ - \alpha_i^-)(\alpha_i^+ - \alpha_i^-)\, \mathrm{x}_i \cdot \mathrm{x}_j \end{bmatrix} \quad (17)$$

with $0 \leq \alpha_i^+ \leq C^+$, $0 \leq \alpha_i^- \leq C^-$ and $\sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) = 0 \forall_i$.

Substituting equation 12 into equation 1

$$y' = \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) x_i \cdot x_i' + b \tag{18}$$

Support Vectors x_S can be found with the indices where $0 < \alpha^+ < C^+$ and $0 < \alpha^- < C^-$ and $\xi_i^+ = 0 (\text{or } \xi_i^- = 0)$

and b can be derived by:

$$b = t_S - \varepsilon - \sum_{m \varepsilon S}^{L} (\alpha_m^+ - \alpha_m^-) x_m \cdot x_S' \tag{19}$$

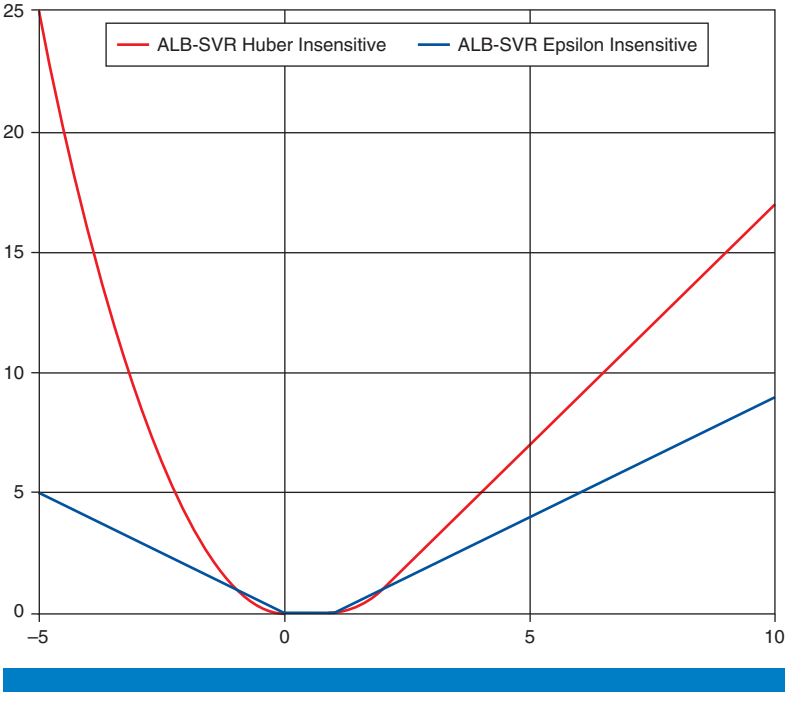


Figure 4: ALB-SVR ε-insensitive and ALB-SVR insensitive Huber loss functions with $\varepsilon = 1$ $\partial = 2$
(Source: American University of Beirut, 2012)

$$L_{\varepsilon \partial Huber ALB-SVR}(t, y) = \begin{cases} 0 & if\ 0 \geq (t - y) \leq \varepsilon \\ (t - y)^2 & if\ (t - y) < 0 \\ ((t - y) - \varepsilon)^2 & if\ \varepsilon < (t - y) < \partial \\ (\partial - \varepsilon)(2 | t - y | - \partial - \varepsilon) & if\ | t - y | \geq \partial \end{cases} \tag{20}$$

with solution given by:

$$\max_{\alpha^+,\, \alpha^-} \begin{bmatrix} \sum_{i=1}^{L} (\alpha_i^+ - \alpha_i^-) t_i - \frac{1}{2C} \sum_{i=1}^{L} (\varepsilon \alpha_i^{+2} - \alpha_i^{-2}) \\ -\frac{1}{2} \sum_{i,j} (\alpha_i^+ - \alpha_i^-)(\alpha_i^+ - \alpha_i^-) \mathrm{x}_i \cdot \mathrm{x}_j \end{bmatrix} \tag{21}$$

and the resulting optimization problem:

$$\min_{\alpha^+,\alpha^-}\left[\begin{array}{c} -\sum_{i=1}^{L}(\alpha_i^+-\alpha_i^-)t_i+\frac{1}{2C}\sum_{i=1}^{L}(\varepsilon\alpha_i^{+2}-\alpha_i^{-2}) \\ +\frac{1}{2}\sum_{i,j}(\alpha_i^+-\alpha_i^-)(\alpha_i^+-\alpha_i^-)\mathrm{x}_i\cdot\mathrm{x}_j \end{array}\right] \quad (22)$$

$$-C\le(\alpha_i^+-\alpha_i^-)\le C \qquad i=1..L \quad (23)$$

$$\sum_1^L(\alpha_i^+-\alpha_i^-)=0 \quad (24)$$

Training Data Sets

Two different power data sets were used for our experimentation. These data sets are used for training function. In our experiments we used *CPU power telemetry data sets* and *DRAM activity data sets* to estimate system power and DRAM power respectively.

"In our experiments we used CPU power telemetry data sets *and* DRAM activity data sets *to estimate system power and DRAM power respectively."*

CPU Power Telemetry Data Set

Power telemetry data set is accessed through power telemetry harness (PCI based Gladiator telemetry Harness (GTH) Card) externally connected to Intel® Xeon® server class dual-socket platform. The data set consists of 640 samples of 6 attributes of telemetry data from a distributed set of physical sensors as shown in Table 1 along with the measured system power (mW). This dataset is then used to train the model that *predicts the system power*.

CPU1 Vtt1	Termination, misc. I/O power
CPU1 Vcc1	Core power
CPU1 Vsa	System agent, Uncore, I/O power
CPU2 Vtt1	Termination, misc. I/O power
CPU2 Vcc1	Core power
CPU2 Vsa	System agent, Uncore, I/O power

Table 1: Gladiator Data Set Attributes
(Source: Intel Corporation, 2012)

DRAM Activity Data Set

The data set taken from David et al.[8] and Stockman et al.[9] consists of 17765 samples of 5 attributes of memory activity counters as described in Table 2 with the actual corresponding power consumed in watts as measured directly by a memory power riser. This dataset is then used to train the model that *predicts the DRAM power*.

Activity	**Units**
Activate(A)	nj/Activate
Read (R)	nj/Read
Write (W)	nj/Write
CKE=High	mW
CKE=Low	mW

Table 2: Memory Power Model Attributes
(Source: Intel Corporation, 2012)

Experiments and Results

We modified the code in LIBSVM[10] for ALB-SVR. For all experiments, we normalized the data and took the average of 10 runs of threefold cross-validation. Using an RBF kernel, we performed a grid search combined with heuristic experimentation for both SVR and ALB-SVR to find the best meta-parameters ε, g, C^+ and C^-.

Table 3 and Figures 5 through 12 show the results of SVR and ALB-SVR for both the loss functions. As can be seen, the number of underestimates is

c	Type	C+	C–	g	ε	∂	% error	% out of bound
CPU	ε-insensitive	512	—	16	0.23		1.72	50.33
CPU	ε-insensitive ALB-SVR	32768	32	1	0.00039		5.72	2.81
CPU	Huber Insensitive SVR	10000	—	1	0.00039	0.01	1.45	50.86
CPU	Huber Insensitive ALB-SVR	10000	100	1	0.00039	0.01	5.33	3.58
DRAM	ε-insensitive	512	—	706	0.10		1.82	57.54
DRAM	ε-insensitive ALB-SVR	1000000	10	706	0.20		5.06	1.74
DRAM	Huber Insensitive SVR	512	—	128	0.1	1.0e-06	1.03	67.07
DRAM	Huber Insensitive ALB-SVR	10000000	1000	128	0.1	1.0e-06	1.50	0.24

Table 3: Comparative Results of SVR versus ALB-SVR
(Source: Intel Corporation, 2012)

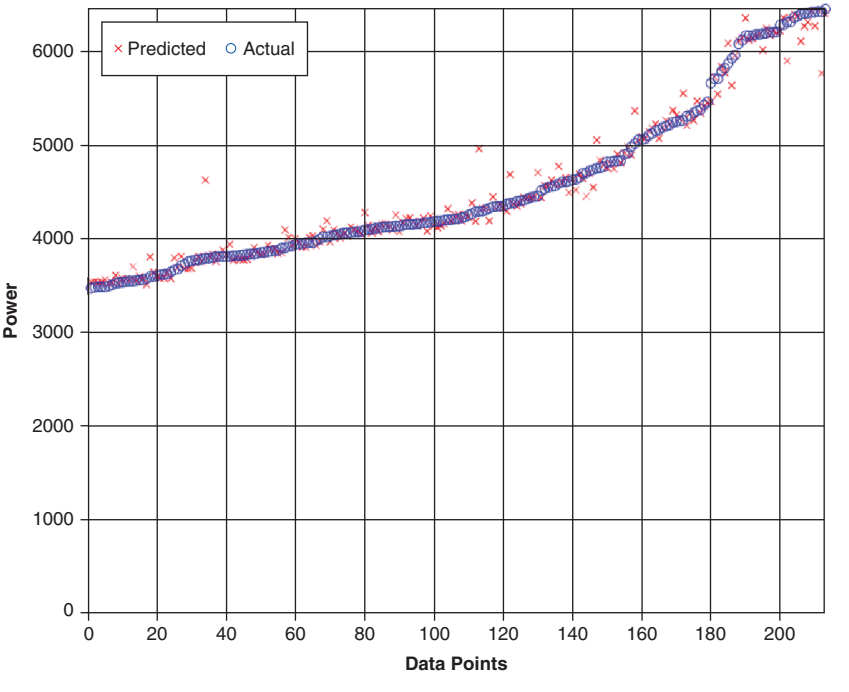


Figure 5: System power estimates using CPU training data (Table 2) with ε-insensitive SVR
(Source: American University of Beirut, 2012)

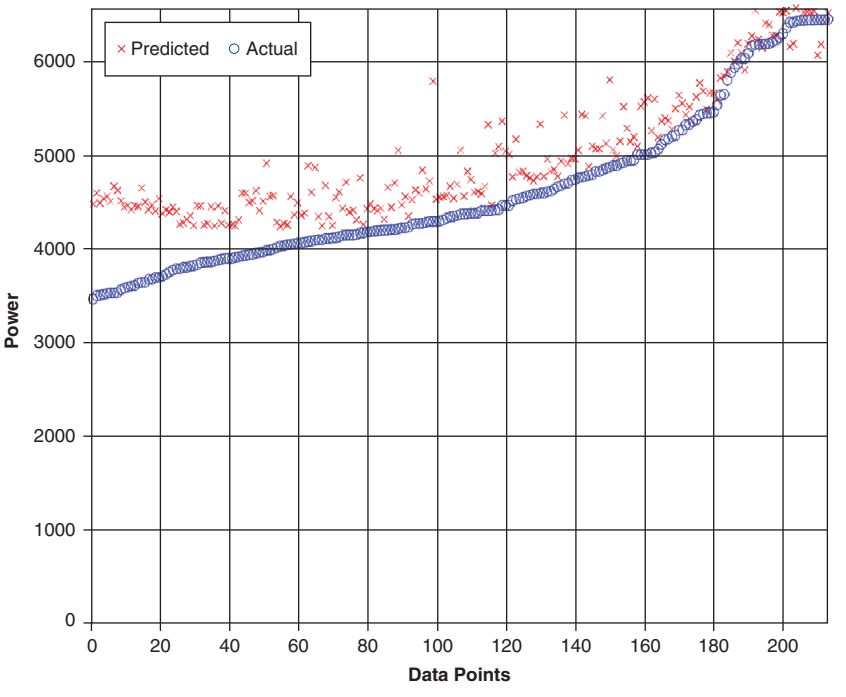


Figure 6: System power estimates using CPU training data (Table 2) with ε-insensitive ALB-SVR
(Source: Intel Corporation, 2012)

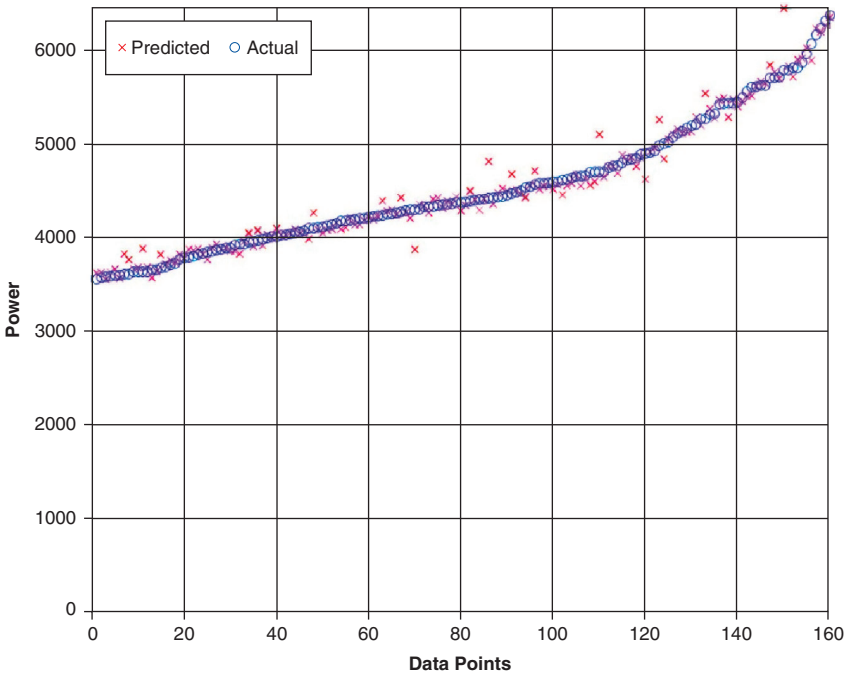


Figure 7: System power estimates using CPU training data (Table 2) with Huber insensitive SVR
(Source: Intel Corporation, 2012)

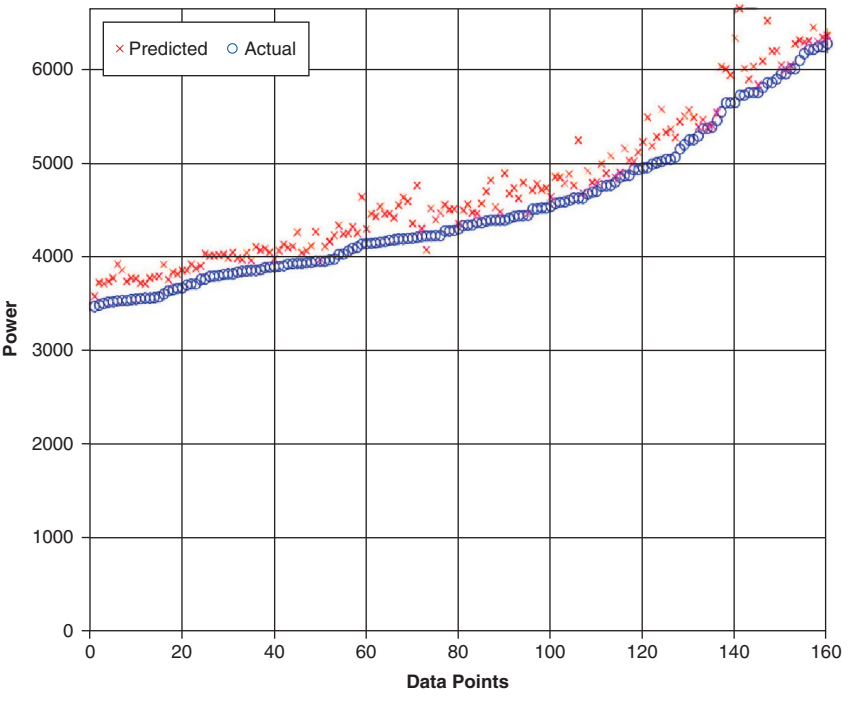


Figure 8: System power estimates using CPU training data (Table 2) with Huber insensitive ALB-SVR
(Source: Intel Corporation, 2012)

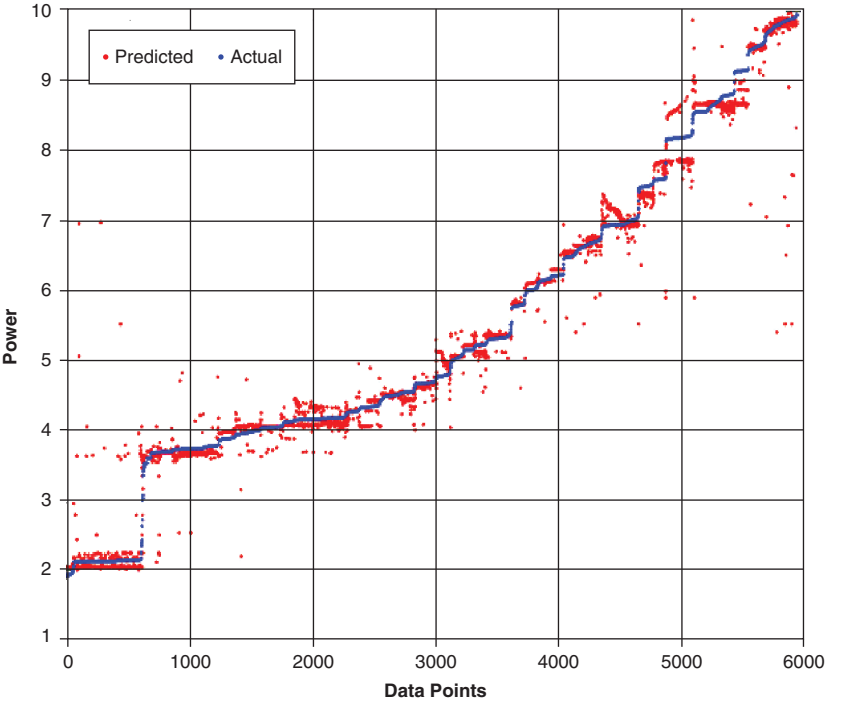


Figure 9: DRAM power estimates using DRAM activity data (Table 3) with ε-insensitive SVR
(Source: Intel Corporation, 2012)

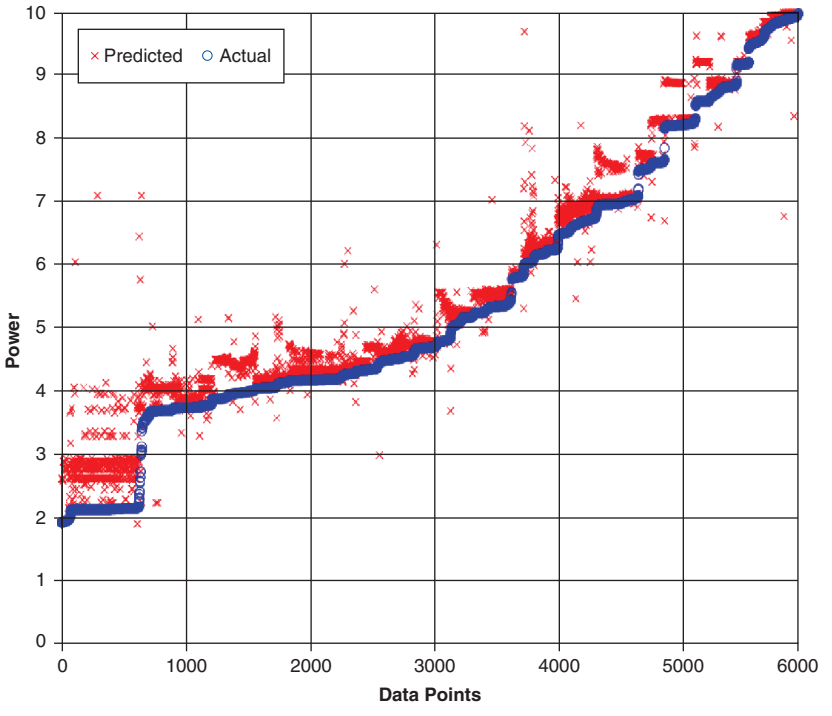


Figure 10: DRAM ower estimates using DRAM activity data (Table 3) with ε-insensitive ALB-SVR
(Source: Intel Corporation, 2012)

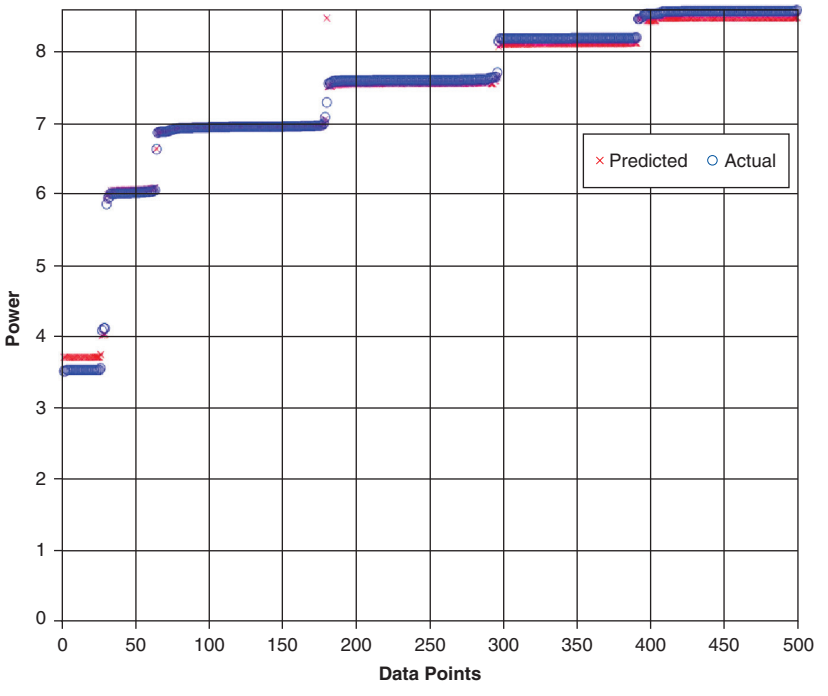


Figure 11: DRAM power estimates using DRAM activity data (Table 3) with Huber insensitive SVR
(Source: Intel Corporation, 2012)

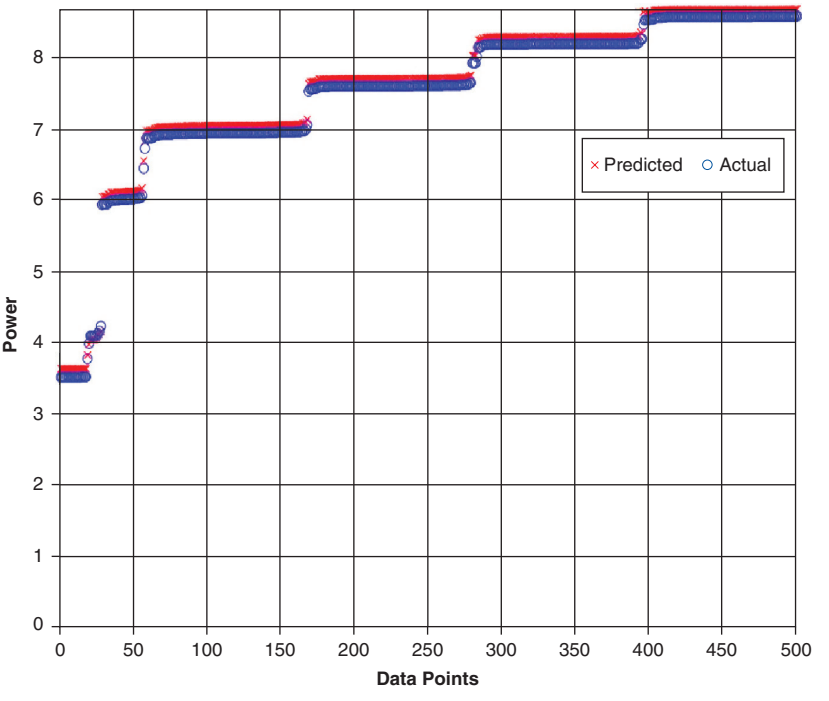


Figure 12: DRAM Power estimates using DRAM activity data (Table-3) with Huber insensitive ALB-SVR
(Source: Intel Corporation, 2012)

"ALB-SVR positions the half tube under the data so that only a small number of points fall below the estimate."

around 50 percent for the SVR, which is because SVR centers the epsilon tube around the data. ALB-SVR positions the half tube under the data so that only a small number of points fall below the estimate. The accuracy of ALB-SVR is necessarily less than that of SVR since the estimation is now skewed lower.

Model performance is evaluated by computing percentage relative error as:

$$E = \frac{1}{L}\sum_{i=1}^{L}\left|\frac{t_i - \hat{y}_i}{t_i}\right|100 \tag{25}$$

The relative error for estimating system power using CPU Power data set was 5.72 percent and for estimating DRAM power using DRAM Activity data set it was 5.06 percent. This is acceptable since we have minimized the number of underestimates. As also can be seen, the number of support vectors are greater in ALB-SVR than in SVR.

Comparison of SVR and ALB-SVR

Comparing ALB-SVR to SVR allows us to look at the tradeoffs involved with using this technique.

Empirical Risk

By substituting the new loss function, ALB-SVR's empirical risk becomes:

$$R_{emp}(y) = \frac{1}{L}\sum_{i=1}^{L} L_{\varepsilon-ALB-SVR}(t_i, y_i) \quad (26)$$

The maximum additional empirical risk for ALB-SVR can be computed to be:

$$\sum_{i\in(y-t)\le\varepsilon}^{L}(y-t) + \sum_{i\in(y-t)>\varepsilon}^{L}\varepsilon \quad (27)$$

Number of Support Vectors and Convergence

In SVR, support vectors (SVs) are those points that lie outside the epsilon tube. The smaller the value of ε, the more points that lie outside the tube and hence the greater number of SVs. In ALB-SVR, we have essentially cut the epsilon tube in half. We no longer have the lower epsilon bound. Therefore, for the same g and epsilon parameters, more points lie outside the tube and there will be a larger number of SVs. This increase in the number of SVs indicates that using ALB-SVR has some negative effects on the complexity of the estimating function. However, as seen in Table 3, the CPU data set did not show a significant increase in SVs. This may be because the data set is relatively small. As also can be seen in Table 3, the number of iterations was smaller in ALB-SVR, indicating the algorithm converged faster and hence this may offset the larger number of SVs using this approach.

For our ALB-SVR model, we used a grid search and heuristics to determine optimal meta-parameters. We achieved the goal of limiting the underestimates to 2.71 percent for the CPU data set and 1.74 percent for the DRAM Activity data set as compared to 50.33 percent and 57.54 percent for SVR.

Conclusion and Future Work

We have shown our novel approach ALB-SVR to be an effective technique to bound an estimation such that underestimates are greatly limited. This comes at the expense of accuracy but nevertheless is helpful for applications that are highly sensitive to such mispredictions such as power estimation. We tested our approach on two different power data sets and achieved accuracy rates of below 6 percent relative percentage error while keeping the number of underestimates below 4 percent. Future work will include different data sets and techniques for more accurately selecting the meta-parameters as well as improving the error percentage.

"We have shown our novel approach ALB-SVR to be an effective technique to bound an estimation such that underestimates are greatly limited."

Acknowledgements

This work is partly supported by MER, a partnership between Intel Corporation and King Abdul-Aziz City for Science and Technology (KACST) to conduct and promote research in the Middle East and the University Research Board at the American University of Beirut.

References

[1] V. Vapnik. *Statistical Learning Theory,* (Wiley, New York, 1998).

[2] K Seok, D Cho, C. Hwang, and J. Shim, "Support vector quantile regression using asymmetric e-insensitive loss function," *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, vol.1, no., pp.V1-438–V1-439, June 2010.

[3] H. Schabe, "Bayes estimates under asymmetric loss," *Reliability, IEEE Transactions on*, vol.40, no.1, pp.63–67, Apr 1991.

[4] J.G. Norstrom, "The use of precautionary loss functions in risk analysis," *Reliability, IEEE Transactions on*, vol.45, no.3, pp.400–403, Sep 1996.

[5] J. Saketha Nath and C. Bhattacharyya., "Maximum Margin Classifiers with Specified False Positive and False Negative Error Rates," *Proceedings of SDM Conference, Minneapolis,* 2007.

[6] Yuh-Jye Lee, Wen-Feng Hsieh, and Chien-Ming Huang, "ε-SSVR: a smooth support vector machine for ε-insensitive regression," *Knowledge and Data Engineering, IEEE Transactions on*, vol.17, no.5, pp. 678–685, May 2005.

[7] Jeh-Nan Pan, Jianbiao Pan, "A Comparative Study of Various Loss Functions in the Economic Tolerance Design," *Management of Innovation and Technology, 2006 IEEE International Conference on*, vol.2, no., pp.783–787, June 2006.

[8] H. David, E. Gorbatov, U. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 14–15, August, 2010.

[9] M. Stockman, M. Awad, R. Khanna, C. Le, H. David, E. Gorbatov, and U. Hanebutte, "A Novel Approach to Memory Power Estimation Using Machine Learning," *International Conference on Energy Aware Computing (ICEAC)*, pp. 1–3, December, 2010.

[10] C. Chang and C. Lin, LIBSVM: a library for support vector machines, 2001. http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[11] L. Bo, L. Wang, and L. Jiao, "Recursive Finite Newton Algorithm for Support Vector Regression in the Primal," *Neural Computation*, v.19 n.4, p.1082–1096, April 2007.

[12] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Machine learning: a review of classification and combining techniques", *Artif Intell Rev,* vol. 26, pp. 159–190, 2006.

Author Biographies

Melissa Stockman is a PhD candidate at the American University of Beirut. Her research areas include computer architecture, machine learning, and support vector machines. She has a Bachelor of Arts degree in mathematics from New York University, New York, New York, and a Master of Science in computer science degree from the George Mason University, Fairfax, Virginia. She can be contacted at melissa.stockman1@gmail.com.

Dr. Mariette Awad is an assistant professor in the Electrical and Computer Engineering Department of the American University of Beirut. She received her PhD in electrical engineering from the University of Vermont in 2007 and she has been a visiting professor at Virginia Commonwealth University, Intel Mobile Group, and MIT. Prior to her academic position, she was with IBM System and Technology group in Vermont as a wireless product engineer. Over the years, her technical leadership and innovative spirit has earned her management recognition, several business awards, and multiple patents at IBM. Her current research interests include machine learning, data mining, data fusion, image recognition and ubiquitous computing. She can be reached at mariette.awad@aub.edu.lb.

Rahul Khanna is currently a platform architect at Intel Corporation involved in development of energy efficient algorithms. Over the past 17 years he has worked on server system software technologies including platform automation, power/thermal optimization techniques, reliability, optimization, and predictive methodologies. He has authored several technical papers and book chapters in the areas related to energy optimization, platform wireless interconnect, sensor networks, interconnect reliability, predictive modeling, motion estimation, and security and holds 27 patents. He is also the co-inventor of the Intel IBIST methodology for high-speed interconnect testing. His research interests include machine learning based power/thermal optimization algorithms, narrow-channel high-speed wireless interconnect and information retrieval in dense sensor networks. Rahul is member of IEEE and the recipient of three Intel Achievement Awards for his contributions in areas related to advancements of platform technologies. He is the author of book *A Vision for Platform Autonomy: Robust Frameworks for Systems*. Rahul Khanna can be reached at rahul.khanna@intel.com

WIRELESS INTERCONNECTS FOR FUTURE COMPUTING SYSTEMS

Contributors

Lingli Xia
Postdoctoral candidate at Oregon State University

Changhui Hu
Marvell Semiconductor

Stephen Redfield
Graduate Student at Oregon State University

Sirikarn Woracheewan
Graduate Student at Oregon State University

Rahul Khanna
Intel Corporation

Jay Nejedlo
Intel Corporation

Huaping Liu
Oregon State University

Patrick Chiang
Oregon State University

Wireless interconnects at low to medium speeds (below 1 Gbps) allow efficient state exchange (power states, thermal throttling states, performance states, component profile deviation), system process control (power states, tuning optimization, emergency triggers for power delivery, and so on), platform test and debug in an isolated environment, on-board component testing, and cooperative tuning and control. Conventional metal wiring is becoming an inevitable difficulty for the future management of the computing platform. This article presents an ultra-wideband (UWB) wireless interconnect solution. The channel characteristics within a computer chassis are analyzed, including the path loss, multipath reflections, and electromagnetic interferences (EMI). To address the above problems, two prototypes of impulse-radio ultra-wideband (IR-UWB) transceivers are proposed. The first prototype has advantage in power consumption and simplicity and is suitable for low data rate communications; however, it ignores the inevitable frequency offset between transmitter and receiver baseband clocks. In the second prototype, pulse injection-locking is employed for receiver clock recovery and synchronization and an equalizer is introduced in the transmitter to relax the multipath reflections. The second prototype is more suitable for high data rate communications.

Introduction

Platform stability and autonomics requires a collective decision process that optimizes the system states for maximum efficiency in terms of energy usage, performance/watt, thermal tuning, power budget re-balancing, and component profiling for failure analysis. Each platform component plays an optimization game involving multiple components, in which each component is assumed to know the equilibrium strategies of the other components in real time, and no component has anything to gain by changing only its own strategy. An essential ingredient to enable such manageability is an efficient telemetry for observability and control through the broadcast nature of the wireless. For example, to achieve high power/performance efficiency in a multi-socket scenario, a CPU socket cannot change its sleep (or performance) state if it is unaware of its neighboring sockets states. Furthermore, these neighboring states need to be sampled and analyzed in real time to be effective. Although the downscaling of CMOS technologies allows the integration of heterogeneous chips on a single die, the future computing platform still needs many specialized chips from different vendors, such as an RF front end, flash memory card, and LCD driver. In a conventional server blade card, except for many-core microprocessors, high-bandwidth serial link interconnects

"An essential ingredient to enable such manageability is an efficient telemetry for observability and control through the wireless broadcast."

and parallel data buses between chips, a PCB card also includes many low-bandwidth, control-based interconnects that are needed for propagating power, thermal, utilization, reliability, QoS, and performance trends of various system components (CPU, DIMM, fans, network, I/O hub) in the form of sense data. These low-speed interfaces include Joint Test Acting Group (JTAG), Peripheral Equipment Interface (PEI), and Simple Serial Transport (SST) interconnects, and so on[1]. In the future computing system, more processers are required on a single board, integrating numerous side-band channels for various control information. The conventional wired interconnects will not be able to scale to future systems without degrading the quality of control.

"One potential solution is the wireless interconnect system."

One potential solution is the wireless interconnect system. A wireless, control-plane communication system has several advantages[1][2] as, (1) wireless interconnects reduce latency, which is critical in a clock distribution system, platform's manageability, and test or debug capabilities; (2) the broadcast nature of a wireless interconnect can provide link performance characteristics globally, enabling dynamic rerouting for performance optimization, which improves the fault tolerance; (3) wireless interconnects have better scalability and modularity as easily adding or subtracting sideband channels without affecting the system's overall performance; (4) wireless interconnects provide an opportunity to reconfigure particular chips with variable bandwidth on the basis of usage requirement; (5) global control data can be sent in a broadcast as opposed to a serial manner; (6) power optimization and cost reduction. However, advantages of wireless interconnects inside a computing channel are accompanied by several significant challenges, such as channel attenuation and multipath interferences. Recently, ultra-wideband (UWB) has become an attraction for short-range wireless interconnects [3][4] because of its high bandwidth, low power, and immunity to multipath fading, and so on.

In-Chassis UWB Channel Characteristics

While UWB wireless communication systems are widely used in a variety of military, commercial, and consumer applications, the channel models for indoor/outdoor propagations and wireless body area networks have been thoroughly studied[5][6][7]. However, a full-size desktop chassis typically houses substantial amounts of metallic objects and is comprised of a highly reflective metallic case, as shown in Figure 1. The complex environment in a computer chassis indicates a unique channel model. Signals within the enclosed area will experience dense multipath channels with the majority of transmissions within a short distance of less than 20 cm.

"The complex environment in a computer chassis indicates a unique channel model."

The UWB channel measurements were taken within the chassis interior of a standard desktop workstation with dimensions $45 \times 20 \times 40$ cm^3. To accurately characterize the channel during normal operation, all miscellaneous components were left inside the computer during data acquisition. Measurements were taken inside a near-static, electromagnetically-shielded lab. A vector network analyzer was used to capture 1601 data points between

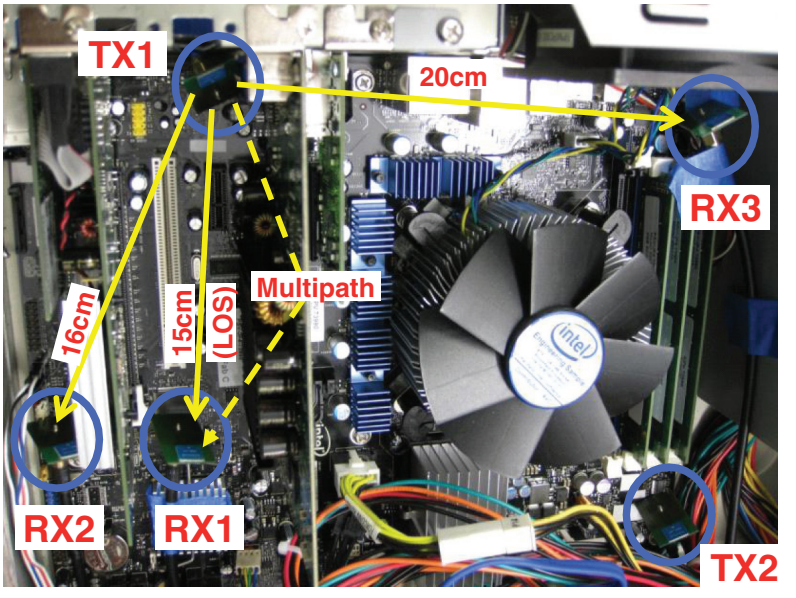


Figure 1: In-chassis communication scenario (Source: Oregon State University, 2010)

3 and 6 GHz, providing a frequency-domain resolution of 1.875 MHz. The EMI measurement was conducted with a spectrum analyzer.

"Path loss of a UWB signal is a function of both the distance and the frequency."

Propagation Path Loss

Because of its large bandwidth, path loss of a UWB signal is a function of both the distance between the transmitter and receiver (d), and the transmitted frequency (f)[7].

$$L(f, d) = L(f) \cdot L(d) \tag{1}$$

Where $\sqrt{L(f)} \propto f^{-k}$, κ is the decay factor. The distance-dependent component $L(d)$ is derived from Friis' equation and is expressed as

$$L_{dB}(d) = \begin{cases} L_{01} + 10n_1 \log_{10}(d / d_{01}) + S_1, & d_{01} < d \le d_{02} \\ L_{02} + 10n_2 \log_{10}(d / d_{02}) + S_2, & d > d_{02} \end{cases} \tag{2}$$

where L_{01} and L_{02} are the loss at the reference distance d_{01} and d_{02}, respectively. When $d_{01} < d \le d_{02}$, propagation is a near-field scenario; when $d > d_{02}$, propagation is far-field. The near-field and far-field path loss exponents are n_1 and n_2. S_1 and S_2 are the near-field and far-field shadowing components with corresponding standard deviations σ_{S1} and σ_{S2}. Figure 2(a) shows the channel path loss in both near-field and far-field propagation[8]. All parameters related to this model are given in Table 1. For all of these parameters, spatial averaging was implemented[7] to remove the frequency dependence of the path loss, except for the determination of κ. Averaging was performed by repeating the path loss experiment for different transmitting antenna positions and averaging the acquire data at each spatial position.

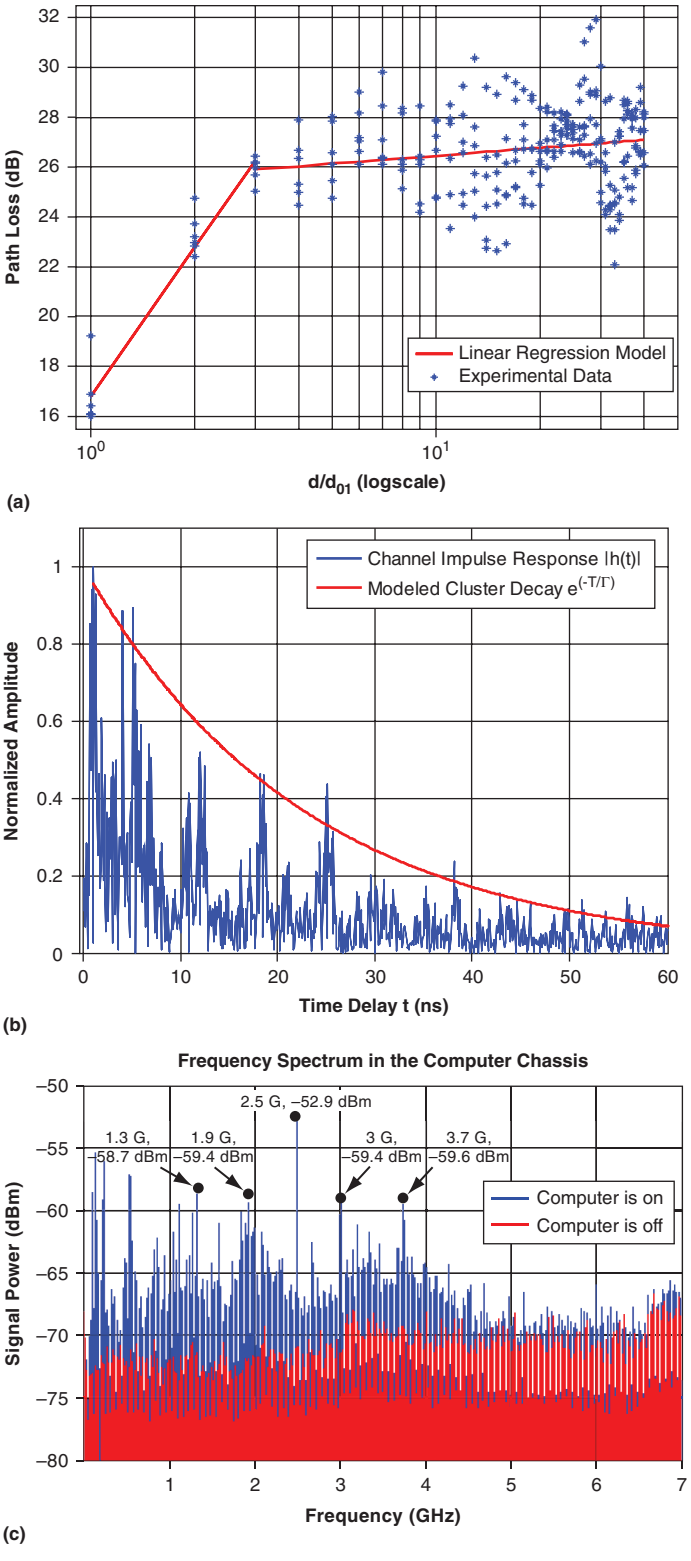


Figure 2: UWB channel model within a computer chassis (a) path loss measurement (b) multipath measurement (c) electromagnetic interferences (Source: Oregon State University, 2010)

Parameter	Near-field	Far-field
Valid Range (cm)	1 to d_{02}	d_{02} to 40
i (for parameters below)	1	2
d_{0i} (cm)	1	3
G_{0i}	16.78	–25.90
n_i	1.99	0.11
σ_{Si}	0.94	1.67
κ (independent of i)	1.31	1.47

Table 1: Channel Propagation Path Loss Model Parameters (Source: Oregon State University, 2010)

Channel Impulse Response

According to the Saleh-Valenzuela model[9] for indoor propagation, the complex baseband impulse response for a general indoor multipath propagation can be expressed as

$$h(t) = \sum_{k,l} \beta_{k,l} \cdot e^{j\theta_{k,l}} \cdot \delta\left(t - T_l - \tau_{k,l}\right) \tag{3}$$

where l is the cluster index, k denotes the kth ray of the lth cluster, $\beta_{k,l}$ denotes the path loss of the kth path of the lth cluster, $\tau_{k,l}$ is the delay of the kth path of the lth cluster relative to the lth cluster arrival T_l, and $\theta_{k,l}$ is the component phase, which is uniformly distributed over [0, 2π]. Through our experimental measurements, we have found that this expression adequately describes multipath propagation within this enclosed environment.

"This expression adequately describes multipath propagation within this enclosed environment."

In a common indoor channel, the number of clusters L is generally modeled as a Poisson-distributed random variable with mean $\bar{L}$. The probability density function (pdf) of L is

$$f_L(l) = \frac{\bar{L}^{\,l} \cdot e^{-\bar{L}}}{l!} \tag{4}$$

Clusters are typically identified by visual inspection[7] while the mean $\bar{L}$ is derived by averaging the observed clusters per impulse response.

Table 2 summarizes the channel impulse response model parameters, which are derived from best-fit algorithms on the measurement data[7]. The cluster arrival rate is Λ, Γ is the inter-cluster decay time constant, γ is the intra-cluster decay time, and $\sigma_{cluster}$ is the cluster shadowing standard deviation. Additionally, we have observed that the distribution of small-scale component weight $\beta_{k,l}$ is Nakagami with a log-normally distributed m-factor. The parameter μ_m and σ_m are the mean and derivation of the m-factor, respectively. The RMS delay spread τ_{rms}, is used to interpret delay dispersion. Since the maximum transmission distance for our channel environment is so short, we found that a mean value $E\{\tau_{rms}\}$ is sufficient to characterize this factor.

The power delay profiles show the received signal power as a function of time delay, giving an intuitive inspection of the multipath channel.

"The power delay profiles give an intuitive inspection of the multipath channel."

$$PDP = 20 \log |h(t)| = 20 \log \left| \sum_{k=0}^{N-1} \Delta f \cdot H(k \cdot \Delta f) \cdot e^{j\frac{2\pi kn}{N}} \right| \quad (5)$$

where $h(t)$ is the UWB channel impulse response (CIR) in the time domain, and $H(f)$ is the measured UWB channel frequency response. Figure 2(b) illustrates the high density of multipath clusters encountered in this environment because of the inherent reflectivity of the metallic surrounding[8].

Parameter	Near-field	Far-field
Valid Range (cm)	1 to d_{02}	d_{02} to 40
i (for parameters below)	1	2
d_{0i} (cm)	1	3
$\bar{L}$	18.27	29.71
Λ (1/ns)	0.377	0.376
Γ (ns)	17.13	23.03
γ (ns)	1.12	1.03
$\sigma_{cluster}$ (dB)	5.55	3.87
μ_m (dB)	1.57	1.76
σ_m (dB)	1.04	0.99
$E\{\tau_{rms}\}$ (ns)	25.65	23.62

Table 2: Channel Impulse Response Model Parameters (Source: Oregon State University, 2010)

Electromagnetic Interferences

Electromagnetic interferences (EMI) from switching noise of the ICs in the chassis can limit the wireless link budget and therefore the maximum data rate that can be achieved[1]. For example, the many parallel, single-ended I/Os necessary for the multiple DIMMs might create significant EMI, making it difficult to send wireless data reliably. To understand the magnitude of possible EMI, we designed a custom test suite for the server blade that fully stresses the I/O interfaces, creating the potential for a significant amount of background crosstalk.

"EMI can limit the wireless link budget and therefore the maximum data rate that can be achieved."

We applied a spectrum analyzer to measure the EMI from 0.1 to 7 GHz within this chassis. We found dominant spectral components at 1.3 GHz, 1.9 GHz, 3 GHz, and 3.7 GHz, with the largest signal spur occurring at 2.5 GHz (–53 dBm), as shown in Figure 2(c). Given that the UWB transceiver will presumably eventually operate within the 3.5–5 GHz band with a peak operating frequency of around 4 GHz, the EMI had a nominal effect on the bit error rate (BER).

IR-UWB Transceiver Design

The analysis of the UWB channel in a computer chassis showed that although the constraints of channel attenuation, multipath inter-symbol interference, and EMI noise are challenging, they were tolerable and acceptable concerning the SNR degradation. The next challenge is developing practical implementations of the wireless transceivers in CMOS technology to enable high bandwidth, low power consumption, robustness regarding multipath and EMI, and low cost.

Previous Studies on IR-UWB Transceivers

"An IR-UWB transmitter is greatly simplified when compared with a conventional radio."

Within a traditional narrowband RF transmitter, a mixer is employed to convert the baseband signal to the RF carrier frequency, requiring a PLL in order to generate the LO carrier. However, in an IR-UWB transmitter, frequency conversion is usually performed by differentiation of a Gaussian pulse output, where the higher the differentiation order, the higher the center frequency. Therefore, an IR-UWB transmitter is greatly simplified when compared with a conventional radio. Because of the ultra-wideband nature of IR-UWB and its spectral overlap with other sensitive frequency bands, the transmitted power spectral density of IR-UWB must be designed not to exceed –41.25 dBm/MHz. This low transmitted power means that conventional power-consuming power amplifiers are not a requirement for these UWB systems, which further improves the power efficiency of IR-UWB systems.

Several types of modulation can be used for pulse-based UWB systems, including Binary Phase Shift Keying (BPSK), On Off Keying (OOK), and Pulse Position Modulation (PPM). BPSK modulation generates 180° phase-shifted pulses while transmitting baseband symbols "1" and "0". OOK is performed by generating transmitted pulses only while transmitting "1" symbols, while PPM is performed by generating pulses at different phase delays. Therefore, BPSK has an advantage over other modulation types due to an inherent 3 dB increase in separation between constellation points[10].

The main block in an IR-UWB transmitter–pulse generator (PG) can be categorized into analog pulse generators and digital pulse generators. An analog PG[11] employs the square and exponential functions of MOS transistors biased in saturation and weak inversion, respectively. However, it suffers from low output amplitude. A digital PG combines the edges of a rectangular signal and its inverted signal to form a very short duration pulse, and then a differential circuit or a multiphase combination circuit[12] is employed to up-convert the pulse without using a local oscillator[10][13]. The main problem with the digital PG is the difficulty in controlling the exact pulse shape and its spectrum due to PVT variations.

"Conventional IR-UWB receivers can be categorized into coherent receivers, noncoherent receivers and direct down-sampling receivers."

Conventional IR-UWB receivers can be categorized into coherent receivers[11][14], noncoherent receivers[15][18], and direct down-sampling receivers[19]. The direct down-sampling receiver is quite straightforward; the received pulse is amplified and then sampled by a multi-gigahertz sampling rate ADC. Although at first glance this architecture seems simple, it is seldom used in the 3–10.6 GHz frequency band for several reasons. First, it is difficult to implement a high gain,

wide-bandwidth RF amplifier (at least 60 dB is required for 10 m transmission range), as it may easily oscillate and also consumes significant power. Second, the design of a multi-gigahertz ADC is not trivial. Although 1-bit resolution may be sufficient[20], this ADC consumes significant power in the clock distribution of the high data rate communications.

Both coherent and noncoherent receivers correlate the received pulse first, such that the center frequency is down-converted to baseband. The difference is that in a coherent receiver, the received pulse correlates with a local template pulse; in a noncoherent receiver, the received pulse correlates with itself. Therefore, a noncoherent technique exhibits the disadvantage that the noise, as well as signal is both amplified at the receiver[16]. Simulation results show that a noncoherent receiver requires at least 6 dB higher SNR than a coherent receiver for a fixed BER[21]. However, the advantage of a noncoherent receiver is that it avoids the generation of a local pulse as well as the synchronization between the local and received pulses. In a coherent receiver, in order to obtain large enough down-converted signal for quantization, the local and received pulses must be synchronized to less than 100 ps[11][22] in 3–5 GHz frequency band, which would be even tougher in 6–10 GHz frequency band. This precise timing synchronization can be achieved with a DLL or PLL, which is very power consuming[11][22]. However, in a noncoherent receiver, only symbol level synchronization between the baseband clock and received data is needed with a resolution of nanoseconds.

"A noncoherent receiver requires at least 6 dB higher SNR than a coherent receiver for a fixed BER."

Two Prototypes of IR-UWB Transceivers

Two IR-UWB transceiver systems will be introduced for different applications. The proposed noncoherent IR-UWB transceiver is low power and simple and is suitable for low date rate communications. The proposed injection-locking IR-UWB transceiver realizes synchronization with injection-locking and is suitable for high data rate communications.

"The proposed noncoherent IR-UWB transceiver is low power and simple and is suitable for low date rate communications."

Proposed Noncoherent IR-UWB Transceiver with Baseband Synchronization

The IR-UWB transmitter is based on a former BPSK-modulated transmitter implementation[23]. Since a noncoherent receiver detects only the energy of the received pulses rather than the phase of the pulses, BPSK modulation is not suitable for the noncoherent receiver. Hence, OOK modulation is chosen in this system. The transmitter implementation includes mode selection and power control blocks, in addition to the pulse generator and output buffer, as shown in Figure 3. The power control block is used to turn off the output buffer during pulses intervals in order to reduce the power consumption. When BPSK is selected, the power control block turns the output buffer on before the rising edge of the clock signal—the FreqCtrl signal is enabled and lasts for about 2 nanoseconds, regardless of whether BBin is "1" or "0"; otherwise, when OOK modulation is selected, the output buffer is enabled only when BBin is "1". Therefore, the introduction of the power control block means that the transmitted power consumption is proportional to the data rate. The output spectrum of IR-UWB transmitter is difficult to control due to PVT deviations and inaccurate parasitic models for differential

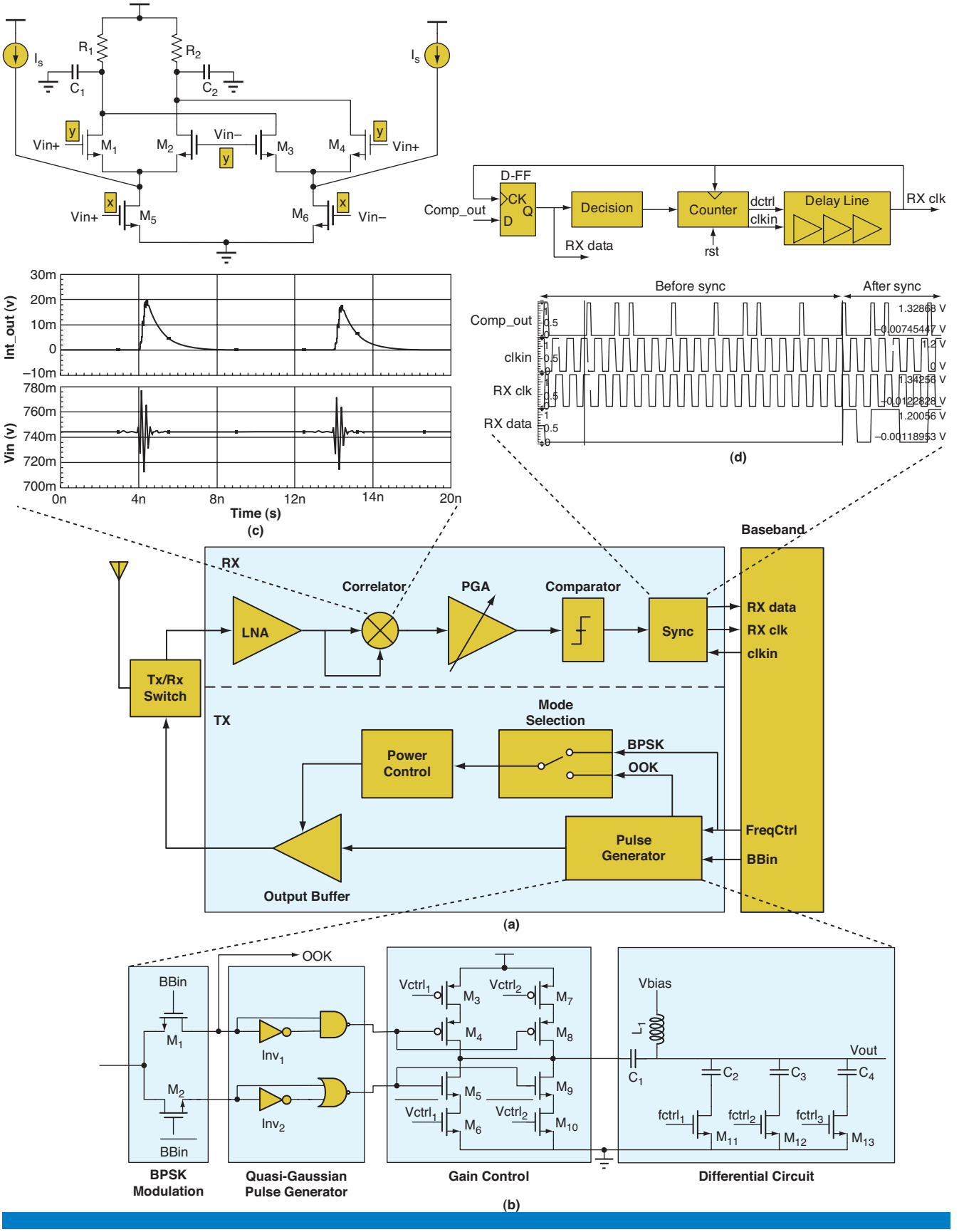


Figure 3: Proposed noncoherent IR-UWB transceiver (a) system architecture (b) pulse generator (c) correlator (d) baseband clock synchronization

(Source: Fudan University, 2010)

circuit. Therefore, this chip implements four-step spectrum control by using signals $fctrl_{1-3}$ showing a measured frequency tuning range of 3.2–4.1 GHz. Furthermore, three-step gain control by signals $Vctrl_{1-2}$ is implemented to enable adaptable output power spectral density in order to meet the FCC spectral mask at a different data rate.

The proposed IR-UWB receiver employs the noncoherent receiver architecture as shown in Figure 3. After first being amplified by the low noise amplifier (LNA), the received pulse is then self-correlated by a correlator, amplified by a programmable gain amplifier (PGA), and then sent to a comparator for digital quantization. Finally the received data is synchronized with the baseband clock.

A correlator is the critical block in the receiver. A conventional correlator consists of a multiplier and an integrator. Previous correlators used in both coherent receivers[11][24] and noncoherent receivers[18] needed to synchronize the received pulse with local controlling signals first. This synchronization process is analogous to the RF front-end synchronization in a coherent receiver requiring a strict timing resolution. In this design, the duty-cycled characteristic of the IR-UWB system is used to remove the timing synchronization. The upper left area of Figure 3 presents the proposed multiplier and integrator-merged correlator. The multiplier employs a Gilbert topology, while the integrator is realized by capacitors C1 and C2. As shown in this figure, after the pulse is multiplied with itself, the integrator begins to integrate, and between the pulses intervals, the integrator starts to discharge and prepare for the next integration.

"The duty-cycled characteristic of the IR-UWB system is used to remove the timing synchronization."

After the received signal is squared and integrated by the correlator, a comparator compares it with a reference voltage and performs digital quantization. However the comparator output is a return-to-zero (RZ) signal, which needs to be converted to a non-return-to-zero (NRZ) signal that can synchronize with the baseband clock. In a coherent receiver, a DLL/PLL is usually introduced to perform synchronization between the received pulse and the local pulse, needing precision on the order of several tens of picoseconds. However, in a noncoherent receiver, the RZ signal quantized by the comparator exhibits a duty cycle on the order of nanoseconds. Therefore, a low jitter DLL/PLL is no longer necessary and a sliding correlator is employed. The digital synchronization circuit is shown in the upper right of Figure 3, where clkin, comp_out, RX clk, and RX data are the baseband clock, the comparator output, the recovered baseband clock, and the recovered data, respectively. With a reset signal, the delay line control signal dctrl is set to 0, such that there is no delay between the RX clk and clkin. Then the Sync block starts operation, and RX clk samples comp_out. If the RX clk is not synchronized with comp_out, the decision block enables the counter that increases the value of dctrl—thus elongating the latency of the delay line until RX clk and comp_out are synchronized.

"A low jitter DLL/PLL is no longer necessary and a sliding correlator is employed."

The proposed IR-UWB noncoherent transceiver is implemented in a 0.13 μm 1P8M CMOS technology. The die area is 2 mm × 2 mm, as shown in Figure 4(a). With a supply voltage of 1.2 V, the power consumption of the transmitter is only 1.2 mW and 2.2 mW when transmitting 50 Mb/s and 100 Mb/s baseband signals, respectively; the power consumption of the receiver is 13.2 mW.

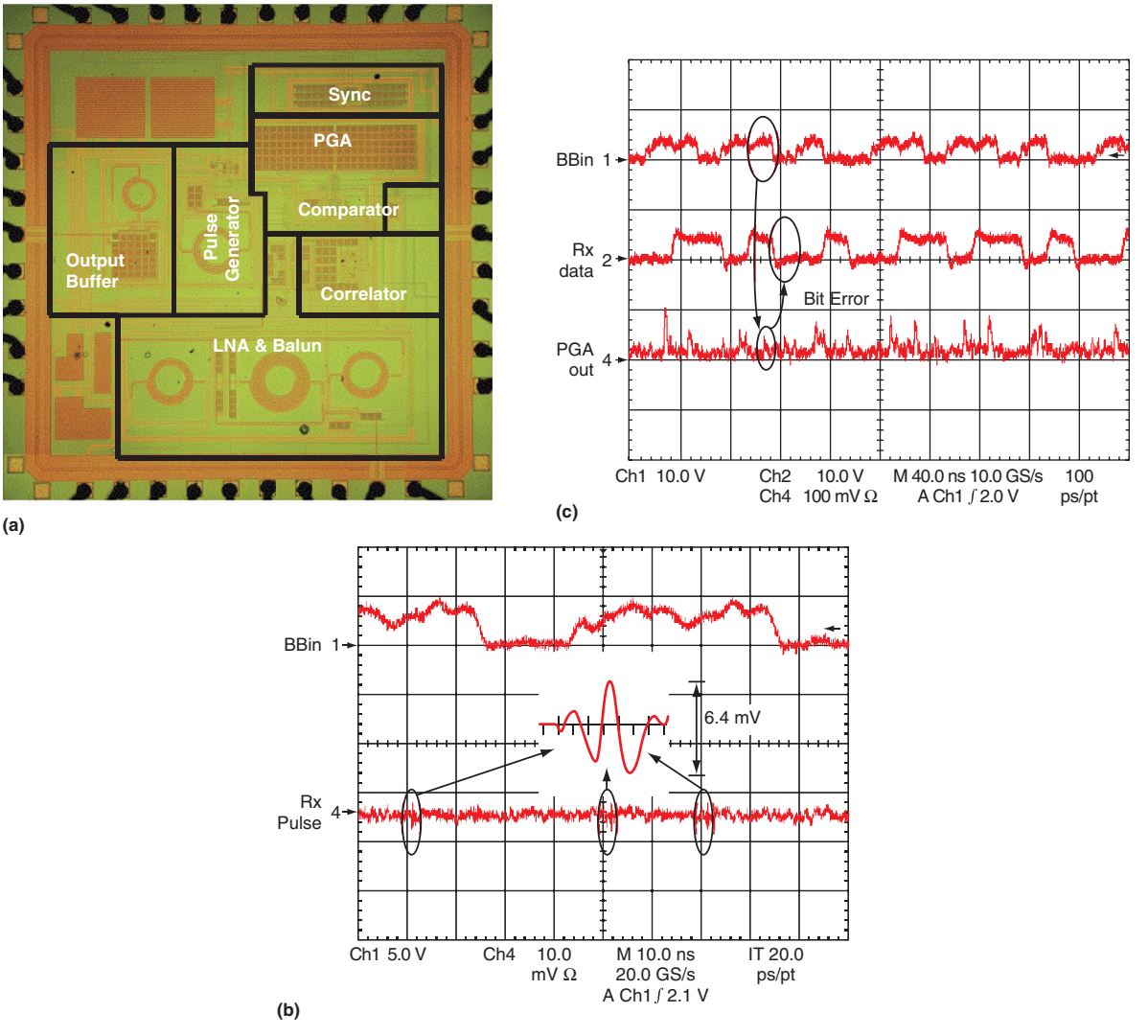


Figure 4: Measurement results of the proposed noncoherent IR-UWB transceiver (a) chip microphotograph (b) received pulses with transmission distance of 10 cm (c) BER performance of the receiver with transmission distance of 10 cm
(Source: Fudan University, 2010)

The amplitude and spectrum tunable transmitter has output pulses with peak-to-peak voltage of 240 mV, 170 mV, and 115 mV and the frequency center of the spectrum has a tuning range of 3.2–4.1 GHz.

The receiver provides a total gain ranging 43–70 dB, in which the LNA exhibits a gain variation of 7.5 dB in high/low gain mode; the PGA incorporates an 8-step, 3-dB gain control with an RMS error of 0.7 dB. The receiver shows a minimum noise figure of 8.6/13.3 dB while operating in high/low gain mode, with a noise figure variation less than 2 dB in the 3–5 GHz frequency band. The 1-dB compression point of the receiver is –28/–22 dBm in high/low gain mode.

BER performance of the receiver is measured by transmitting 50 Mb/s random data from FPGA. The employed antennas are 3–5 GHz monopole omnidirectional antennas, manufactured by Fractus. With transmitted amplitude of 115 mV, the received pulses are attenuated to only 20.4 mV (–50 dBm) and 6.4 mV (–61 dBm) when the distance between the antennas is 1 cm and 10 cm, respectively. The receiver achieves a BER of 10^{-3} when the distance between the antennas is set to 1 cm (–50 dBm). While the distance extends to 10 cm (–61 dBm), the BER performance is greatly deteriorated to over 10^{-2}. As shown in Figure 4(c), the TX pulse is OOK modulated, every pulse represents bit 1 at baseband. The received pulses are correlated and then amplified by the PGA, where PGA out is the buffered output of the PGA. A bit error occurred in the synchronized RX data as the received pulses are distorted by the antennas and the transmission channel.

"A bit error occurred as the received pulses are distorted by the antennas and the transmission channel."

Table 3 lists the performance summary of the proposed noncoherent IR-UWB transceiver.

Parameter	Measurement Results
Technology	0.13µm CMOS
Die Size	2 mm × 2 mm
Modulation	OOK
Data Rate	50–100 Mbps
VCO Frequency Range	3–5 GHz
Transmitted Pulse Width	1ns
Rx NF	8.6 dB
Rx Gain	70 dB
Rx IP1dB	–28 dBm
Rx Sensitivity	–50 dBm at 50Mbps, BER < 10^{-3} –61 dBm at 50 Mbps, BER > 10^{-2}
Energy Efficiency	Tx: 22 pJ/b; Rx: 0.13 nJ/b at 100 Mbps

Table 3: Performance Summary of the Proposed Noncoherent IR-UWB Transceiver
(Source: Fudan University, 2010)

Proposed IR-UWB Transceiver with Injection-Locking Synchronization

The proposed noncoherent receiver above greatly relaxed the difficulty in synchronization; however, the inevitable frequency offset between the baseband clocks of the transmitter and receiver still exists, which needs to be compensated in the digital baseband circuit. In the proposed receiver in Figure 5, the receiver clock is extracted from the received impulses using pulse injection-locking. The injection-locking-VCO (ILVCO) employs 4-bit capacitance bank to tune the VCO free-running frequency. The closer the ILVCO free-running frequency is to the input pulse frequency, the smaller the jitter of the recovered clock. Hence, the receiver clock is automatically phase aligned with the received pulse, exhibiting neither clock offset nor phase drift.

"The receiver clock is extracted from the received impulses using pulse injection-locking."

Additionally, the initial phase difference between the received impulse and the receiver clock can be statically adjusted at startup by using a programmable phase shifter in the receiver clocking path, aligning the receiver sampling point with the optimal SNR position of the incoming impulses. The sampling clock of the 5-level flash ADC is generated by dividing the VCO output, which is the same as the impulse data rate. In the transmitter, a 3–5 GHz LC-VCO output is clock-gated by a baseband pulse that generates the transmitted pulses. The baseband, pulse-shaping control block ("pulse window") enables tunable pulse widths between 0.4–10 ns.

"This proposed receiver clock recovery uses pulse injection-locking from the transmitted pulses, similar to subharmonic injection-locking."

This proposed receiver clock recovery uses pulse injection-locking from the transmitted pulses, similar to subharmonic injection-locking in Lee et al.[25][26] As shown in the lower left of Figure 5, Region I denotes the region where the offset frequency is smaller than the locking range of the injection locked VCO, where the VCO noise is suppressed by the injected signal. Region II is the competition region, where the VCO phase noise is the result of the competition between the injected signal and the VCO free-running signal. In Region III, beyond the injected signal frequency, the VCO phase noise is dominated by the VCO free-running phase noise. Similar to a subharmonic injection-locked PLL[26], for this pulse injection-locked VCO, the effective division ratio N can be expressed as:

$$N = \frac{f_{out}}{\alpha \cdot \beta \cdot DR_{inj} \cdot n} = \frac{f_{out}}{\alpha \cdot \beta \cdot DR_{inj} \cdot (W_{pulse} / T_{out})} = \frac{1}{\alpha \cdot \beta \cdot DR_{inj} \cdot W_{pulse}} \quad (6)$$

where α is the probability that data is "1", β is the roll-off coefficient due to pulse-shaping at the transmitter output compared with an uniformly-gated, sine-wave pulse; DR_{inj} is the data rate; f_{out} and T_{out} are the ILVCO output signal frequency and period; and W_{pulse} is the pulse width. Similar to Lee et al.[25], the phase noise degrades as 20 log N dB, compared with the injected signal. From Equation 6, we can see that an increase in the injection pulse rate or pulse width reduces the phase noise of ILVCO output, because more external clean energy is injected into the noisy oscillator.

"An increase in the injection pulse rate or pulse width reduces the phase noise of ILVCO output."

An injection-locked VCO suppresses the noise within the locking range, similar to a first-order PLL, where the bandwidth ω_{BW} is equal to the locking range ω_L. Similar to the subharmonic injection-locked PLL, the locking range ω_L degrades as N increases. The locking range of a sine wave injected VCO is described in Razavi[27] and Adler[28]:

$$\omega_L = \frac{\omega_{out}}{2Q} \cdot \frac{I_{inj}}{I_{osc}} \cdot \frac{1}{\sqrt{1 - \frac{I_{inj}^2}{I_{osc}^2}}} \quad (7)$$

where Q represents the quality factor of the tank, and I_{inj} and I_{osc} represent the injected and oscillation currents of the LC-tank VCO. With pulse injection-locked VCOs, the effective injection current is $I_{inj,eff} = I_{inj}/N$, because less

current is injected when compared with full sine-wave injection. Consequently, the locking range of a pulse injection locked VCO is modified as:

$$\omega_L = \frac{\omega_{out}}{2Q} \cdot \frac{I_{inj}}{I_{osc}} \cdot \frac{1}{N} \cdot \frac{1}{\sqrt{1 - \frac{I_{inj}^2}{I_{osc}^2 \cdot N^2}}} \approx \frac{\omega_{out}}{2Q} \cdot \frac{I_{inj}}{I_{osc}} \cdot \frac{1}{N} \quad (8)$$

Channel multipath is a major constraint in a computer chassis due to short distances and the metallic case. Fortunately, in a chassis, the wireless transceiver locations are stationary, so that the exact distances and time delay of the multipath is a priori predictable and time invariant. A multiple transmitter equalizer is designed that can reduce the two most severe multipath reflections. As shown in Figure 5, Tap1 and Tap2 are sign- and coefficient-programmable delayed versions of the main signal, with delay time of τ_1 and τ_2, respectively.

"Channel multipath is a major constraint in a computer chassis due to short distances and the metallic case."

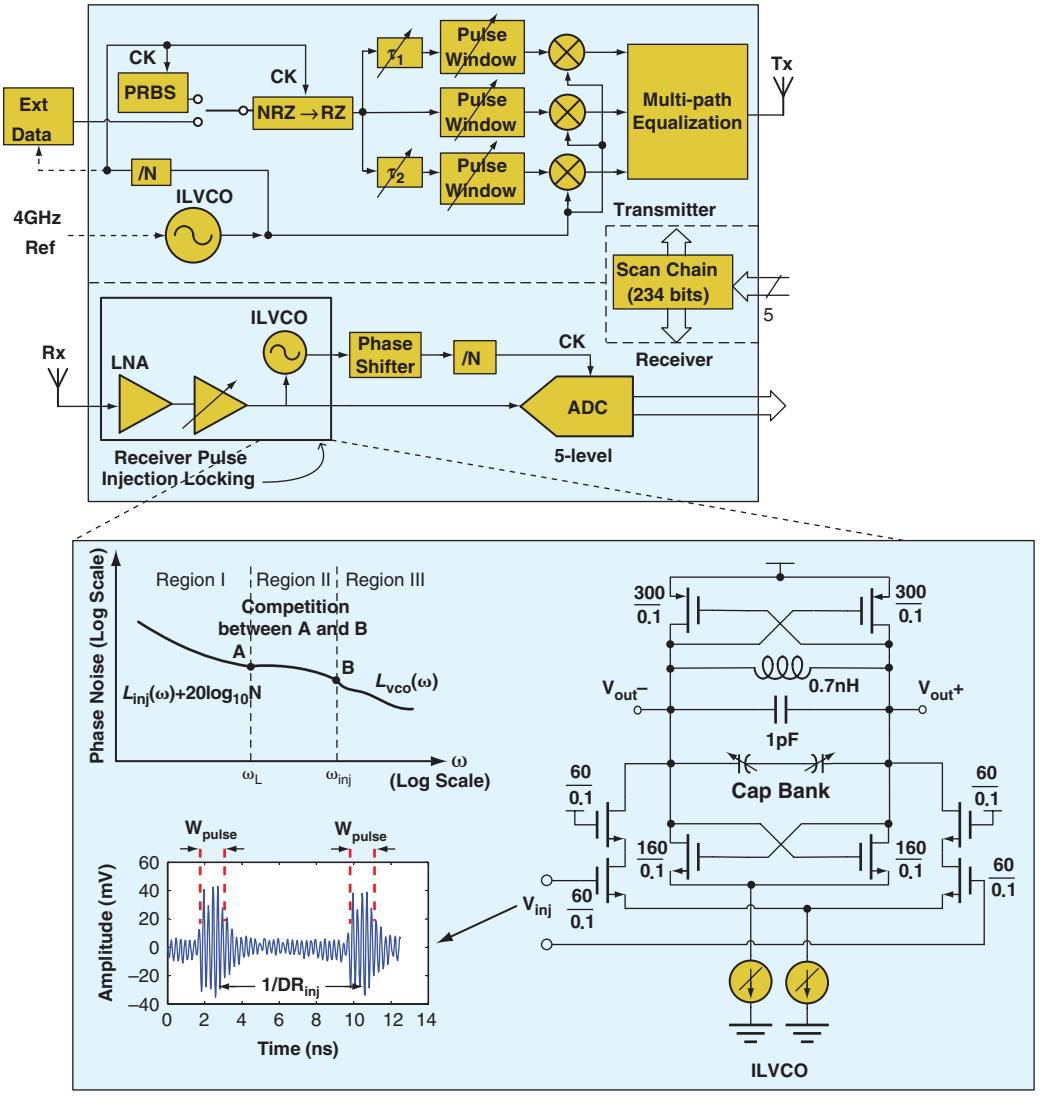


Figure 5: Proposed IR-UWB transceiver with injection-locking synchronization (Source: Oregon State University, 2010)

The 2 mm^2 IR-UWB transceiver[29] is built in a 90 nm CMOS 1.2 V mixed-signal technology, as shown in Figure 6(a). The on-chip scan-chain is controlled by a computer via a Ni-DAQ interface. Indoor free space measurements are performed with transmission distance of 10–20 cm.

The measured amplitude of the pulse is 160 mVpp, with a nominal pulse width of 1 ns and data rate of 62.5 Mbps. The frequency spectrum fulfills the FCC UWB spectral mask except for the GPS band, which can be improved by incorporating more design attention to spectral shaping in the transmitter output. The maximum transmission data rate is 500 Mbps.

"The maximum transmission data rate is 500 Mbps."

The measured S11 is below −10dB in 3–5 GHz frequency band. After the recovered IL-VCO clock locked to the LNA output, with a 1-ns pulse width and a data rate of 250 Mb/s, the recovered clock jitter is 7.6 ps-RMS. For the same pulse width, the data rate of 125 Mbps and 500 Mbps are also measured, with RMS jitter of 8.0 ps and 23 ps. Due to limited bandwidth of LNA, the inter-symbol interference (ISI) seems worse at the high data rate of 500 Mbps, increasing the clock jitter.

Figure 6(b) shows the measured injection-locking range versus pulse width and pulse repetition rate. As can be seen, a wider pulse width and higher data rate improve the locking range, as more transmitted pulse energy synchronizes the receiver IL-VCO. Figure 6(b) also shows the measured close-in phase noise, from free-running without injection, to pulse injection rates (DR_{inj}) of 125 Mbps, 500 Mbps and sine-wave injection. Lower phase noise is exhibited at higher injection rates, as the phase updates occur at a higher frequency, similar to the dynamics in a first-order PLL. The result also verifies Equation 6, showing an approximately 12-dB phase noise difference between 125 Mbps and 500 Mbps pulse injection rates. Without pulse injection, the free-running VCO shows very large phase noise at a low frequency offset.

While a long string of empty data transitions would result in loss of phase synchronization, conventional DC-balanced codes such as 8b/10b can limit the maximum run length. Transmission using the on-chip PRBS-15 modulator, exhibiting a maximum string length of 14 zeros, showed no loss in receiver phase synchronization.

The indoor free-space measurement setup uses two UWB antennas that are placed 10 cm apart. Figure 6(c) show the transmitted digital data, received pulses after LNA gain, the recovered Rx clock, and finally the received demodulated data at 500 Mbps.

"Interferers will increase the recovered clock jitter and increase the BER."

Because this receiver is injection locked, interferers will increase the recovered clock jitter and increase the BER, so it is important to measure interference performance. By putting a single tone interferer through a UWB antenna close to the receiver antenna, characterizing the received interference power at receiver input, and increasing the interfere power till the BER reaches 10^{-3},

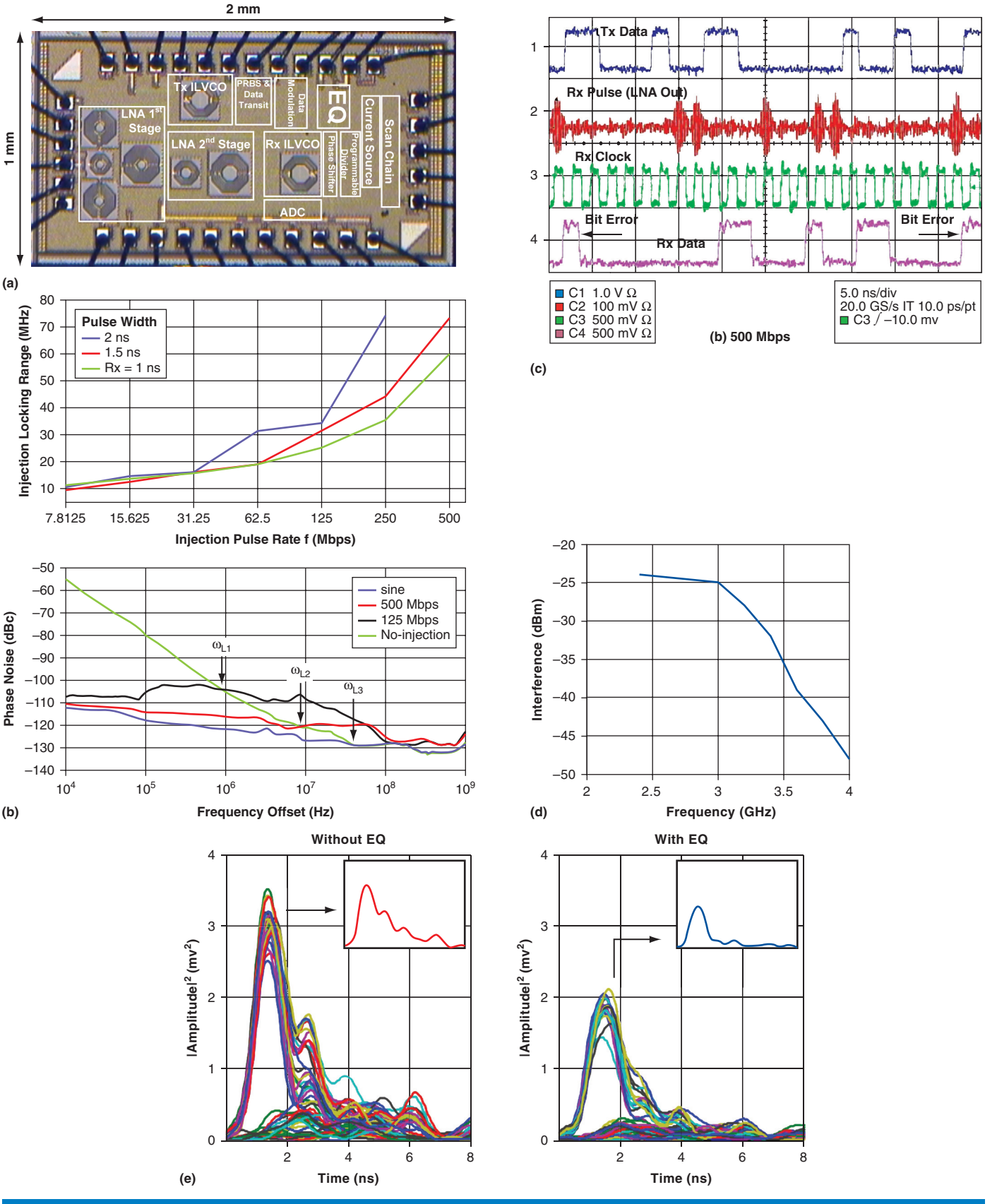


Figure 6: Measurement results of the proposed IR-UWB transceiver with injection-locking synchronization (a) chip microphotograph (b) pulse injection lock range and pulse injection locked VCO phase noise (c) data transmission and receive (d) measured maximum tolerable interference power (e) measured received signal with and without multipath equalization inside computer chassis (Source: Oregon State University, 2010)

then we get the maximum tolerable power at receiver input. With a communication distance of 14 cm, 125 Mpbs 110 mVpp 1-ns wide pulses are transmitted for interferer test. The measured interference performance is shown in Figure 6(d) for both in band and out of band. The maximum tolerable interferer power is −50 dBm at 4 GHz, and −25 dBm at 2.4 GHz. And considering the EMI in Figure 2(c), this receiver can work robustly in a computer chassis.

"Multipath reflections affect the signal differently in short-distance channels and long-distance channels."

The computer chassis channel exhibits intensive multipath reflections. Multipath reflections affect the signal differently in short-distance channels and long-distance channels (relative to the data rate): 1) for short channels, multipath reflections are close to the main signal (direct path), causing intra-symbol interference (while OOK modulation is somewhat enhanced by this additive energy from multipath reflections, BPSK modulation would be severely limited due the sign change inversion); 2) For long channels, multipath reflections show longer delay from the main signal, and may fall in the next symbol. Both intraference and interference can degrade the BER.

"The multipath equalizer can cancel multipath reflections in both short-distance and long-distance channels."

The multipath equalizer can cancel multipath reflections in both short-distance and long distance channels for this OOK IR-UWB transceiver. In a computer chassis, all antennas are stationary, resulting in a fixed amplitude and time delay for the multipath signal that arrive at each receiver. Hence, the two-tap coefficient delay, amplitude, and sign of the equalizer were calibrated at reset time, and adjusted differently for each of the multipath propagations.

For simplicity, the received pulses and recovered clock are captured by a high sampling rate oscilloscope, and then the data are processed in MATLAB. In this case, the quantization noise of the flash ADC in the receiver is eliminated. Figure 6(e) shows the pulse response (after squaring and low-pass filtering) before and after equalization is applied, for one of the receivers on the motherboard. On the left, a single pulse response is observed with several multipath pulse interferers causing a long pulse tail. On the right, a single pulse is observed where the first tap equalization is activated, significantly reducing the multipath reflections. At a data rate of 250 Mbps, the recovered ADC clock jitter was improved significantly after applying the equalizer, reducing RMS clock jitter by 27.4 percent at RX1 in Figure 1, while the motherboard was operational. Within an enclosed chassis that exhibits significant multipath interference, at 250 Mbps BER is improved from 0.0158 to 0.0067 without/with first-tap equalization enabled respectively. While the proposed equalization can help cancel the multipath reflections, it is difficult in practice to eliminate them entirely.

Table 4 lists the performance summary of the proposed injection-locking IR-UWB transceiver.

Parameter		Measurement Results	
Technology		90 nm CMOS	
Die Size		1 mm × 2 mm	
Modulation		OOK	
Data Rate		7.8125–500 Mbps	
VCO Frequency Range		3.7–4.5 GHz	
Transmitted Pulse Width		0.5–10 ns	
Rx Sensitivity(free space)		–64 dBm at 125 Mbps, BER < 10–3	
Rx Sensitivity(free space)		–60 dBm at 500 Mbps, BER < 10–1	
Energy Efficiency (with ADC)		Tx: 90 pJ/b; Rx: 90 pJ/b at 500 Mbps	
In-chassis BER (TX1–RX1) at 125 Mbps	w/oEMI	w/oEQ	1.7 × 10–3
		wEQ	3.3 × 10–4
	wEMI	w/oEQ	2 × 10–3
		wEQ	3.3 × 10–4

Table 4: Performance Summary of the Proposed Injection-Locking IR-UWB Transceiver
(Source: Oregon State University, 2010)

Summary

Use of low power wireless interconnects provides bidirectional telemetry and eases the manageability by reducing the routing complexities relating to numerous component interconnects and replacing them with a single transceiver per component. The broadcast nature of wireless makes available the equilibrium strategy to all other components in real-time without adding any routing complexity so that they can optimize themselves based on the collective strategy. Wireless chip-to-chip interconnects can provide benefits that are unattainable by other interconnect technologies and therefore should be considered for next-generation, many-socket computing platforms. In this article, the UWB channel model within a computer chassis is analyzed. The two proposed IR-UWB transceivers are both applicable. The noncoherent IR-UWB transceiver with baseband synchronization is low power and low complexity in the RF front end, but needs further frequency offset compensation in digital signal processing. The injection-locking IR-UWB receiver realizes both phase and frequency synchronization in the RF front end and a two-tap equalizer is employed in the transmitter to cancel the multipath signals.

References

[1] P. Y. Chiang et al., "Short-range, wireless interconnect within a computing chassis: design challenges," *IEEE Design & Test of Computers,* vol. 27, no. 4, pp. 32–43, July/Aug. 2010.

[2] B. A. Floyd, C. Hung, K. K. O, "Intra-chip wireless interconnect for clock distribution implemented with integrated antennas, receivers and

transmitters," *IEEE J. Solid-State Circuits*, vol. 37, no. 5, pp. 543–552, May 2002.

[3] Y. Zheng, Y. Zhang, Y. Tong, "A novel wireless interconnect technology using impulse radio for interchip communications," *IEEE Transactions on Microwave Theory and Techniques*, Vol. 54, No. 4, pp. 1912–1920, Apr. 2006.

[4] J. Gelabert, D. Edwards, C. J. Stevens, "UWB wireless interconnect scheme for communication devices within small conducting enclosure," *Proceeding of the 5th European Conference on Antennas and Propagation*, pp. 1743–1747, Apr. 2011.

[5] Z. Irahhauten, H. Nikookar, G. J. M. Janssen, "An overview of ultra wide band indoor channel measurements and modeling," *IEEE Microwave and Wireless Components Letters*, Vol. 14, No. 8, pp. 386–388, Aug. 2004.

[6] L. Xia, S. Redfield, P. Chiang, "Experimental characterization of a UWB channel for body area networks," *EURASIP Journal on Wireless Communications and Networking*, Article ID: 703239, 2011.

[7] A. F. Molisch et al., "IEEE 802.15.4a channel model-final report," *Tech. Rep. Doc. IEEE 802.*15-04-0662-02-004a, 2005.

[8] S. Redfield et al., "Understanding the ultrawideband channel characteristics within a computer chassis," *IEEE Antennas and Wireless Propaga. Lett.*, vol. 10, pp. 191–194, 2011.

[9] A. Saleh and R.A. Valenzuela, "A statistical model for indoor multipath propagation," *IEEE J. Select. Areas Commun.*, Vol. 5, no. 2, pp. 128–137, Feb. 1987.

[10] D. D. Wentzloff, A. P. Chandrakasan, "Gaussian pulse generators for subbanded ultra-wideband transmitters," *IEEE Trans. Microw. Theory Tech.*, Vol. 54, No. 4, Apr. 2006. pp. 1647–1655.

[11] Y. Zheng, et al., "A CMOS carrier-less UWB transceiver for WPAN," *in IEEE ISSCC Dig. Tech. Papers*, pp. 116–117, 2006.

[12] H. Kim, Y. Joo, "Fifth-derivative Gaussian pulse generator for UWB system," *in IEEE RF IC Symp.*, pp. 671–674, 2005.

[13] T. Phan, V. Krizhanovskii, S. G. Lee, "Low-power CMOS energy detection transceiver for UWB impulse radio system," *in IEEE Proc. CICC.*, pp. 675–678, 2007.

[14] J. Hu, et al., "A 0.17-nJ/Pulse IR-UWB receiver based on distributed pulse correlator in 0.18-μm Digital CMOS," *in IEEE RFIC Symp.*, pp. 543–546, 2009.

[15] T. Phan, V. Krizhanovskii, S. G. Lee, "Low-power CMOS energy detection transceiver for UWB impulse radio system," *in IEEE Proc. CICC.*, pp. 675–678, 2007.

[16] L. Stoica, et al., "An ultrawideband system architecture for tag based wireless sensor networks", *IEEE Trans. Veh. Technol.*, Vol. 54, No. 5, pp. 1632–1645, Sep. 2005.

[17] P. Mercier, et al., "Ultra-low-power UWB for sensor network applications," *in IEEE Proc. ISCAS*, pp. 2562–2565, 2008.

[18] F. S. Lee, A. P. Chandrakasan., "A 2.5 nJ/b 0.65V 3-to-5GHz subbanded UWB receiver in 90nm CMOS," *IEEE J. Solid-State Circuits*, pp. 116–117, 2007.

[19] H. Lee, et al., "A 15mW 69dB 2Gsample/s CMOS analog front-end for low-band UWB applications," *in IEEE Proc. ISCAS,* pp. 368–371, 2005.

[20] C. Yang, K. Chen, T. Chiueh, "A 1.2V 6.7mW impulse-radio UWB baseband transceiver," *in IEEE ISSCC Dig. Tech. Papers*, pp. 442–443, 2005.

[21] L. Xia et al., "0.15-nJ/b 3-5-GHz IR-UWB system with spectrum tunable transmitter and merged-correlator noncoherent receiver," *IEEE Trans. Microwave Theory & Tech.*, vol. 59, no. 4, pp. 1147–1156, Apr. 2011.

[22] N. Sasaki, K. Kimoto, W. Moriyama, T. Kikkawa, "A single-chip ultra-wideband receiver with silicon integrated antennas for inter-chip wireless interconnection," *IEEE J. Solid-State Circuits*, Vol. 44, No. 2, pp. 382–392, Feb. 2009.

[23] L. Xia, Y. Huang, Z.Hong, "Low power amplitude and spectrum tunable IR-UWB transmitter," *Electron. Lett.*, Vol. 44, No. 20, pp. 1200–1201, Sep. 2008.

[24] L. Liu, T. Sakurai, M. Takamiya, "A 1.28mW 100Mb/s impulse UWB receiver with charge-domain correlator and embedded sliding scheme for data synchronization," *in IEEE Symp. on VLSI Circuits*, pp. 146–147, 2009.

[25] J. Lee, H. Wang, W. Chen, Y. Lee, "Subharmonically Injection-Locked PLLs for Ultra-Low-Noise Clock Generation," *ISSCC Dig. Tech. Papers*, pp. 92–93, Feb. 2009.

[26] J. Lee, H. Wang, "Study of subharmonically Injection-Locked PLLs," *IEEE J. Solid-State Circuits*, vol. 44, no. 5, pp. 1539–1553, May. 2009.

[27] B. Razavi, "A study of injection locking and pulling in oscillators," *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1415–1424, Sep. 2004.

[28] R. Adler, "A study of locking phenomena in oscillators," *Proc. IEEE*, vol. 61, pp. 1380–1385, Oct. 1973.

[29] C. Hu et al., "A 90 nm-CMOS, 500 Mbps, 3-5 GHz Fully-Integrated IR-UWB Transceiver With multipath Equalization Using Pulse Injection-Locking for Receiver Phase Synchronization," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1076–1088, May 2011.

Author Biographies

Lingli Xia received a PhD in microelectronics and solid state electronics in 2010 from Fudan University, Shanghai, China. She is currently a postdoctoral candidate in electrical engineering at Oregon State University. Her research interests include RF front-end circuits and digital baseband circuits for wireless communication systems. She can be contacted at xia@eecs.oregonstate.edu.

Changhui Hu is an RFIC designer at Marvell Semiconductor. He received a PhD in 2011 in electrical engineering from Oregon State University. Previously he has been a research assistant in the Centre for Wireless Communications (now Institute for Infocomm Research, ASTAR) in Singapore, an RF/Analog IC designer with the startup company Advanced RFIC Singapore, and with OKI Techno Center Singapore (now known as Wipro Techno Center), where he worked on WLAN and UWB projects. Most recently he worked as a corporate application engineer at Mentor Graphics Asia, supporting Calibre products. He can be contacted at changhuihu@gmail.com.

Stephen Redfield received his master's degree in electrical engineering from Oregon State University with a focus on channel measurement and modeling. He is currently pursuing a PhD in the Department of Electrical and Computer Engineering. His research interests include UWB imaging techniques for through-wall and medical applications. He can be contacted at redfiels@engr.orst.edu.

Sirikarn Woracheewan is currently a preamp validation engineer in the Storage Peripheral Division at LSI Corporation. Her background is in analog/mixed-signal circuits and UWB wireless. She received a B.E. degree in electronics engineering from King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, and an master's degree in electrical engineering at Oregon State University. She can be contacted at woraches@onid.orst.edu.

Rahul Khanna is a platform architect at Intel Corporation involved in development of energy efficient algorithms. Over the past 17 years he has worked on server system software technologies including platform automation, power/thermal optimization techniques, reliability, optimization, and predictive methodologies. He has authored several technical papers and book chapters in the areas related to energy optimization, platform wireless interconnects, sensor networks, interconnect reliability, predictive modeling, motion estimation, and security, and holds 27 patents. He is also

the co-inventor of the Intel IBIST methodology for high-speed interconnect testing. His research interests include machine learning based power/thermal optimization algorithms, narrow-channel high-speed wireless interconnects, and information retrieval in dense sensor networks. Rahul is member of IEEE and the recipient of three Intel Achievement Awards for his contributions in areas related to advancements of platform technologies. He is the author of the book *A Vision for Platform Autonomy: Robust Frameworks for Systems*. Rahul Khanna can be reached at rahul.khanna@intel.com

Jay Nejedlo is a Senior Staff Architect and Technical Lead for Intel's Platform Validation Architecture group and has 28 years experience in silicon and platform architecture, validation, and test. He is the creator of the Intel's IBIST validation methodology as well as their system level memory sub-system BIST methodology. He architected and implemented Intel's first platform HVM OS-less functional test methodology and also founded their HALT/HASS Reliability Test Methodology. He has filed more than 20 patents over the past decade. He can be contacted with jay.nejedlo@intel.com.

Huaping Liu is currently a Professor of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR. He received the B.S. and M.S. degrees in electrical engineering from Nanjing University of Posts and Telecommunications, Nanjing, China, in 1987 and 1990, respectively, and the Ph.D. degree in electrical engineering from New Jersey Institute of Technology, Newark, in 1997. From July 1997 to July 2001, he was with Lucent Technologies, Whippany, NJ. Dr. Liu served as an Associate Editor for the IEEE Transactions on Vehicular Technology and IEEE Communications Letters from 2009 to 2011. He is currently an Editor for the Journal of Communications and Networks. He can be contacted with hliu@eecs.oregonstate.edu.

Patrick Chiang is currently an Assistant Professor of Electrical Engineering and Computer Science at Oregon State University, Corvallis, OR and a visiting professor at Fudan University, Shanghai, China. He received the B.S. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1998, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 2001 and 2007. In 1998, he was a design engineer at Datapath Systems (now LSI Logic). In 2002, he was a research intern at Velio Communications (now Rambus). In 2004, he was a consultant at Telegent Systems. He is the recipient of a 2010 Department of Energy Early CAREER award, and a 2012 NSF-CAREER award. He is an associate editor of IEEE Transactions on Biomedical Circuits and Systems, and on the technical program committee for the IEEE Custom Integrated Circuits Conference. He can be contacted with pchiang@eecs.oregonstate.edu.

NESTED QOS: ADAPTIVE BURST DECOMPOSITION FOR SLO GUARANTEES IN VIRTUALIZED SERVERS

Contributors

Hui Wang
Rice University

Kshitij Doshi
Software and Services Group, Intel Corporation

Peter Varman
Rice University

Server consolidation in virtualized data centers introduces new challenges for resource management, capacity provisioning, and guaranteeing application quality of service (QoS). The bursty nature of typical server workloads makes it difficult to provide response time guarantees without significant overprovisioning, resulting in low utilization and higher infrastructure and energy costs. In this article we present Nested QoS, a formal model that specifies application QoS by a response time distribution based on the burstiness of the workload. The workload is adaptively decomposed into classes with different response time guarantees and scheduled using an Earliest Deadline First policy. A procedure for determining the decomposition parameters is developed, and empirical results showing the benefits of decomposition and adaptive parameter setting are presented.

Introduction

"VM-based server consolidation in data centers introduces new challenges for resource management, capacity provisioning, and guaranteeing application performance."

Large virtualized data centers that multiplex shared resources among hundreds of clients form the backbone of the growing cloud IT infrastructure. The increased use of VM-based server consolidation in such data centers introduces new challenges for resource management, capacity provisioning, and guaranteeing application performance. Service level objectives (SLOs) are employed to assure client applications a specified performance quality of service (QoS), like minimum throughput or maximum response time. The service provider should allocate sufficient resources to meet the stipulated QoS goals, while avoiding overprovisioning that leads to increased infrastructure and operational costs. Accurate capacity estimation of even a single application in isolation is difficult due to the bursty nature of server workloads[9][16][20]; dynamic sharing by multiple clients further complicates the problem. Performance SLOs range from simply providing a specified floor on average throughput (for example, I/Os per second or IOPS) to providing guarantees on the response times of individual requests. Throughput guarantees can often be enforced using scheduling techniques based on fair queuing (FQ)[3][6][7][8][11]. However, guaranteeing response times[5][10][18] requires that the input workload be suitably constrained.

"The Nested QoS *service model enables clients to specify a response time distribution based on workload characteristics and pricing structure."*

In this article we propose a service model called *Nested QoS* that enables clients to flexibly specify their performance requirements in terms of a distribution of response times, based on workload characteristics and pricing structure. The model formalizes the observation that workload burstiness results in a disproportionate fraction of server capacity being used simply to handle the small tail of highly bursty requests. In the Nested QoS model, a workload is dynamically decomposed into multiple QoS classes, each with a different response time guarantee. Bursts of different intensities are identified and their requests assigned to different classes, which are isolated from each other so

that their performance can be guaranteed. In this way, requests arriving during a highly bursty period are prevented from delaying subsequent well-behaved requests. In the absence of such enforced isolation, the response times of both the bursty requests as well as the following well-behaved requests will be significantly degraded over the durations that it takes for the request backlogs to dissipate.

In earlier works[13][14][15] we described a workload decomposition scheme to identify bursts and schedule requests to reduce capacity. However, this framework is not backed up by an underlying SLO model. There are several difficulties in specifying desired performance formally with an intuitive but enforceable SLO contract. For instance, client requirements are often informally expressed by statements like "95 percent of requests must have a response time of less than 50 ms." However, such a requirement can only be met (even theoretically) if there are well-defined restrictions on the workload; otherwise, an adversarial client can arbitrarily increase the workload beyond the available capacity. Additionally, there is ambiguity over the time granularities over which such guarantees must hold, which can feed back to even more awkward and hard-to-measure restrictions on the input workload.

Performance SLO models should be intuitive, easy to monitor, and mutually verifiable in case of dispute. The Nested QoS model provides such a formal but intuitive, flexible, and enforceable way to specify the notion of graduated QoS, where a single client's SLO is specified in the form of a spectrum of response times rather than a single worst-case guarantee. The model properly generalizes SLOs based on a single response time (for example, see Cruz[5], Gulati et al.[10], and Sariowan[18]), thereby providing the opportunity for trading significant reductions in capacity requirements of the server for small changes in performance.

"Performance SLO models should be intuitive, easy to monitor, and mutually verifiable in case of dispute."

"The Nested QoS *model trades significant reduction in capacity requirements for small changes in performance."*

Our work is related to the ideas of differentiated service classes in computer networks[4][12][17]. However, we believe our model and analysis are substantially different from these works. Network QoS is largely concerned with providing throughput guarantees and reducing network congestion by anticipatory packet dropping. In contrast our focus is on providing response time guarantees by adaptive parameter estimation and capacity provisioning. Furthermore, we believe there is inherent merit in understanding how these techniques can be applied to the server environment.

In the next section, "Nested QoS Model," we describe the Nested QoS model and its implementation. Analysis of the server capacity based on the model parameters is presented in the section "Capacity Analysis of Nested QoS." In "Parameter Estimation" we describe how model parameters can be estimated based on a fast iterative simulation of a trace sample drawn from the workload. "Evaluation of Nested QoS" presents empirical results to demonstrate the benefits of Nested QoS using several block-level storage server traces. The article concludes with a summary of our findings.

Nested QoS Model

The workload W of a client consists of a sequence of requests that are sent to the server at arbitrary times. For specificity, we consider a block-level I/O workload, whose accesses have been broken into requests for fixed-size disk blocks after

filtering by the buffer cache. An arriving request is classified into one of several service classes based on the current state of the system. The service class to which the request is assigned determines its maximum response time. Classification is done based on the SLO agreement; the classifier will place a request into a class with a lower response time in preference to one with a higher one, unless doing so would violate the arrival rate specification of the SLO.

"The performance SLO is determined by multiple nested service classes with different response time guarantees."

"A token bucket regulates traffic admitted to a class based on burst and average arrival rate parameters."

In the Nested QoS model, the performance SLO is determined by multiple nested classes $C_1, C_2, \ldots, C_n$. Figure 1 is a conceptual depiction of the model for the case of three classes. A class C_i is specified by three parameters: $(\sigma_i, \rho_i, \delta_i)$, where (σ_i, ρ_i) are token bucket[17][19] parameters, and δ_i is the response time guarantee. A token bucket regulates the traffic admitted to a class based on its two parameters: the burst parameter σ and the long-term arrival rate ρ. Traffic that is compliant with a (σ, ρ) token bucket has the following property: the number of requests admitted in any interval of length t is upper bounded by $\sigma + \rho \times t$. A token bucket is used to provide an upper limit on the traffic admitted to each class in the Nested QoS model.

The requests in class C_i consist of a maximal-sized subsequence of W that is compliant with a (σ_i, ρ_i) token bucket: that is, in any interval of length t the number of requests in class C_i is upper bounded by $\sigma_i + \rho_i \times t$, and no additional request of W can be added to the sequence without violating the constraint. The token bucket provides an envelope on the traffic admitted to each class by limiting its maximum instantaneous burst size (σ_i) and arrival rate (ρ_i). All requests in C_i will be guaranteed a maximum response time of δ_i.

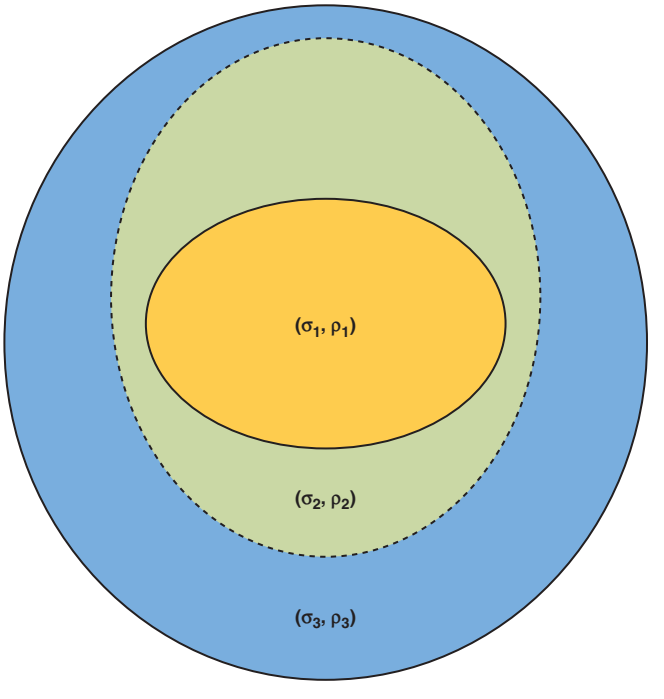


Figure 1: Nested QoS model
(Source: Rice University, 2012)

The nested nature of the classes C_i implies that all requests in C_i also belong to C_j for all $j > i$. Hence, for instance in Figure 1, all requests that are admitted to C_1 are also members of C_2 and C_3. All the requests in C_3 are guaranteed a response time δ_3. Of these requests, those that are also in C_2 are guaranteed a smaller response time δ_2, while those who make it to C_1 are guaranteed the smallest response time δ_1. Nesting of the classes requires that $\sigma_i \leq \sigma_{i+1}$, $\rho_i \leq \rho_{i+1}$ and $\delta_i \leq \delta_{i+1}$.

"The nested nature implies that all requests in C_i also belong to C_j for all $j > i$."

As an example, consider a Nested QoS model with three classes. Suppose that the parameters of C_3, C_2 and C_1 are (30, 120 IOPS, 500 ms), (20, 110 IOPS, 50 ms), and (10, 100 IOPS, 5 ms) respectively. The parameters of C_1 specify that all the requests in the workload that lie within the (10, 100 IOPS) envelope will have a response time guarantee of 5 ms; the requests within the less restrictive (20, 110 IOPS) arrival constraint have a latency bound of 50 ms, while those conforming to the (30, 120 IOPS) arrival bound have a latency limit of 500 ms.

Implementation of Nested QoS Model

Figure 2 shows a possible implementation of the Nested QoS model. It consists of two components: request classification and request scheduling (not shown

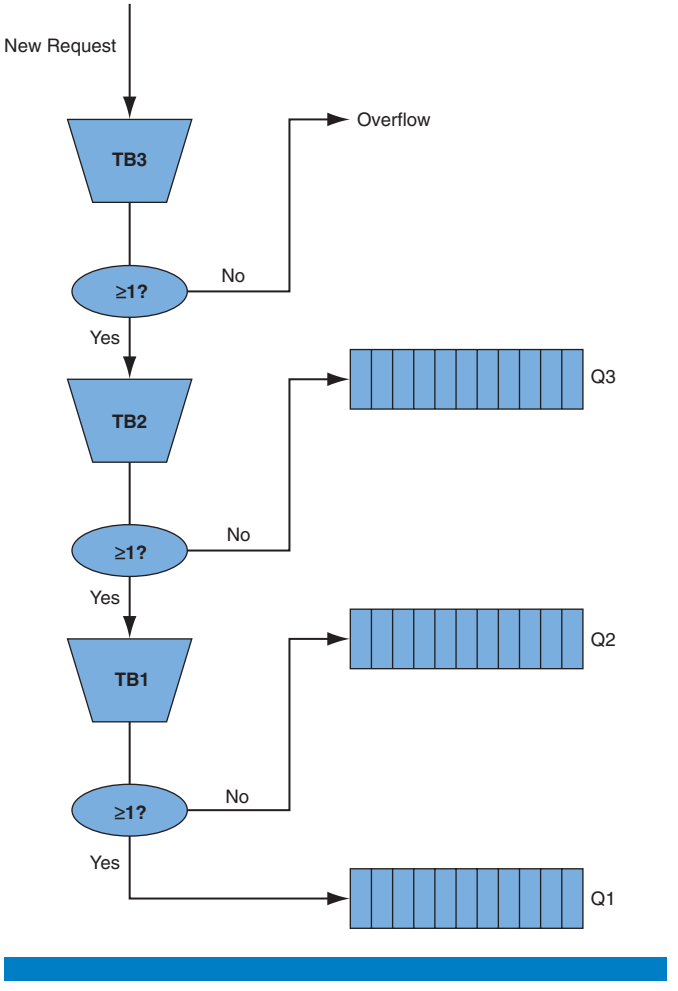


Figure 2: Cascaded token-bucket implementation (Source: Rice University, 2012)

in the figure). The classification module assigns each incoming request to the appropriate class. The scheduling module chooses the request with the earliest deadline from one of the classes to dispatch to the server when it is free.

"The request classifier is implemented using a cascade of token buckets attached to FCFS queues."

Request Classifier

The request classifier is implemented using a cascade of token buckets, B_1, $B_2, \ldots, B_n$ (innermost is B_1) attached to FCFS queues $Q_1, Q_2, \ldots, Q_n$. The buckets filter the arriving workload so that queue Q_1 receives all the requests of class C_1, Q_2 receives requests of $C_2 - C_1$, and Q_3 receives requests of $C_3 - C_2$. By ensuring that requests in queue Q_i meet a response time of δ_i, the SLO of the Nested QoS model can be met. Any requests that do not meet the arrival constraint of the outermost class are simply dropped or served on a best-effort basis. For notational simplicity, we assume a hypothetical queue Q_{n+1} that handles the overflow requests.

"In case of dispute, the SLO specification can be checked against the percentage of requests meeting different response time guarantees. "

Note that the token bucket specification is an intrinsic property of the workload based on its burst and rate characteristics, and is independent of any implementation of the Nested QoS model. In case of dispute, the workload can be profiled to find the percentage of requests that satisfied each token bucket SLO specification, and compared with the percentage of requests that actually met the response time guarantee for that class. If a client sends more requests than allowed by the SLO, the extra requests will be automatically assigned to a class with a higher response time. However, all requests within the traffic envelope of a specified class will meet their stipulated deadlines.

The token bucket parameters regulate the number of requests that pass through it in any interval. Initially bucket B_i is filled with σ_i tokens. An arriving request removes a token from the bucket (if there is one) and passes through to B_{i-1} (or Q_1 if i is 1); if there is less than one token in B_i at that time, the request goes into the queue Q_{i+1} instead. B_i is continuously filled with tokens at a constant rate ρ_i, but the maximum number of tokens in the bucket is capped at σ_i.

The algorithm for request classification is shown in Figure 3. The implementation of token bucket B_i uses four variables Sigma[i], Rho[i], NumTokens[i] and LastUpdateTime[i]. The first two are the token bucket parameters as described above. NumTokens[i] tracks the number of tokens in the bucket at any time. It is initialized to Sigma[i]; an arriving request will decrement it by 1 provided that would not make its value negative. The variable LastUpdateTime[i] tracks the time at which that bucket was last replenished with tokens. This is needed since the refilling of the token buckets will be done only at discrete times.

Procedure *RequestArrival* indicates the steps taken by the classifier when a new request arrives at time t. The classes are searched one-by-one in order, starting from the outermost class C_n, to see if the request can be admitted into that class. The request is placed in the lowest-level class that succeeds. If none of the classes can admit the request, it is simply dropped. The procedure first

```
RequestArrival(Request r, Time t)
Begin
for (i = n; i > 0; i--) {
    UpdateBucket(i, t);
    if (NumTokens[i] ≥ 1)
            NumTokens[i] = NumTokens[i] - 1;
    else
        break;
    }
    Insert r into queue Q_{i+1} with deadline t + δ_{i+1};
End
```

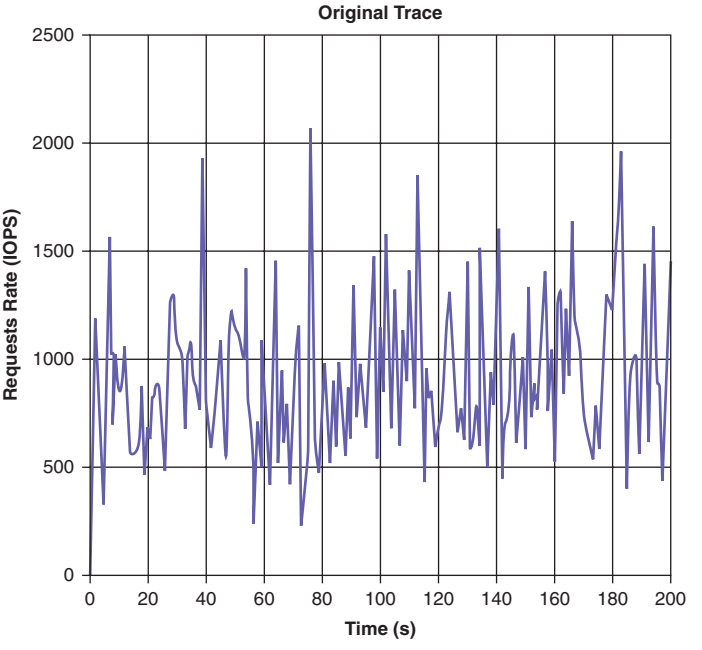
```
UpdateBucket(int BucketId, Time t)
Begin
ElapsedTime = t – LastUpdateTime[BucketId];
LastUpdateTime[BucketId] = t;
NumTokens[BucketId] += ElapsedTime * Rho[BucketId];
If (NumTokens[BucketId] > Sigma[BucketId])
     NumTokens[BucketId] = Sigma[BucketId];
End
```

Figure 3: Classification algorithm
(Source: Rice University, 2012)

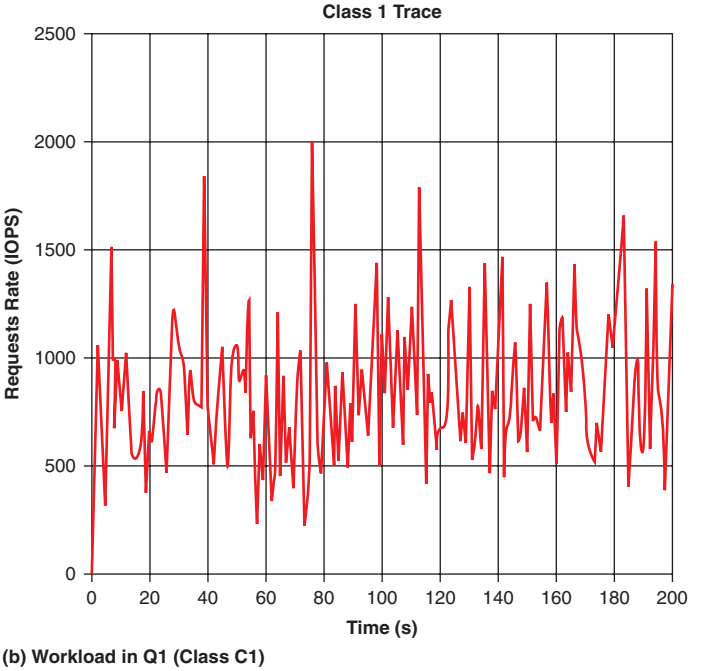
makes a call to *UpdateBucket* to replenish the bucket with tokens that have been generated since its last update. If the number of tokens in the bucket B_i is less than one, the request is not admitted into class C_i and placed in queue Q_{i+1}. The request is tagged with the deadline by which it should complete service; this is the arrival time t plus the response time guarantee for that class. Note that *NumTokens* accumulate continuously as real-valued quantities, even though they deplete as integers; and that, similarly, Sigma and Rho are, in general, real-valued quantities.

"A request is tagged with the deadline by which it should complete service."

Figure 4 shows the result of classification of a segment of the Exchange workload[2] as it goes through the token bucket network. Figure 4(a) shows the arrival pattern during the first 200 seconds of the original workload, aggregated in one-second intervals. The workload is passed through three cascaded token buckets B_1, B_2, B_3 with parameters (36, 6000), (72, 6600), and (144, 7200), respectively. The parameters are chosen so that 90 percent of the workload requests are placed in class C_1, 95 percent of the workload is classified as C_2, and 100 percent of the workload is in class C_3. Figures 4(b), 4(c), and 4(d) show the decomposed workload in classes C_1, $C_2–C_1$ and $C_3–C_2$ respectively. These portions of the workload in queues Q_1, Q_2, and Q_3 respectively will be assigned different response times, and as shown later in the section "Evaluation of Nested QoS," results in significant reduction in capacity requirements.

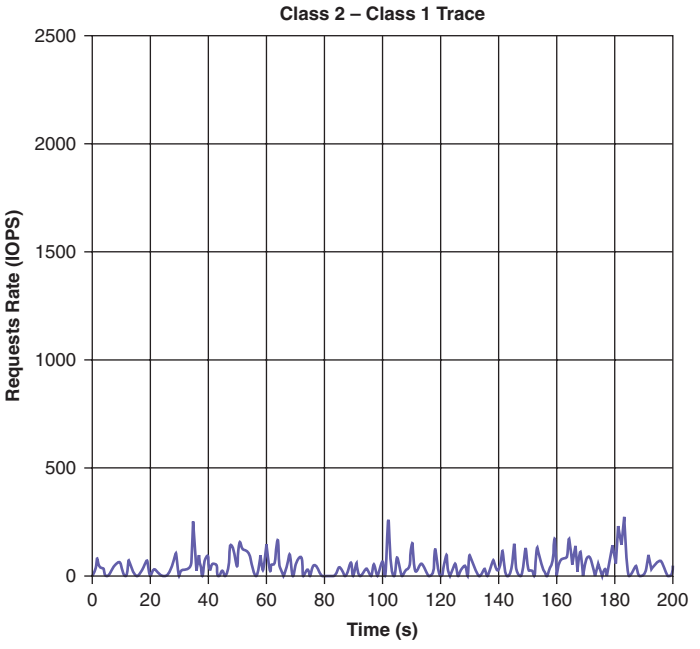


(a) Original Exchange Workload Trace

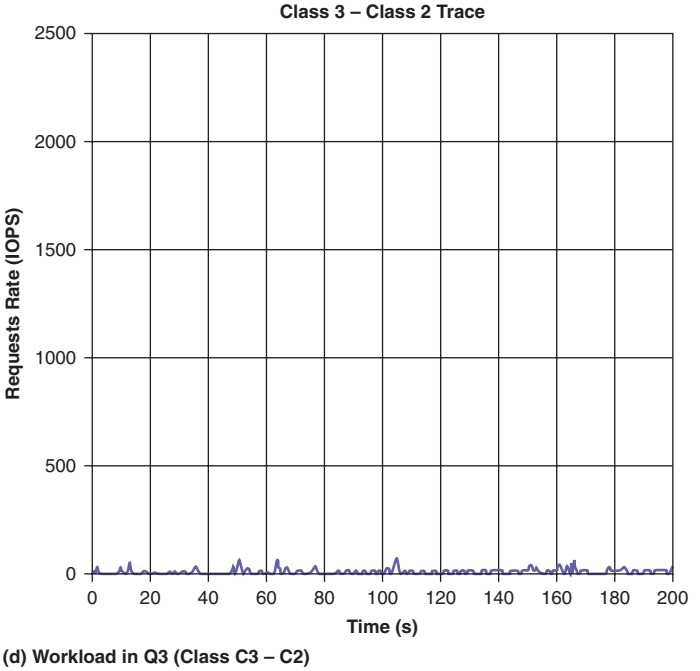


(b) Workload in Q1 (Class C1)

Figure 4 (a)-(b) Decomposition of workload trace into classes (Source: Usenix 3rd Workshop on I/O Virtualization, 2011)



(c) Workload in Q2 (Class C2 – C1)



(d) Workload in Q3 (Class C3 – C2)

Figure 4 (c)-(d) Decomposition of workload trace into classes (Source: Usenix 3rd Workshop on I/O Virtualization, 2011)

Request Scheduler

The scheduler services requests in the queues $Q_1, Q_2, \ldots, Q_n$ based on their deadlines using an earliest, deadline first (EDF) policy. Each request is tagged with a deadline when it is inserted into one of the queues. Whenever the server becomes idle, the scheduler checks the request at the head of each these queues. It dequeues the request with the smallest deadline and dispatches it to the server. Using EDF scheduling results in the smallest capacity necessary to

"EDF scheduling results in the minimum server capacity necessary to meet all deadlines."

schedule all the requests by their deadline. In the section "Capacity Analysis of Nested QoS" we will compute the capacity required to ensure that all requests admitted under the Nested QoS policy meet their response time requirements when using an EDF scheduler.

Capacity Analysis of Nested QoS

"The Capacity Theorem provides a tight upper bound on the capacity in terms of token bucket parameters."

In this section we derive an analytical formula for the capacity required to meet the response time guarantees in the Nested QoS model. The main result is stated in the Capacity Theorem that provides a tight upper bound on the capacity required to meet the specified deadlines in terms of the token bucket parameters.

Capacity Estimation

Definition 1: Define $\eta_i(t)$ to be the number of tokens in bucket B_i at time t. By definition, $\eta_i(0) = \sigma_i$ for all $i = 1, \ldots, n$.

Definition 2: Define $N_t(a, b)$ to be the maximum number of requests with deadline less than t, which enter any of the queues $Q_1, Q_2, \ldots, Q_n$ in the interval $[a, b)$.

Lemma 1 below states that bucket B_i has at most 1 token more than the number of tokens in B_{i+1} at any time. The lemma can be proved by induction over the arrival instants of requests. For the base case, the Lemma holds since $\sigma_i \leq \sigma_{i+1}$, for all $i = 1, \ldots, n-1$. The details of the proof are omitted.

Lemma 1: $\eta_i(t) \leq \eta_{i+1}(t) + 1$, for all $i = 1, \ldots, n-1$.

The Capacity Theorem upper bounds the capacity required for servicing all requests admitted into the queues Q_i of the Nested QoS model by the cascaded token buckets. The proof proceeds by upper bounding the number of requests entering the system whose deadlines are less than or equal to an arbitrary but fixed time t. These requests are partitioned into disjoint sets based on the time interval in which they arrive, and each set is associated with the set of requests admitted by a specific token bucket. By adding together the upper bounds on the number of requests admitted by each such token bucket the result will follow.

Capacity Theorem: The capacity C required scheduling all requests in the Nested QoS model satisfies:

$$C \leq \max_{j=1,\ldots,n}\{\sigma_j/\delta_j + \sum_{1\leq k<j} (1 + \rho_k(\delta_{k+1} - \delta_k))/\delta_j, \rho_j\}$$

Proof: We bound the maximum number of requests that need to finish by time t, where $t = 0$ is the start of a system busy period. Let m, $1 \leq m \leq n$, be the largest index for which $t \geq \delta_m$. Define $\tau_i = t - \delta_i$, $1 \leq i \leq m$, and for notational convenience let $\tau_{m+1} = 0$. Then $N_t(0, t) = \sum_{1\leq i\leq m} N_t(\tau_{i+1}, \tau_i)$. Now $N_t(\tau_{i+1}, \tau_i)$ consists exactly of the requests that have been admitted by bucket B_i in $[\tau_{i+1}, \tau_i)$. Hence,

$$N_t(\tau_{i+1}, \tau_i) \leq \eta_i(\tau_{i+1}) + \rho_i \times (\tau_i - \tau_{i+1}) - \eta_i(\tau_i)$$

Summing both sides for all $i = 1, \ldots, m$

$$\sum_{1 \le i \le m} N_t(\tau_{i+1}, \tau_i) \le \sum_{1 \le i \le m} \rho_i \times (\tau_i - \tau_{i+1}) + \sum_{1 \le i \le m} (\eta_i(\tau_{i+1}) - \eta_i(\tau_i))$$

Rewriting the last summation of the right hand side of the equation:

$$\sum_{1 \le i \le m} (\eta_i(\tau_{i+1}) - \eta_i(\tau_i)) = \sum_{1 \le i \le m} (\eta_i(\tau_{i+1}) - \eta_{i+1}(\tau_{i+1})) + \eta_m(\tau_{m+1}) - \eta_1(\tau_1)$$

Now, from Lemma 1, $\eta_i(\tau_{i+1}) \le \eta_{i+1}(\tau_{i+1}) + 1$ so by substituting and dropping all negative terms:

$$\sum_{1 \le i \le m} N_t(\tau_{i+1}, \tau_i) \le \sum_{1 \le i \le m} (1 + \rho_i \times (\tau_i - \tau_{i+1})) + \eta_m(\tau_{m+1})$$

Now,

$$\eta_m(\tau_{m+1}) = \eta_m(0) = \sigma_m$$

$$\tau_i - \tau_{i+1} = \delta_{i+1} - \delta_i,\ i = 1, \ldots, \mathrm{m} - 1$$

$$\tau_m - \tau_{m+1} = t - \delta_m$$

Hence,

$$\sum_{1 \le i \le m} N_t(\tau_{i+1}, \tau_i) \le \sigma_m + \sum_{1 \le i \le m-1} (1 + \rho_i \times (\delta_{i+1} - \delta_i)) + \rho_m \times (t - \delta_m)$$

The capacity (C) required to finish these $N_t(0, t)$ requests by time t is upper bounded by $N_t(0, t)/t$. Hence:

$$C \le \sigma_m/t + \sum_{1 \le i < m} (1 + \rho_i \times (\delta_{i+1} - \delta_i))/t - \rho_m \times \delta_m/t + \rho_m$$

Now, if $(\sigma_m + \sum_{1 \le i < m} (1 + \rho_i \times (\delta_{i+1} - \delta_i))) < (\rho_m \times \delta_m)$ the inequality reduces to: $C \le \rho_m$. Otherwise, the RHS is maximized when t takes on its smallest value, which is δ_m. In this case, the inequality reduces to:

$$C \le \sigma_m/\delta_m + \sum_{1 \le i < m} (1 + \rho_i \times (\delta_{i+1} - \delta_i))/\delta_m$$

The above two inequalities must hold for all values of t, and hence for all possible values of m, $1 \le m \le n$.

Putting it all altogether we get:

$$C \le \max_{m=1,\ldots,n} \{\sigma_m/\delta_m + \sum_{1 \le i < m} (1 + \rho_i(\delta_{i+1} - \delta_i))/\delta_m, \rho_m\}$$

In an ideal situation, if the tokens are updated only in integer units, Lemma 1 will be simplified to $\eta_i(t) \le \eta_{i+1}(t)$ for all $i = 1, \ldots, n - 1$; and the Capacity Theorem will be simplified to $C \le \max_{j=1,\ldots,n} \{\sigma_j/\delta_j + \sum_{1 \le k < j} \rho_k(\delta_{k+1} - \delta_k)/\delta_j, \rho_j\}$. We will use this ideal case in the rest of the article. The following corollaries consider special cases of the above Theorem that provide for simplified capacity equations[21]. The first result considers the case when all the token buckets have the same rate ρ, and the second considers an interesting case when the parameters of the token buckets are multiples of a base value.

Corollary 1.1: The capacity required for all requests to meet their deadlines in the Nested QoS model, when all ρ_i are equal to ρ, is given by: $\max_{1 \le j \le n} \{\sigma_j/\delta_j + \rho \times (1 - \delta_1/\delta_j), \rho\}$.

Corollary 1.2: Let all ρ_i be equal to ρ, and $\alpha = \delta_{i+1}/\delta_i$, $\beta = \sigma_{i+1}/\sigma_i$ and $\lambda = \beta/\alpha$ be constants. The server capacity required to meet SLOs is no more

than: $\max_{1\le j\le n}\{\rho, \lambda^j \times (\sigma_1/\delta_1) + \rho \times (1 - 1/\lambda^j)\}$. For $\lambda < 1$, the server capacity is bounded by $\sigma_1/\delta_1 + \rho$, which is less than twice the capacity required for servicing the innermost class C_1.

The final corollary asserts that using an EDF scheduler, the capacity defined in the Capacity Theorem is sufficient to meet all deadlines. We omit the details of a simple proof by contradiction.

Corollary 2: If the server has capacity at least that derived in the Capacity Theorem, and requests are scheduled using an EDF policy, then all requests will meet their deadlines.

We finally show that the Capacity Theorem provides a tight upper bound by demonstrating a workload that requires the derived capacity in order to meet the stipulated deadlines. The adversarial workload consists of a burst of size σ_n at time $t = 0$, followed by a continuous request stream arriving at the uniform rate ρ_n. Clearly, the capacity should be at least ρ_n, since otherwise one more of the queues Q_i, $i = 1, \ldots, n$, will grow without bound.

The total number of requests that arrive in the interval $[0, t]$ is $(\sigma_n + t \times \rho_n)$. All these requests will be admitted by the outermost token bucket, and will be distributed among the queues as follows: Q_i, $i = 1, \ldots, n$, will receive $(\sigma_i - \sigma_{i-1}) + t \times (\rho_i - \rho_{i-1})$ requests, where σ_0 and ρ_0 are defined to be 0. Consider the number of these requests that have a deadline δ_m, for arbitrary but fixed m, $1 \le m \le n$. These will be the requests in queues Q_j, $1 \le j \le m$, that arrived during the interval $[0, \delta_m - \delta_j]$. The number of such requests in Q_j is $(\sigma_j - \sigma_{j-1}) + (\delta_m - \delta_j) \times (\rho_j - \rho_{j-1})$. Summing over all the queues Q_1 to Q_m, the total number of requests with deadline δ_m is:

$$\sum_{1\le j\le m} (\sigma_j - \sigma_{j-1}) + (\delta_m - \delta_j) \times (\rho_j - \rho_{j-1}) = \sigma_m + \sum_{1\le j\le m} \rho_j \times (\delta_{j+1} - \delta_j)$$

The minimum capacity required to finish these requests by δ_m is $\sigma_m/\delta_m + \sum_{1\le j<m} \rho_j \times (\delta_{j+1} - \delta_j)/\delta_m$. This is within a small additive term of the capacity bound, showing that the capacity required to service this workload matches the Capacity Theorem.

Parameter Estimation

"By profiling, workload requirements are translated to token bucket parameters and capacity estimates."

We now describe how the Nested QoS parameters of a workload will typically be determined. The client decides the number of classes, the fraction of the workload in each class, and the response time requirement for the class. By profiling the workload the provider translates these requirements to token bucket parameters and capacity estimates for the workload.

We consider in detail the case of two guaranteed classes C_1 and C_2, satisfying fractions f_1 and f_2 of the workload and having response time guarantees δ_1 and δ_2. First we estimate the capacity C_i required for fraction f_i of the workload to meet deadline δ_i, for $i = 1$ and $i = 2$ independently. This can be found by simulating the arrivals to a fixed-length queue (of size $C_i \times \delta_i$) and using

a greedy drop algorithm to handle queue overflow (see [14]). The capacity is varied (using a binary-search like method) till the fraction of requests overflowing the queue falls just below $1 - f_i$. The maximum of C_1 and C_2 is a lower bound on the capacity required by Nested QoS.

The token bucket parameters are chosen to minimize the capacity required by the Capacity Theorem. Figure 5 describes the iterative procedure for a two-class Token Bucket system. The capacity in this case is given by the maximum of: $\{\sigma_1/\delta_1, \sigma_2/\delta_2 + \rho_1(1 - \delta_1/\delta_2), \rho_1, \rho_2\}$. To simplify this estimation, at each candidate capacity point, we let σ_1 and ρ_2 assume the largest values compatible with the chosen capacity, and search for σ_2 and ρ_1 that satisfy the $\{\delta_i, f_i\}$ objectives. This search is carried out by iterative trace simulation.

"The token bucket parameters are chosen to minimize the capacity bound in the Capacity Theorem."

We begin with a capacity estimate C starting with the lower bound described above. We select the largest possible σ_1 that with capacity C can meet a

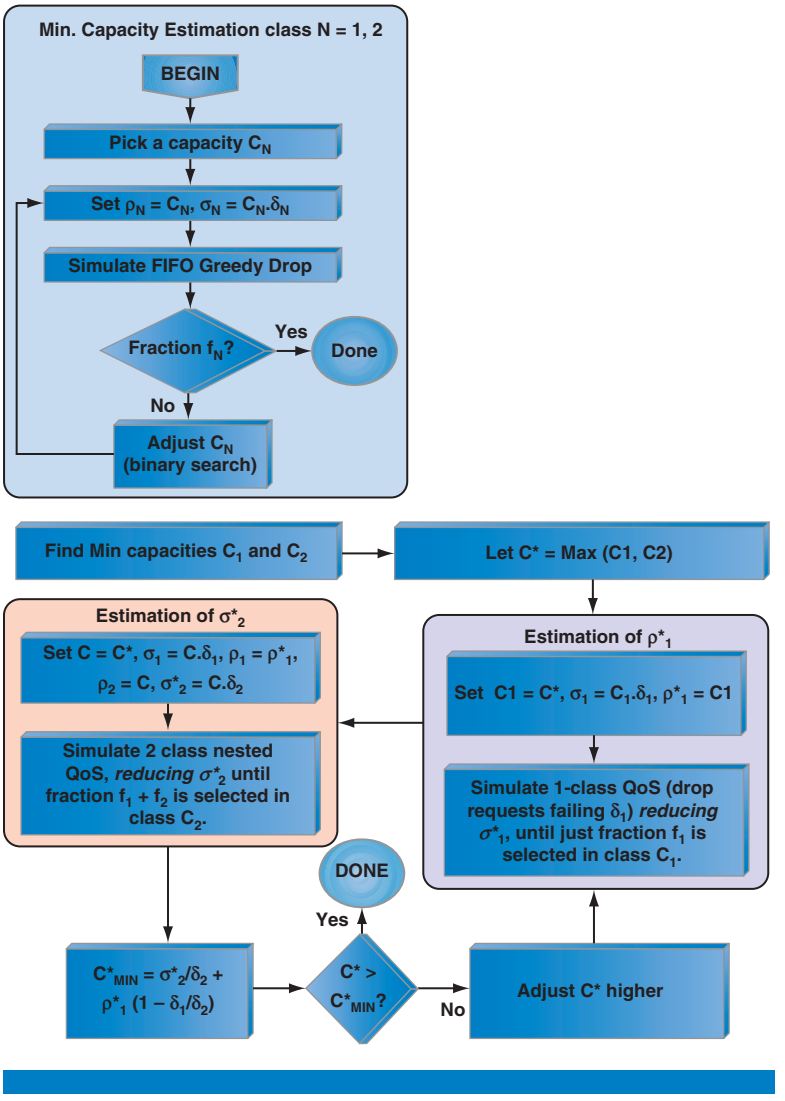


Figure 5: Iterative Calculation of Token Bucket Parameters and Capacity (Source: Rice University, 2012)

deadline δ_1, that is, $\sigma_1 = C \times \delta_1$. Next, we find the smallest value of ρ_1 that along with the chosen value of σ_1 allows a fraction f_1 of the workload to pass bucket B_1. We choose ρ_2 equal to C and then find the smallest σ_2 that along with the chosen value of ρ_2 allows a fraction f_2 of the workload to pass bucket B_2. We compute the capacity C' required by the Capacity Theorem using these parameters. If $C' > C$, we increase C and repeat the procedure; else the required capacity is C and the token bucket parameters are as determined.

"It may be preferable to change the parameters adaptively to react to significant changes in workload behavior."

The capacity and token bucket parameters for a workload can be determined by off-line profiling of workload traces. These settings are then used during actual runtime operation. Such an approach is suitable for workloads that are relatively stable and whose overall statistical profile does not vary substantially from run to run. On the other hand, in situations where there may be periodic or unexpected changes in the workload during operation, it may be preferable to change the parameters adaptively to react to significant changes in workload behavior. In this environment, a monitoring agent triggers an alarm when the performance changes significantly; it may be sufficient to use a coarse measure like smoothed average latency rather than the exact SLO specifications to check for such situations. A runtime profiler is invoked to determine the new parameters necessary to meet SLO specifications with the changed workload characteristics, and additional capacity is requested for the workload if necessary. If the capacity request is granted, the token bucket parameters are changed based on the newly profiled values. In the section "Evaluation of Adaptive Parameter Setting" we evaluate the impact of adaptively setting parameters based on profiling a sample prefix of a workload.

Evaluation of Nested QoS

We implemented the Nested QoS model in a process-driven system simulator and evaluated the performance separately with five block-level storage workload I/O traces from the UMass Storage Repository[1] and SNIA IOTTA Repository[2]: WebSearch1(W1), WebSearch2(W2), FinTrans(W3), OLTP(W4), and Exchange(W5). W1 and W2 are traces from a web search engine and consist of user web search requests. W3 and W4 are traces generated by financial transactions running at large financial institutions. W5 trace is from a Microsoft Exchange* Server. The parameters for each workload are shown in Table 1 below. The values were found by profiling the workloads to guarantee at least 90 percent requests in class C_1.

	W1	W2	W3	W4	W5
σ_1	4.0	4.0	3.0	2.0	36.0
ρ_1 **(IOPS)**	450	430	300	250	3600
δ_1 **(ms)**	10.0	10.0	10.0	10.0	10.0

For all workloads: $\rho_{i+1} = \rho_i$, $\sigma_{i+1} = 2\sigma_i$, $\delta_{i+1} = 10\delta_i$.

Table 1: QoS Parameters for Simulated Workloads
(Source: Rice University, 2012)

Capacity and Performance Tradeoffs

Figure 6 compares the capacity required by the workloads for the Nested and Single-Level QoS models. The latter requires all requests to meet the δ_1 response time. The capacity is significantly reduced by spreading the requests over multiple classes. Figure 7 shows the distribution of response times. In each case a large percentage (90–92 percent) of the workload meets the 10-ms response time bound, and (except for FT workload) only a small 0.5 percent (or less) requires more than

"The capacity is significantly reduced by spreading the requests over multiple classes."

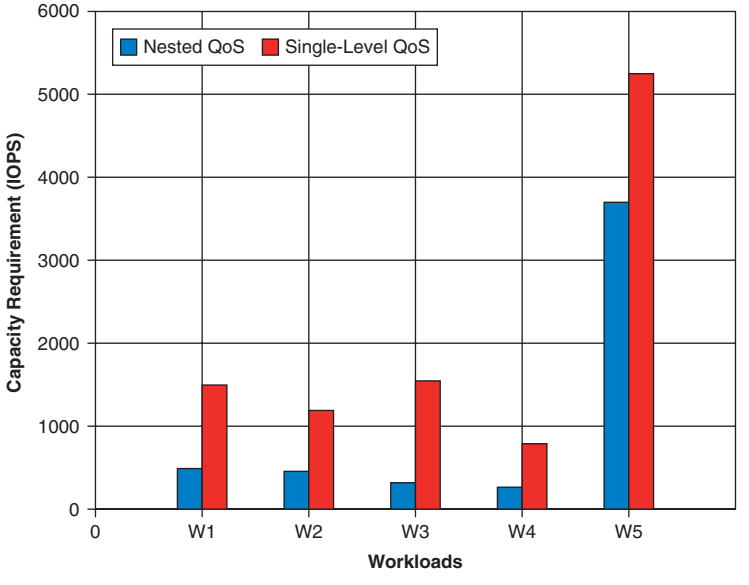


Figure 6: Capacity requirement for Nested QoS and Single level QoS (Source: Rice University, 2012)

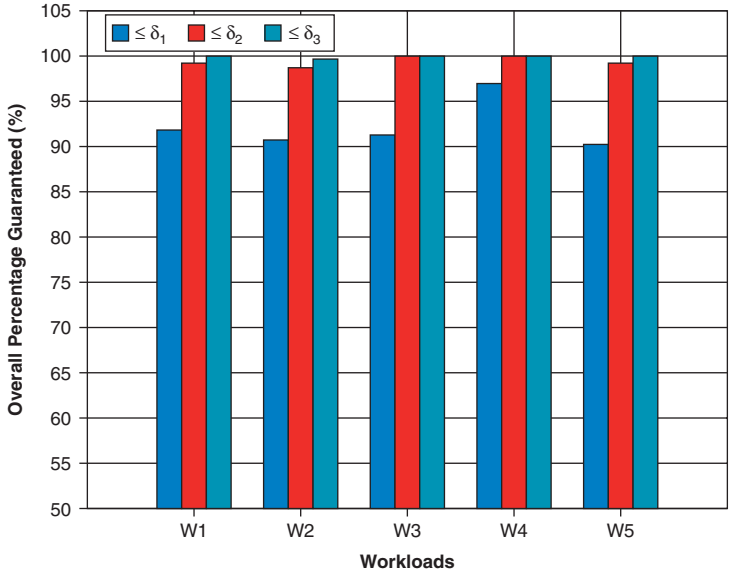


Figure 7: Performance for Nested QoS (Source: Rice University, 2012)

100 ms. The capacity required for Nested QoS is several times smaller than that for Single-Level QoS, while the service seen by the clients is only minimally degraded.

"In a shared environment, each workload is independently decomposed into classes based on its Nested QoS parameters."

Multiplexing Multiple Workloads

In a shared environment, each workload is independently decomposed into classes based on its Nested QoS parameters. The server provides capacity Φ_j for workload j based on its capacity estimate using the formula in the section "Capacity Analysis of Nested QoS," and provisions a total capacity of $\sum_j \Phi_j$. A standard proportional scheduler [3, 7] allocates the capacity to each workload in proportion to its Φ_j. When workload j is scheduled, it chooses the request from its class queues with the smallest deadline. Figure 8 shows the organization for serving multiple clients.

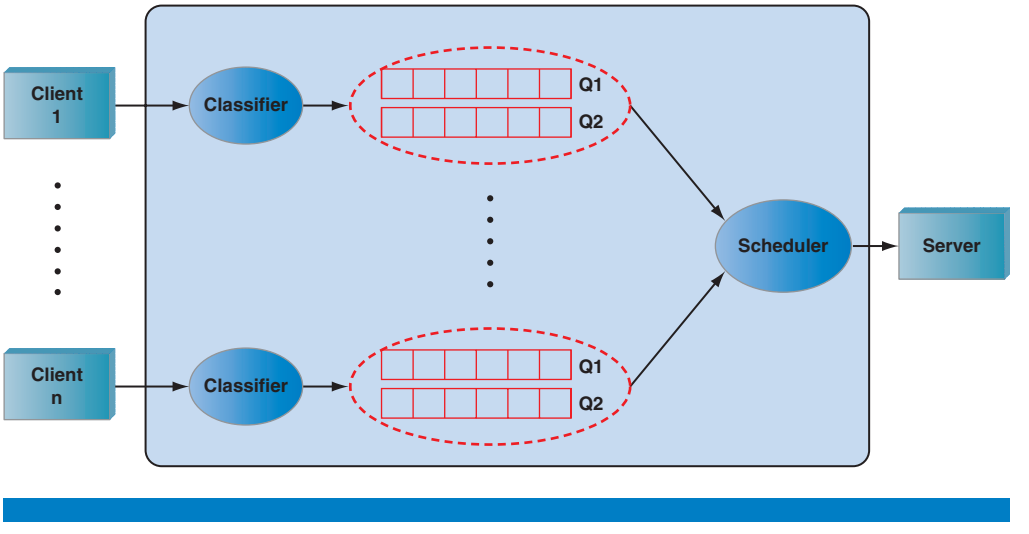


Figure 8: Nested QoS model for multiple workloads
(Source: Rice University, 2012)

"Scheduling the queues globally using EDF makes it difficult to direct capacity changes to specific workloads."

An alternative to using a proportional scheduler is to use EDF scheduling globally across the queues of all the clients. The advantage of using global EDF scheduling is its ability to exploit the heterogeneity of the workloads to reduce the overall capacity requirements[5][10]. On the other hand, scheduling the queues globally using EDF makes it difficult to direct capacity changes to specific workloads[22]. A drop or increase in system capacity could be allocated unfairly to the workloads based on internal timing dynamics of the scheduler. In contrast, a proportional scheduler always allocates capacity based on the individual Φ_j settings of the workload.

In the following two sections we illustrate two basic properties of the Nested QoS framework: intra-client robustness to workload variation and inter-client isolation. We compare Nested QoS to two other well-known scheduling approaches: pClock[10] that uses EDF to guarantee response times of requests, and WF^{2}Q[3] that is used for proportional share scheduling.

Robustness to Workload Violation

In the experiment, we use the two block-level workloads: W1 and W2. W1 is a financial transaction workload with a long-term average arrival rate of about 115 IOPS; W2 is a proxy workload with a long-term average arrival rate of around 21 IOPS.

The arrival patterns of the two workloads are shown in Figure 9(a). By profiling the workloads, the token bucket parameters for the three classes of W1 are set to (7, 130 IOPS), (14, 143 IOPS) and (50, 158 IOPS), while the parameters for the token buckets of W2 are set to (6, 120 IOPS),

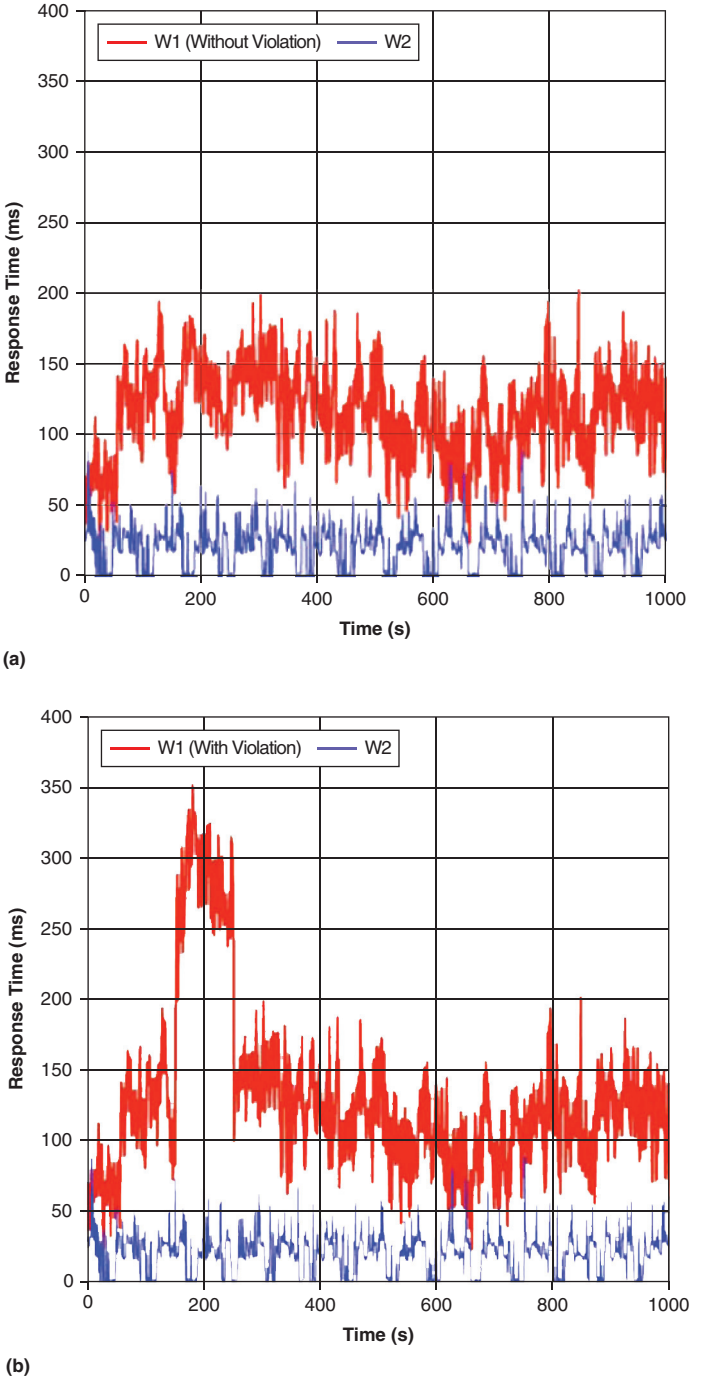


Figure 9: Arrival pattern for (a) Both W1 and W2 are well behaved. (b) W1 violates SLAs and sends more requests during time 150–250 s
(Source: Rice University, 2012)

(15, 125 IOPS), and (50, 130 IOPS). A system capacity of 276 IOPS is provisioned for the two workloads. With this capacity, all three methods (Nested QoS, pClock, and WF^{2}Q) can guarantee that at least 90 percent of the requests finish within a 50 ms deadline, and 95 percent of the requests finish within a deadline of 500 ms.

In a second experiment shown in Figure 9(b), W1 is perturbed by artificially injecting additional traffic. Specifically, the instantaneous arrival rate of W1 is increased to around 260 IOPS between times 150–250 seconds. During this period its arrival rate exceeds its long-term average, and violates the stipulated SLO based on the original W1 workload. The violation is relatively small and corresponds to less than about 10 percent of the entire trace.

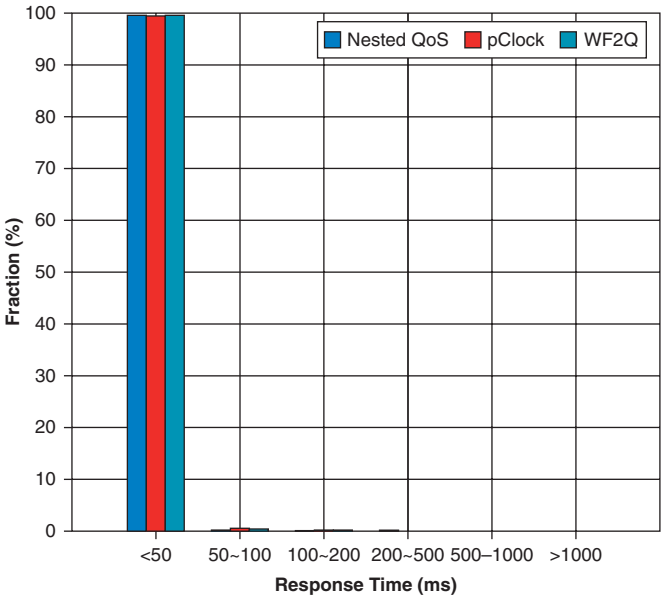
First we will explore how Nested QoS isolates the bad regions of a workload where the instantaneous traffic rate exceeds stipulated SLO-based arrival rates. This isolation protects the good regions of the workload from the delay caused by the burst and maximizes the number of requests that meet their deadlines. A sever capacity of 276 IOPS is provided for all the three scheduling methods being evaluated.

Figure 10(a) shows the performance of the unmodified workload W1 using the three scheduling algorithms. As can be seen, with any of the schedulers more than 90 percent of the requests finish within the stipulated 50 ms response time bound. However, the picture changes significantly when a portion of the workload behaves badly. Figure 10(b) shows the response time distribution for the modified W1, which sends extra requests during the 150–250-second interval. All methods show a degradation in performance in this situation, but the degradation is different in the three cases. Nested QoS still allows 90 percent of the requests to meet their 50 ms deadline; however pClock and WF^{2}Q are noticeably degraded, and only about 76 percent of their requests meet the 50 ms deadline. The majority of requests that miss the deadline in the latter two schemes are delayed significantly, with response times exceeding 1 second. On the other hand, the roughly 10 percent of requests missing their deadline in Nested QoS still receive reasonable service and have response times roughly uniformly distributed between 50 ms and 1 s, since they will be assigned to classes C_2 and C_3 before being relegated to best effort service.

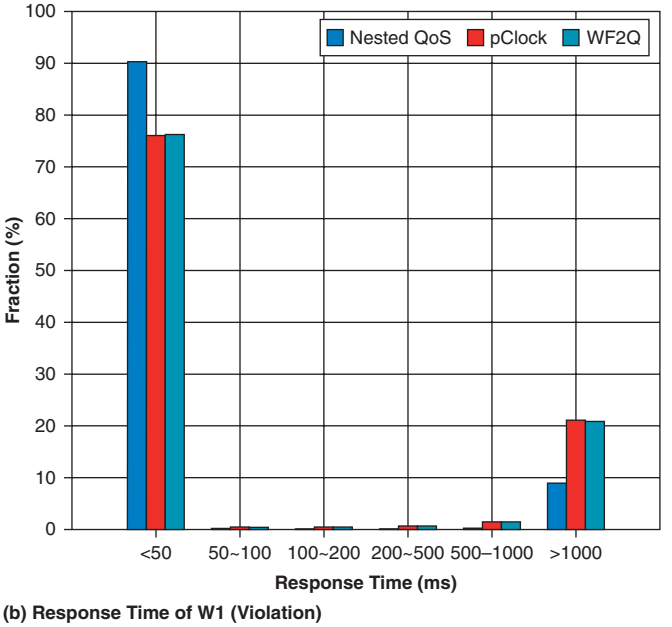
"With Nested QoS most of the requests during this interval still meet their deadline."

"Nested QoS diverts extra requests to higher-level classes, isolating them from the well-behaved requests."

The measured response times during and after the badly-behaved region are shown in Figures 11(a) and (b) for the Nested QoS and pClock schedulers respectively. As can be seen, with Nested QoS most of the requests during this interval still meet their deadline, and only a few of them have longer response time. The well-behaved requests both before and after $t = 150$ s are not affected by the extra requests. In contrast, pClock delays all the requests of W1 not only during the interval (150–250)s, but all the way after the burst to about 270 s. This is because when the violation happens, Nested QoS diverts the extra requests to the higher level classes C_2 and C_3, isolating them from the well-behaved requests, allowing them to meet their guaranteed deadlines.



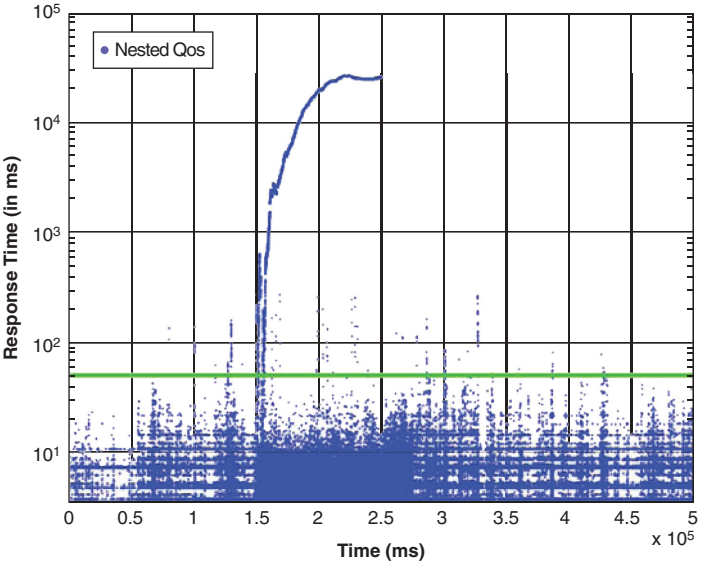
(a) Response Time of W1



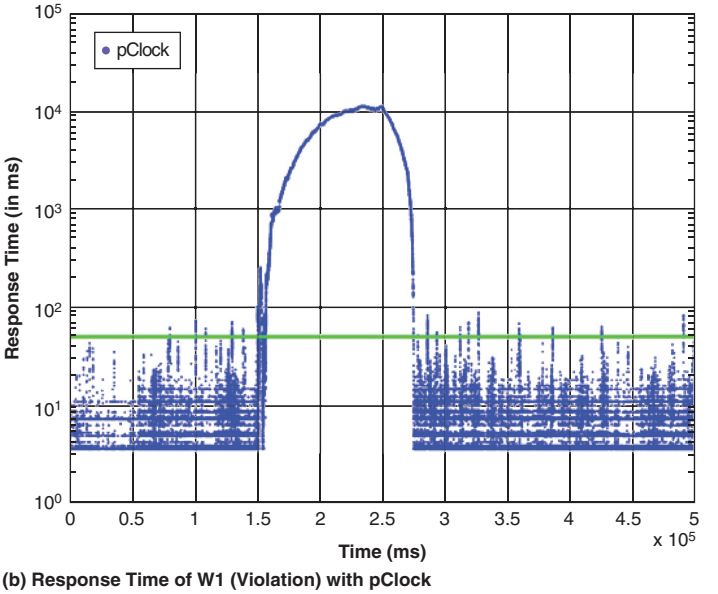
(b) Response Time of W1 (Violation)

Figure 10: Response time distribution for W1 (well-behaved) and W1 (with violation) with three scheduling methods: Nested QoS, pClock, WF^{2}Q.
(Source: Rice University, 2012)

The performance of W2 is the same for both the original and the modified W1 workload. We do not show the performance of W2 here because it is isolated from W1 as discussed in the next section.



(a) Response Time of W1 (Violation) with Nested QoS



(b) Response Time of W1 (Violation) with pClock

Figure 11: W1 violates its SLA and sends more requests from 150 s to 250 s. Nested QoS isolates the bad region and still guarantees the well-behaved part. However pClock delays all of W1's requests from 150 s all the way up to 270 s. (Source: Rice University, 2012)

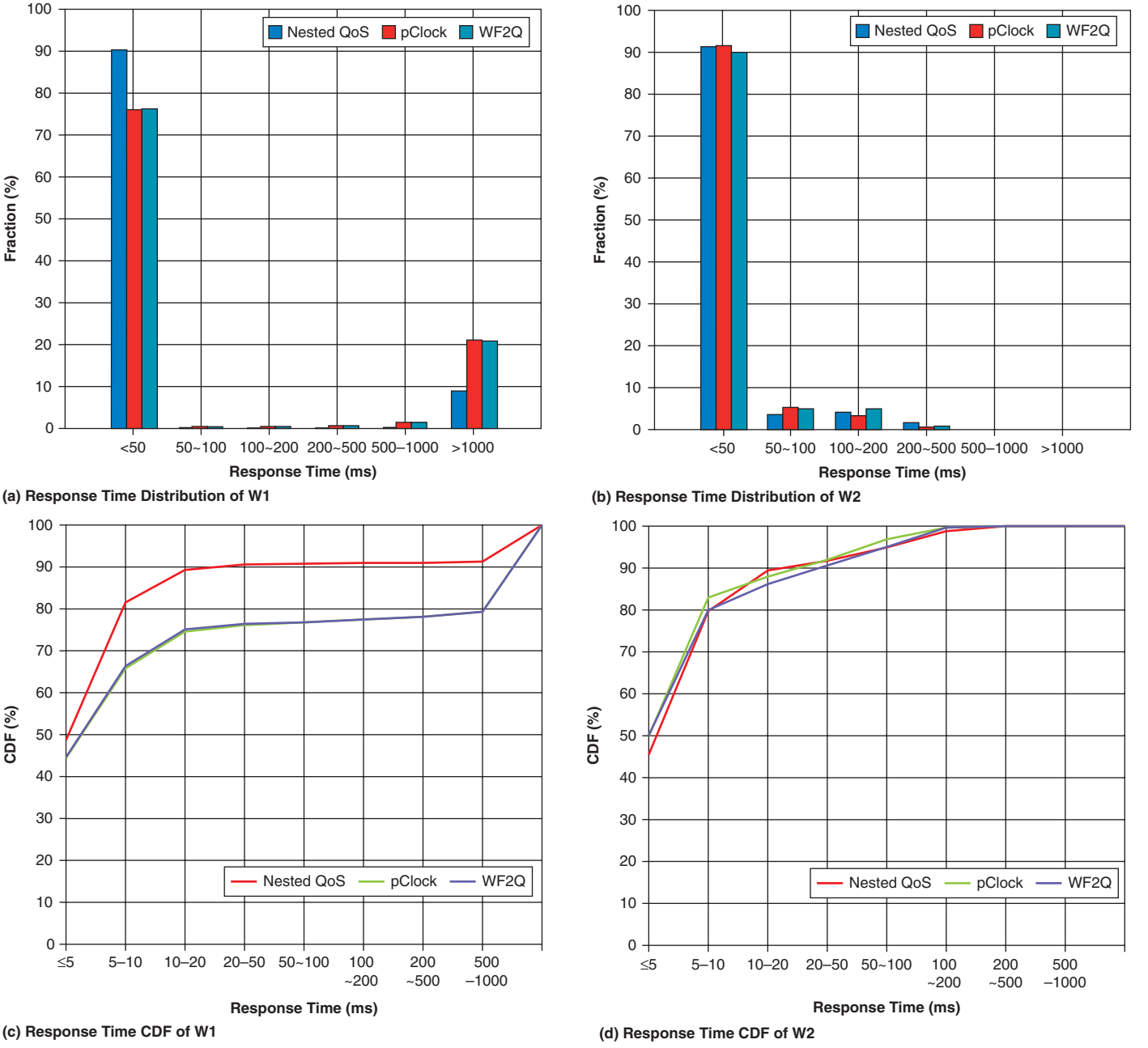
"Nested QoS has the ability to isolate the bad regions of a workload and protect subsequent well-behaved portions."

In general, Nested QoS outperforms the other two methods because of its ability to isolate the bad regions of a workload and protect subsequent well-behaved portions from their effects. In contrast, traditional fair schedulers isolate workloads from each other but cannot protect a workload from its own bad behavior. Hence, a local violation in a small area of the workload can result in performance degradation over a sizable extended portion of the workload.

Workload Isolation

Workload isolation is a basic requirement in shared server systems. In this experiment, we verify that Nested QoS can isolate well-behaved workloads from badly behaved ones. We look at the performance of the well-behaved workload W2 when W1 violates arrival requirements. A good method should insulate W2 from the bad behavior of W1 and guarantee its performance. Figures 12(a) and (b) show the response time histogram of the badly behaved W1 and W2 with the three scheduling methods: Nested QoS, pClock, and WF^{2}Q. Figures 12(c) and (d) show the response time cumulative distributions. We can see that

"Workload Isolation is a basic requirement in shared server systems."



(a) Response Time Distribution of W1

(b) Response Time Distribution of W2

(c) Response Time CDF of W1

(d) Response Time CDF of W2

Figure 12: Response time distribution and CDF of W1, W2 with three scheduling methods: Nested QoS, pClock, and WF^{2}Q (Source: Rice University, 2012)

"The parameter estimation method can be used to adapt the capacity distribution in response to changes in request arrival patterns."

the well-behaved workload W2 is isolated from the bad behavior of W1. The performance of W2 does not change when W1 sends more requests.

Evaluation of Adaptive Parameter Setting

In the section "Parameter Estimation," we described an iterative procedure for selecting the token bucket parameters and estimating minimum capacities. Because of its fast convergence in determining the Nested QoS parameters, the method of parameter estimation described earlier can be used to adapt the capacity distribution from an elastic server, in response to changes in request arrival patterns. In this section we describe the results of dynamically setting Nested QoS token bucket parameters by profiling a short segment of a workload. For the experiment we used the first Financial Trace (FT) as the baseline. In order to emulate dynamic changes to the workload, the trace was speeded up twofold and threefold to obtain the modified traces FT^2 and FT^3 respectively.

Each workload consisted of first 100,000 requests from the original FT trace. This 100,000 request portion was split up into 10 segments of 10,000 requests each. The first segment ("base FT trace") was used a training segment for the remaining nine segments; the token bucket parameters were estimated by profiling this segment using the procedure described in "Parameter Estimation." The entire trace was then simulated with these parameters, and the percentage of requests meeting their SLO-stipulated deadlines was measured. The results were then compared with the situation when the training was done statically based on the original, non-speeded-up baseline FT trace. For the experiment, the SLO required 90 percent of the requests workload to meet a 20 ms deadline and 95 percent to meet a 40 ms deadline.

Figures 13 and 14 show the performance of FT^2 and FT^3 workloads in the two situations. In Figure 13(a) the percentage of requests meeting the 20 ms deadline is shown for FT^2 in two cases: (1) when the parameter estimation is done statically (static trained) and (2) when the training is dynamic based on the first segment of FT^2 (dynamically retrained). With the static training the percentage of the workload complying with the SLO is between 70 percent and 85 percent compared to 90 percent in the adaptive case. Note that the SLO was set to achieve 90 percent in Class 1, so the adaptive training based on the first segment does a good job in this case. Figure 14(a) shows a similar comparison for FT^3. In this case, the difference between the static and adaptive cases is more pronounced, with only between 50 percent and 70 percent of the requests meeting the SLO deadline for the static case versus the expected 90 percent in the adaptive case.

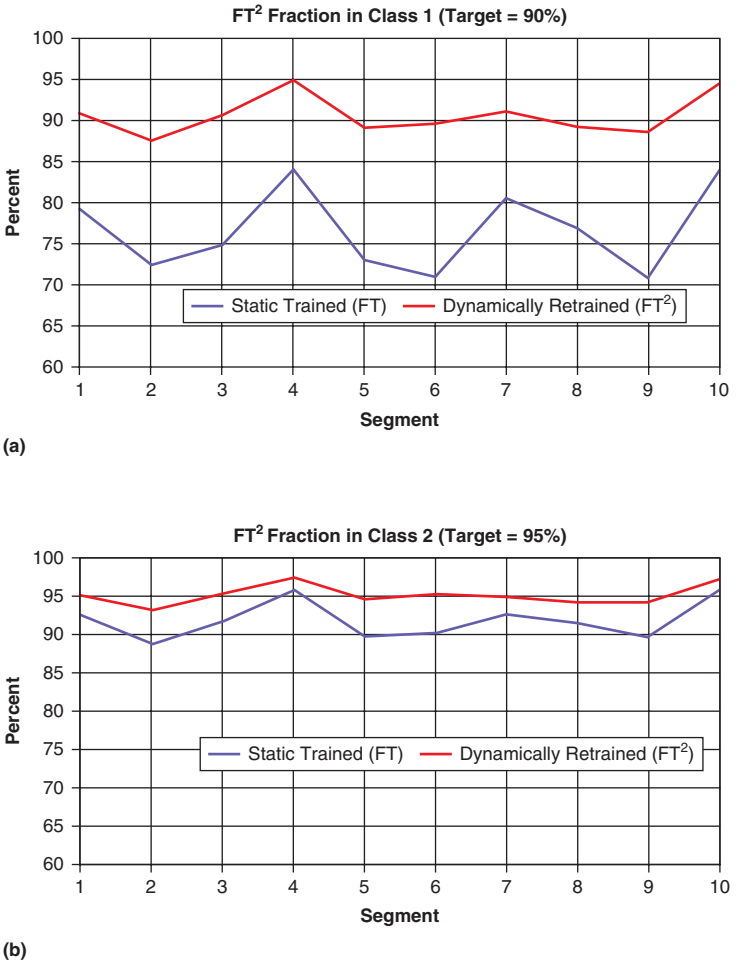


Figure 13: Workload FT². Percentage of requests meeting (a) Class 1 response time limit of 20 ms and (b) Class 2 response time limit of 40 ms. SLO objectives are 90% for Class 1 and 95% for Class 2. (Source: Rice University, 2012)

Figures 13(b) and 14(b) show the results for Class 2, where a similar behavior between static and adaptive parameter settings can be observed. We also conducted experiments with different response times, 10 ms and 20 ms for classes 1 and 2 respectively, which are not reported. The trends in that case were similar though the differences were smaller. The smaller differences are understandable: the stricter response times of this experiment translated to having a larger baseline capacity (in order to meet the more stringent deadlines). The larger baseline capacity provided greater slack for the statically trained parameter set, and therefore produced smaller differences than the experiment reported in Figures 13 and 14.

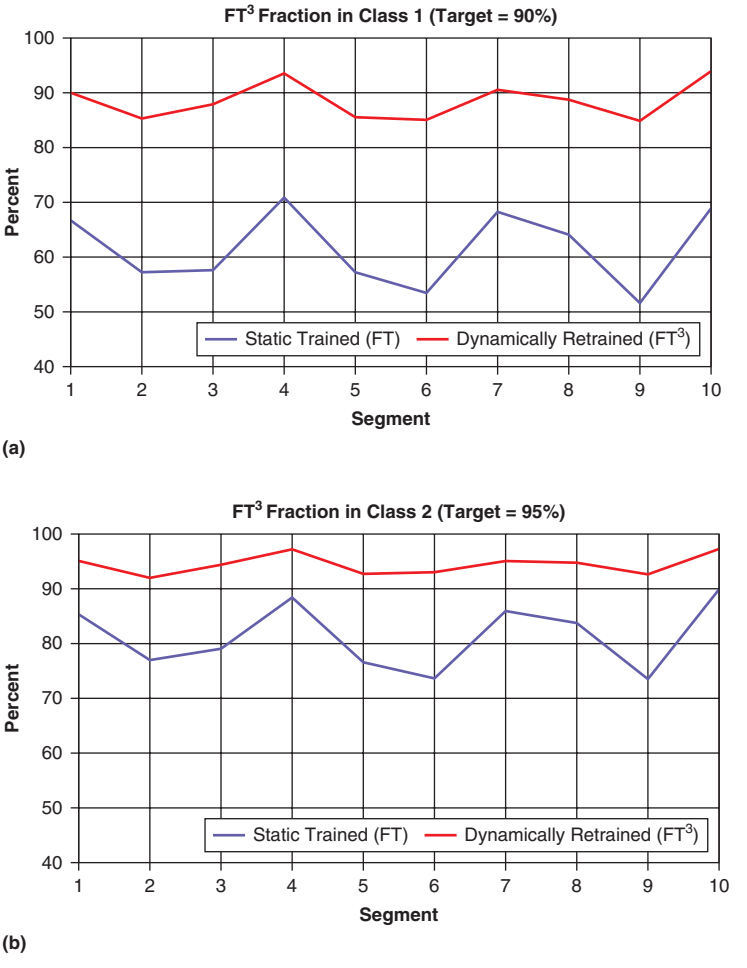


Figure 14: Figure 14: Workload FT³. Percentage of requests meeting (a) Class 1 response time limit of 20 ms and (b) Class 2 response time limit of 40 ms. SLO objectives are 90% for Class 1 and 95% for Class 2 (Source: Rice University, 2012)

Summary

The Nested QoS model provides several advantages over usual SLO specifications: (1) large reduction in server capacity without significant performance loss (2) accurate analytical estimation of the server capacity (3) providing flexible SLOs to clients with different performance/cost tradeoffs, and (4) providing a clean conceptual structure of SLOs using workload decomposition. Our work continues to explore relating workload characteristics with the nested model parameters, generalized parameter estimation and optimization within the framework of adaptive control theory, alternative scheduling strategies for multiple decomposed workloads to exploit statistical multiplexing, and Linux block-level implementation.

Acknowledgements

The research of H. Wang and P. Varman was partially supported by NSF Grants CNS 0917157 and CCF 0541369.

References

[1] Storage Performance Council (UMass Trace Repository), 2007. http://traces.cs.umass.edu/index.php/Storage.

[2] SNIA: IOTTA Repository, 2009. http://iotta.snia.org.

[3] J. C. R. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queuing. In INFOCOM 1996, pages 120–128, March, 1996.

[4] C.-S. Chang. Performance guarantees in communication networks. Springer-Verlag, London, UK, 2000.

[5] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. IEEE Journal on Selected Areas in Communications, 13(6):1048–1056, 1995.

[6] S. Golestani. A self-clocked fair queuing scheme for broadband applications. In INFOCOMM 1994, pages 636–646, April 1994.

[7] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queuing: a scheduling algorithm for integrated services packet switching networks. IEEE/ACM Transactions on Networking, 5(5):690–704, 1997.

[8] A. G. Greenberg and N. Madras. How fair is fair queuing. Journal ACM, 39(3):568–598, 1992.

[9] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT '09), 2009.

[10] A. Gulati, A. Merchant, and P. Varman. pClock: An arrival curve based approach for QoS in shared storage systems. In ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), June 2007.

[11] A. Gulati, A. Merchant, and P. Varman. mClock: Handling throughput variability for Hypervisor IO scheduling . In 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Oct. 2010.

[12] J.-Y. Le Boudec and P. Thiran. Network Calculus: a theory of deterministic queuing systems for the Internet. Springer- Verlag, Berlin, Heidelberg, 2001.

[13] L. Lu, K. Doshi, and P. Varman. Workload decomposition for QoS in hosted storage services. In 3rd Workshop on Middleware for Service Oriented Computing (MW4SoC), 2008.

[14] L. Lu, K. Doshi, and P. Varman. Graduated QoS by decomposing bursts: Don't let the tail wag your server. In 29th IEEE International Conference on Distributed Computing Systems, (ICDCS), June 2009.

[15] L. Lu, K. Doshi, and P. Varman. Decomposing workload bursts for efficient storage resource management. IEEE Transactions on Parallel and Distributed Systems, 22(5), 2011, pp. 860–873.

[16] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through i/o off-loading. In 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2008.

[17] K. I. Park. QoS in packet networks. Springer, USA, 2005.

[18] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quality of service guarantees via service curves. In Proceedings of the International Conference on Computer Communications and Networks, pages 512–520, 1995.

[19] J. Turner. New directions in communications (or which way to the information age?). Communications Magazine, IEEE 24 (10), pp. 8–15.

[20] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[21] H. Wang, and P. Varman, Nested QoS: Providing flexible QoS in shared IO environments, Usenix 3rd Workshop on I/O Virtualization, (WIOV'11), June, 2011.

[22] H. Wang and P. Varman, Flexible resource sharing in virtualized environments, ACM International Conference on Computing Frontiers, (CF'11), May, 2011.

Author Biographies

Hui Wang is a graduate student at Rice University. Her research interests are in QoS scheduling, storage, and distributed and operating systems. She received her bachelor's degree from Shandong University and master's degree in computer science from Rice University.

Kshitij Doshi is a principal engineer in the Software and Services Group at Intel Corporation. He has a Bachelor of Technology degree in electrical engineering from Indian Institute of Technology (Mumbai), and a master's degree and PhD in computer engineering from Rice University. His research interests span operating systems, optimization of performance, power, and energy in enterprise solutions, database architectures, and virtual machines. He can be contacted at kshitij.a.doshi@intel.com

Peter Varman is a professor in the Departments of Electrical and Computer Engineering and Computer Science at Rice University. From 2002 through 2005 he was Program Director for computer systems architecture at the National Science Foundation in Washington DC. During 2011-2012 he was a scholar in residence at VMware in Palo Alto, where he worked on issues relating to resource management for virtualization and cloud computing. He has also held short-term visiting positions at IBM T.J. Watson and IBM Almaden Research Labs, Duke University, and NTU, Singapore. His research interests span the areas of virtualization and res ource management, cloud computing, computer architecture, storage systems, and applied algorithms. He earned a Bachelor's of Technology degree in electrical engineering from IIT, Kanpur and a PhD from the University of Texas at Austin.

SELF-ORGANIZING SYSTEM-ON-CHIP DESIGN

Contributors

Rafael de la Guardia
Intel Labs Guadalajara

Carlos Gershenson
Universidad Nacional Autónoma de México

"Organic principles are in the solution path for important challenges in logic organization, data movement, circuits, and software."

Self-organization in the context of computing systems refers to a technological approach to deal with the increasing complexity associated with the deployment, maintenance, and evolution of such systems. The terms self-organizing and autonomous are often used interchangeably in relation to systems that use organic principles (self-configuration, self-healing, and so on) in their design and operation. In the specific case of system on chip (SoC) design, organic principles are clearly in the solution path for some of the most important challenges in areas like logic organization, data movement, circuits, and software[47]. In this article, we start by providing a definition of the concept of self-organization as it applies to SoCs, explaining what it means and how it may be applied. We then provide a survey of the various recent papers, journal articles, and books on the subject and close by pointing out possible future directions, challenges and opportunities for self-organizing SoCs.

Introduction

Autonomic computing has been a popular research topic, especially since the publication of the influential paper by Kephart and Chess[19] outlining IBM's vision on how to deal with the increasing complexity associated with the deployment, maintenance, and evolution of enterprise systems. In a broader context, autonomic computing describes the application of advanced technology to the management of advanced technology[10]. Dobson et al.[10] include organic computing, bio-inspired computing, self-organizing systems, ultra-stable computing, autonomous and adaptive systems, to name a few, under the term autonomic.

In the specific case of SoC, organic principles (self-configuration, self-healing, and so on) have been proposed by various authors to deal with the enormous challenges of designing and actually delivering reliable, high performance and ultra low power systems as process variations, transient faults (soft errors), thermal effects, and aging become harder to manage with advanced process technology. Sander et al.[35] point to a recent shift in manufacturing technology from zero defects to a design for yield approach, accepting functional imperfection will happen. Variations in SoC occur at various temporal and spatial scales. Bull, et al.[3] classify various types of variations ranging from static, local inter-die process variations to fast, dynamic variations that develop in a few cycles or less, such as PLL jitter and capacitive coupling effects. Fault-tolerant circuits, buses, and caches are used to cope with these variations. Terms like resiliency, redundancy, adaptivity, approximate arithmetic, error detection, and error correction appear often in connection with these techniques. Constantinescu, et al.[6] warn about new error sources related to the increased

complexity of the fault-tolerance mechanisms themselves, which manifest in particular when circuits operate in "corner cases," which are difficult to validate.

In the next section, the concept of self-organization is introduced, noting when it is useful for designing systems. The section "Proposals" presents a review of the literature, divided into four subsections, dealing with reliability, survivability, power/performance optimization, and temperature control. Future directions and open challenges are outlined in the section "Future Directions." This is followed by a section where we draw some conclusions. Table 1 shows a glossary of terms used throughout the paper.

Acronym	Description
TDDB	Time-dependent dielectric breakdown
NBTI	Negative bias temperature instability
HCI	Hot carrier injection
RAMP	An architecture-level model to track microprocessor lifetime reliability
V_{dd}, V_{th} and F_{max}	Supply voltage, threshold voltage and frequency of an electronic component, respectively
IPC	Instructions per cycle
EDP	Energy-delay product
MTTF	Mean time to failure
DTM	Dynamic thermal management
DVFS	Dynamic voltage and frequency scaling
BIST	Built-in self test
CMP	Chip multiprocessor
FPGA	Field-programmable gate array

Table 1: Glossary
(Source: Intel Corporation, 2011)

Self-Organization

Self-organization is a property evident in several biological systems, such as insect colonies, flocks of birds, and schools of fish. It can be characterized as a global pattern (organization) emerging from local interactions (self).

"Self-organization is a property evident in several biological systems, such as insect colonies, flocks of birds, and schools of fish."

For engineering purposes, self-organization can be used as a guiding principle to design and control systems[13]. Components are designed in such a way that they will find solutions to problems as they interact. This is useful for "non-stationary" problems, where the requirements are dynamic and thus the predictability is limited. As elements interact, they self-organize adaptively to novel circumstances, ideally matching the scale(s) at which problems change.

"For engineering purposes, self-organization can be used as a guiding principle to design and control systems."

Adaptation can be seen as a useful change in a system as a response to perturbations. Living systems are constantly adapting to changes in their environment, so they have been a source of inspiration for engineering adaptive systems. Self-organization is one method that can be used to build adaptive systems.

Proposals

Figure 1 shows a hierarchical organization of the survey. The four sections on reliability, resiliency, power/performance optimization and temperature control correspond to the main objectives of most of the self-organizing proposals found in the literature. Some of the specific techniques, like task scheduling or voltage/frequency scaling, were used to achieve more than one of the above mentioned objectives.

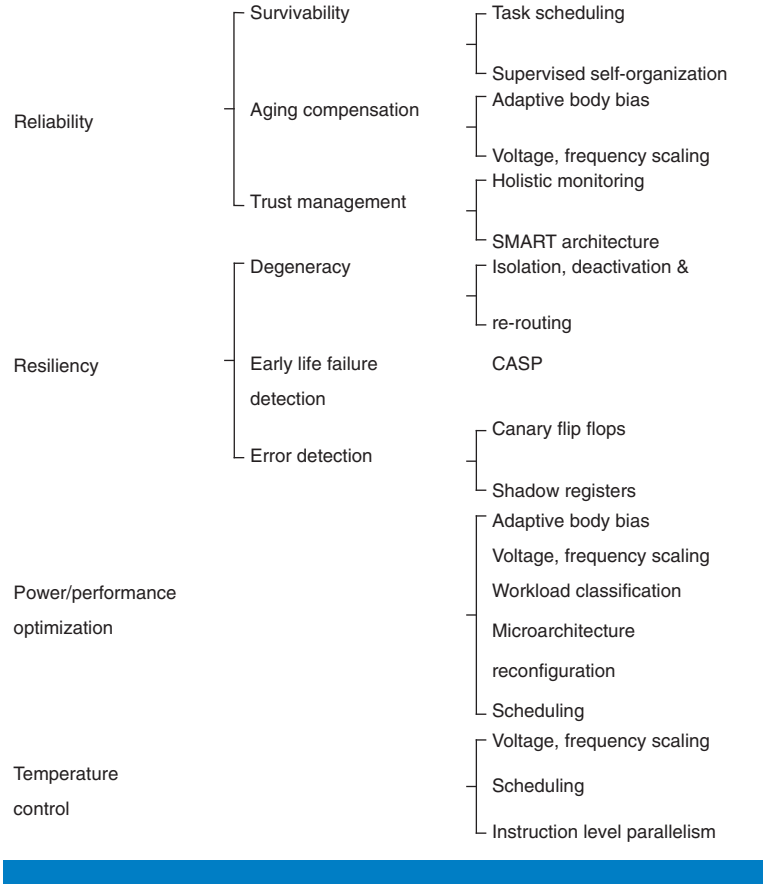


Figure 1: A hierarchical organization of the survey (Source: Intel Corporation, 2011)

Reliability

"Reliability targets are satisfied by adapting dynamically to usage."

Srinivasan, et al.[39] proposed lifetime reliability awareness at the microarchitectural level to qualify processors instead of the traditional approach that uses a worst-case scenario. Reliability targets are satisfied by adapting dynamically to usage. They contributed an architectural level model (RAMP) that tracks reliability and a dynamic management technique that works in parallel with DTM. RAMP includes models of all the critical failure mechanisms (electromigration, stress migration, TDDB, and thermal cycling) to compute a processor's MTTF as a function of temperature and utilization. They divided the processor in various structures and applied RAMP to each (floating-point unit, register files, branch predictor, caches, load-store queue, and so on). A complete

simulation tool would use the RAMP models plus a timing simulator (workload behavior) and power/thermal simulators (power/thermal profiles).

Survivability

Feng, et al.[12] proposed a reliability-centric scheduling system called Maestro, which assigns threads to cores in a CMP system based on estimated damage extent within the cores and thermal footprints of the running applications, shown in Figure 2. They reported significant peak temperature variations (10 percent to 40 percent) between processors and large variations (10 percent to 20 percent peak deltas) across modules inside the processors when running various fixed point and floating point SPEC2000 workloads (http://www.spec.org). These variations are expected to have a drastic impact on mean time to failure. The scheduler requires circuit level sensors for health monitoring that explicitly exploit statistically measurable degradation in timing paths at the microarchitectural level. The authors focused on two failure mechanisms, namely, NBTI and TDDB. They examined three scheduling policies: 1) a greedy policy that preserves even the weakest core; 2) an adaptive policy that promotes survival of the fittest by maximizing lifetime reliability of the CMP; and 3) a naïve round-robin policy as baseline. A 38 percent improvement in CMP lifetime and up to 180 percent improvement in lifetime throughput was observed in Monte Carlo simulations of a 16-core CMP using either the adaptive or greedy policies.

"Circuit level sensors for health monitoring explicitly exploit statistically measurable degradation in timing paths at the microarchitectural level."

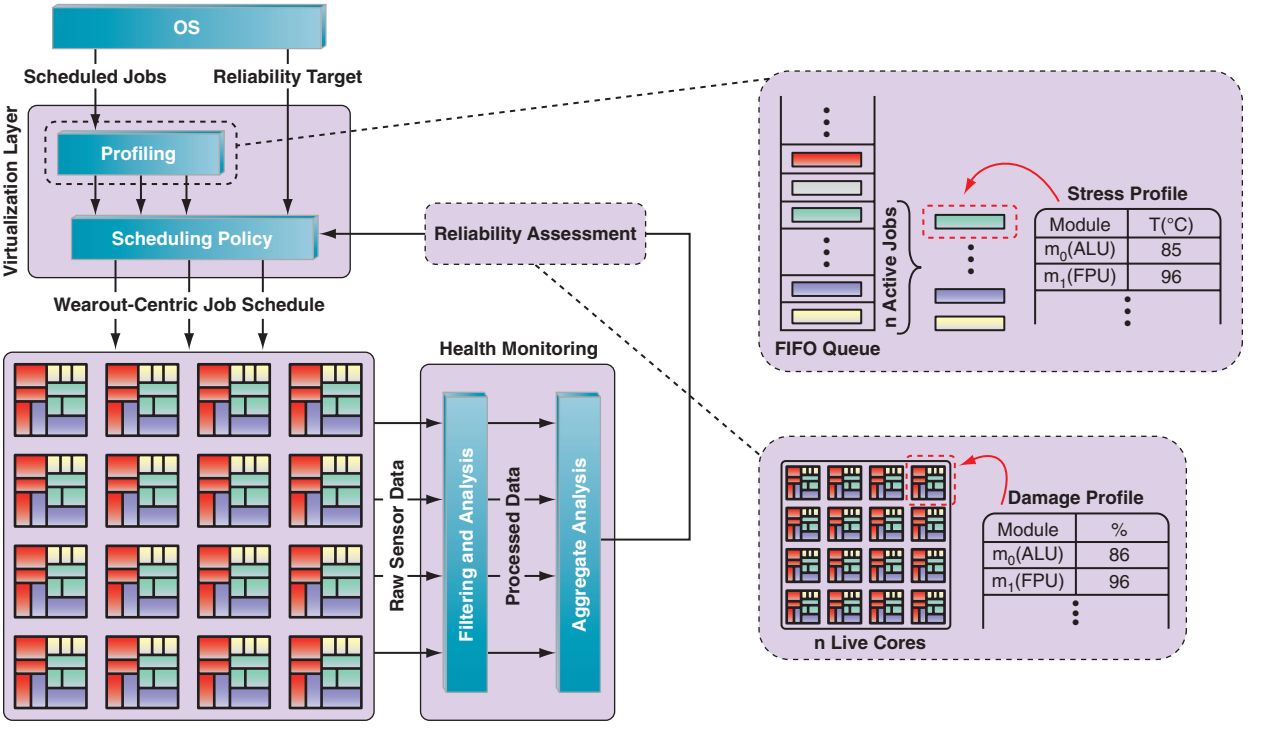


Figure 2: Block diagram of the Maestro introspective reliability management system (Source: HiPEAC 2010[48])

Aging Compensation

Sander, Bernauer, and Rosentiel[35] described a framework for self-adaptation in SoC towards electronic system level reliability. They considered the effect on reliability due to the characteristics of real applications, the environment and user behavior, on one hand, and the correct modeling of physical effects, on the other.

"Adaptive body bias (BB) compensates the effect that bias temperature instability has on circuit performance over its lifetime."

Kumar, et al.[23] used adaptive body bias (BB) to compensate the effect that bias temperature instability has on circuit performance over its lifetime. Previously ABB had only been used for leakage/performance tradeoffs. Their adaptive technique was based on a lookup table to map V_{dd} and BB values to the cumulative time of NBTI stress (aging) on the circuit. BB can be used to speed up a circuit as it ages by decreasing its V_{th} and thereby using the available leakage slack, at a cost of a substantial power overhead, particularly towards the end of life of the device when higher BB values are needed (to recover speed). Hence adaptive BB is used in combination with adaptive supply voltage to minimize the total power overhead (active power plus leakage) such that the delay at any time is less than or equal to a delay specification for the device.

"A system reliability manager senses the impact of power delivery, temperature, and the workload on the hardware platform."

Khan and Kundu[21] proposed a framework to manage transistor aging, where the chip not only tests itself but also adapts to the changing conditions, providing higher performance for all applications during the initial years and graceful performance degradation as the device ages. The system level architecture is based on virtualization of a system reliability manager that senses the impact of power delivery, temperature, and the workload on the hardware platform, and subsequently responds by adapting the supply voltage and/or operating frequency, as shown in Figure 3. Accurate estimates of

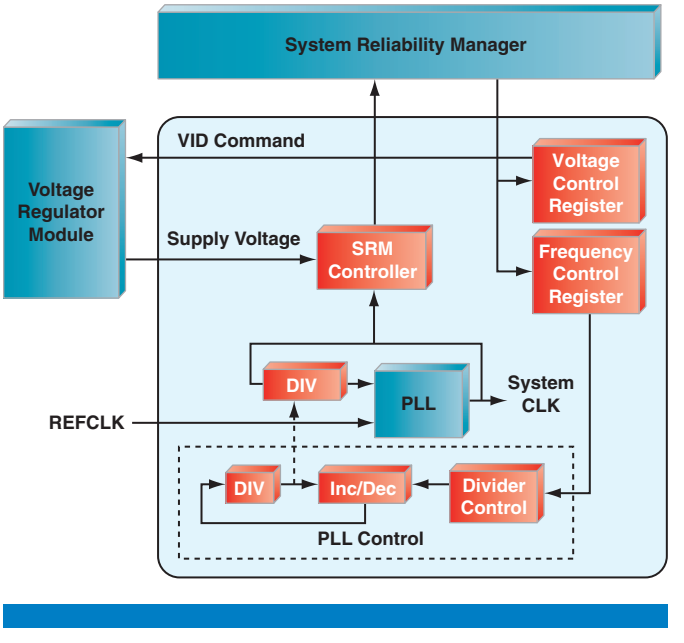


Figure 3: System reliability manager architecture (Source: DATE09[49])

F_{max} and V_{th} are essential, hence the software periodically runs system BIST and stress tests to update these values as the device ages. Simulations performed by the authors showed that worst-case MTTF rate of change is related to the workloads' thermal and performance behavior. They proposed using workload information to determine when the reliability manager should be invoked to reconfigure the system.

Li, et al.[24] described a framework for diagnostics called CASP (concurrent autonomous chip diagnostics using stored test patterns). Among the key features of CASP claimed by the authors are that it's designed for minimal performance and power impact so it can provide high test coverage with zero downtime (it can diagnose one or more cores as the others continue to operate). CASP was designed to take advantage of existing design-for-test functionality and for minimal design flow impact (on-line test controller, small off-chip buffer, architecture supplement to isolate core under test). The authors proposed using CASP to implement self-healing transistor aging, using common tune parameters (supply voltage and clock frequency) to compensate for aging and prevent delay- and fault-induced errors due to aging, maximize computational power efficiency (total number of clock cycles over lifetime divided by energy) and/or maximize system lifetime. For example, increasing supply voltage results in decreasing delay, increasing leakage current, and increasing aging. On the other hand, decreasing frequency results in decreasing bit error rate, decreasing power, and decreasing performance.

"Increasing supply voltage results in decreasing delay, increasing leakage current, and increasing aging."

"Decreasing frequency results in decreasing bit error rate, decreasing power, and decreasing performance."

Yi, et al.[43] proposed an aging test strategy including delay measurement and adaptive test scheduler. They considered path delay increase as a function of time caused by NBTI, HCI, TDDB, electro- and stress-migration, and delay measurements as a function of voltage and temperature.

Trust Management

Pionteck and Brockmann[30] presented a methodological framework for trust management, consisting of architecture and specific methods. According to the authors, uncertainty (untrustworthiness) affecting SoC dynamic thermal and reliability management is caused by: 1) unpredictability of workloads, for example due to load changes and local loops for error compensation, 2) sensors, 3) actors, 4) thermal dissipation, 5) the environment, 6) physical models used for tuning thermal management systems, and 7) by changes in chip properties (aging). Their proposed SMART architecture complements the traditional functional units with so called robustness units (RUs), which can be local, regional, or global, interconnected through a robustness network, as shown in Figure 4. Readings from sensors in an RU come supplemented with a trust level that varies between zero and one, representing the reliability of the sensors. Similarly, their proposed framework attaches to each actuator in an RU a trust level that is applied when predicting its influence on the hardware. Global RUs address global goals, like survivability and triggering actions to fulfill outer power and performance goals. The authors proposed fuzzy control techniques for trust level processing and robust learning classifier systems extended with the integration of dynamically changing trust levels.

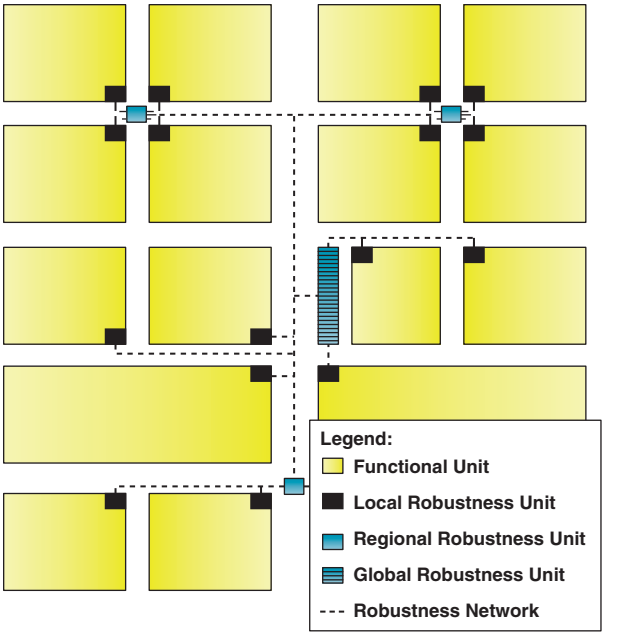


Figure 4: Schematic chip structure (Source: DSN-W 2010[50])

Kramer, et al.[22] proposed a cooperative, coordinated, system-wide monitoring infrastructure for self-organizing, massively parallel and heterogeneous systems, shown in Figure 5. Their proposed infrastructure is divided into low level monitoring (LLM) using associative counter arrays (ACA) to track any number of concurrent events, and high level monitoring (HLM) for data analysis based on an adaptive, rule-based approach. ACA provide cache-inspired uniform event coding to characterize the state of the system or application behavior. Their proposed organic processing cells include one LLM instance directly interfaced with an ACA in each component (such as CPU, cache, and so on). At a higher level, an HLM network consumes data from various LLM sources, using event lists for state evaluation and classification. Evaluation rules for state classification in the HLM nodes are derived at runtime and can be updated or new rules added at any time.

Gherman, et al.[14] devised delay control structures and architectural support for concurrent self-test of in-order pipelines based on the opportunistic re-execution of operations during naturally occurring stall cycles.

Resiliency

"Resiliency can be understood as the ability of systems of withstanding perturbations."

Resiliency can be understood as the ability of systems of withstanding perturbations. In other words, a resilient system will continue to function in spite of changes to the system.

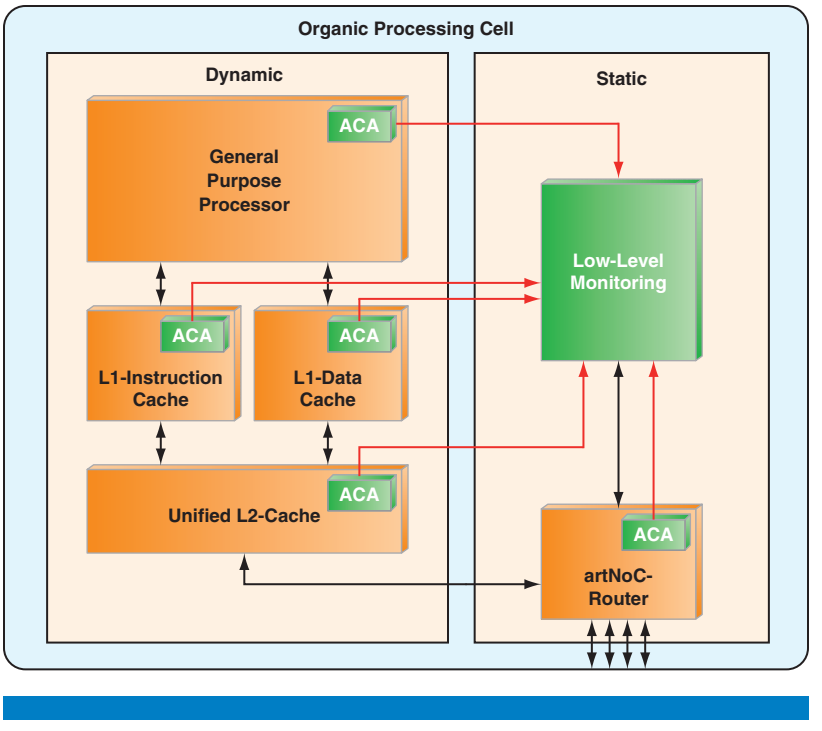


Figure 5: Hierarchical monitoring
(Source: Organic Computing—A Paradigm Shift for Complex Systems, 2011[51])

Degeneracy

Degeneracy is the ability of performing the same function by different components of a system.

"Degeneracy is the ability of performing the same function by different components of a system."

Polack[31] proposed adopting self-organizing techniques that previously had been successfully used in robot swarm organisms to achieve survivability in computer architectures. Survivability means that the system continues to provide some functionality in the presence of faults. This is possible because at least some of the robots (or the elements of the system) that are structurally different are able to perform the same function via degeneracy. Hence, critical elements adopt a survival mode under certain conditions. The system determines what functions can continue based on known relationships between health and tasks. Furthermore, immune-inspired fault tolerance systems in robotic swarm organisms are lightweight, making them attractive for use in self-organizing computer architectures.

Collet, et al.[5] described a self-organization approach for multicore chips in massively defective technologies designed to detect and tolerate both permanent faults (which escaped self-diagnosis) and transient faults on line. They assumed a system with possibly hundreds of uniform cores, caches, and routers where not only permanent defects but latent defects (time-dependent device degradation and material wear out; system failure at any time in the field) are significant. Self-organization makes sense in this case

"The self-organizing approach included self-test, autonomous isolation, auto-discovery of valid routes, and deactivation of defective modules."

because increased complexity results in reduced (external) controllability and observability. They considered a 2D array topology (symmetrical) with cores, routers, and caches, focused on architectural fault-tolerance, given the fault-tolerance hierarchy of circuits, architecture, scheduling, allocation, and execution. The self-organizing approach included self-test, autonomous isolation, auto-discovery of valid routes, and deactivation of defective modules. The self-diagnosis sequence goes from interconnects to routers to cores. Their main contribution was that they separated the self-testing of interconnects and routers from the software-based self-test of cores. They used diagnosis based on BIST using a test data generator, test error detector, and a maximal aggressor fault model to detect crosstalk defects.

Gupta, et al.[15] created StageNet, which is a reconfigurable and adaptable substrate that replaces direct connections at pipeline stage boundary by crossbars, thereby enabling creation of logical cores (called StageNetSlices) by grouping pipelines in different ways. The proposed infrastructure can be used, for example, to isolate failures by routing around defects.

Early Life Failure Detection

Li, et al.[25] proposed concurrent autonomous chip self-test using stored test patterns as an efficient on-line self-test strategy for uncore (logic blocks including cache, DRAM and I/O controllers) that can be used both for aging and early life failure detection (see also the earlier section "Aging Compensation"). According to the authors, their method exhibits a high test coverage (with extensive patterns in off-chip nonvolatile storage), minimal system level impact (<1 percent area and power, <3 percent performance) with a frequent self-test (1 second every 10 second). As opposed to a self-test targeting aging where self-test can be invoked every day, targeting early life failures (infant mortality) requires much more frequent testing, which poses a performance impact challenge.

"Targeting early life failures (infant mortality) requires much more frequent testing, which poses a performance impact challenge."

Error Detection

Hashimoto[16] constructed an adaptive speed control with timing error prediction using canary flip flops (FFs), shown in Figure 6. When timing margin decreases, timing errors occur in canary FFs before the main FFs capture a wrong value, thanks to a buffer delay. A warning signal is then generated that speeds up the affected circuit; conversely the circuit is slowed down if no warnings occur. There is a tradeoff between the optimum number of canary FFs and area overhead. The author used a Markov chain model, taking into account the probability of path activation to determine timer error rate and power dissipation. The concept was tested in a silicon prototype where a timing warning signal was connected to a speed control unit which selected the body bias voltage applied to a Kogger-Stone adder circuit. Insertion location and time delay of a canary FF was configurable as it was determined that the critical path is not always the best location for optimum tradeoff between power and area. A 46 percent power savings using adaptive speed control was observed compared to traditional guard-banded design for worst case.

Zeppenfeld, et al.[45] proposed an autonomic SoC architecture using a fault-tolerant CPU data path, a learning classifier table (see also the section "Power

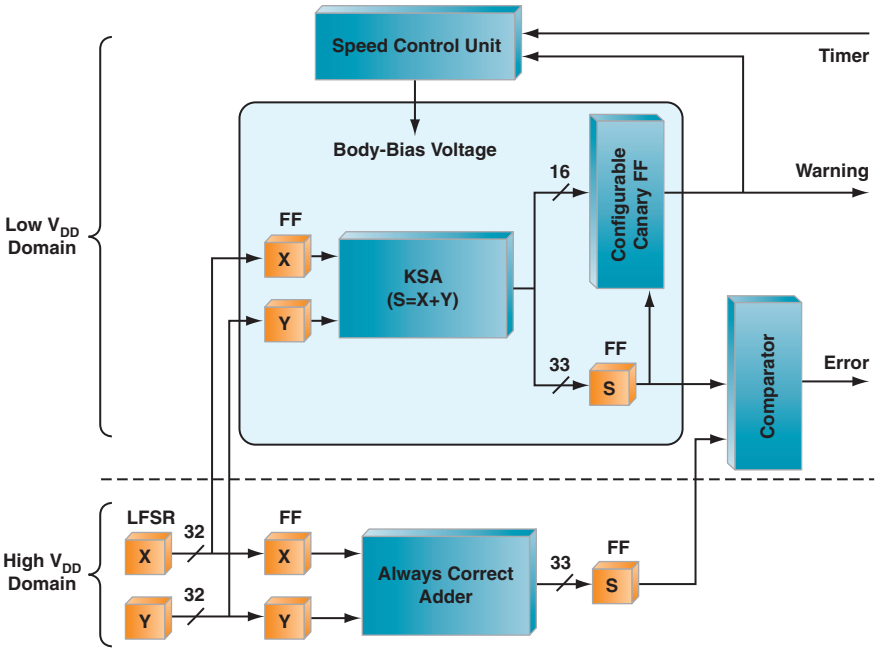


Figure 6: 32-bit Kogge-Stone adder controlled adaptively with configurable canary FF (Source: ASP-DAC 2011[52])

and Performance Optimization"), and an autonomic element interconnect structure, shown in Figure 7. The authors analyzed many types of error detection (based on hardware redundancy, self-checking arithmetic circuits and time redundancy) in terms of footprint, timing, and transient error detection capabilities and the possibility of intellectual property reuse. They selected a shadow register technique with error correction that doesn't require a complete pipeline flush but involves a constant pipeline stall penalty of two cycles.

"A shadow register technique with error correction that doesn't require a complete pipeline flush but involves a constant pipeline stall."

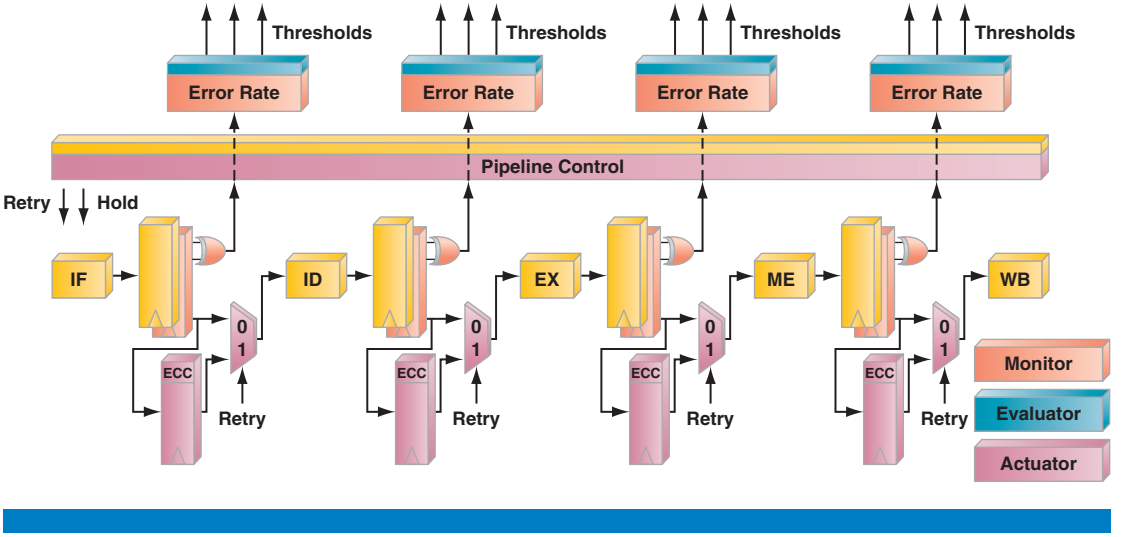


Figure 7: CPU data path with shadow registers for error detection and correction (Source: ISORCW 2010[53])

Minimum cycle overhead was considered critical to prevent the spreading of effects of an errant operation to other parts of the autonomic SoC.

Power and Performance Optimization

Puschini, et al.[32] considered the following global multiple objective optimization problem: manage the power and temperature for hot spot reduction and performance control through task synchronization for a given application with functional dependencies in an MP-SoC with multiple processing elements, each of which includes processor, memory, and peripherals. Their proposed solution is to use DVFS for each tile computed at system level using game theory. Their design used processing elements connected via asynchronous network-on-chip with the required bandwidth and latency. They found that functional dependencies between tasks at the application level lead to processing element frequency adjustment to guarantee task synchronization. Hence, the tradeoff between temperature and task synchronization had to be considered when mapping applications to the MP-SoC.

"Functional dependencies between tasks at the application level lead to processing element frequency adjustment to guarantee task synchronization."

"Proposed a simplified form of learning classifier system with a dynamically changing reward prediction, called a learning classifier table (LCT)."

Zeppenfeld, et al.[45] proposed a simplified form of learning classifier system with a dynamically changing reward prediction, called a learning classifier table (LCT), which features a fast rollback actuation path from monitors to actuators. An aggregator on top of an autonomic element monitor generates a condensed monitor signal (necessary to cope with the rate of data capture) that is passed on to the learning classifier, which then generates the output that is fed to the actuator back in the functional element. The LCT consists of a list of condition, action fitness values corresponding to the rules of the learning system. The rule to apply is determined by weighted roulette wheel selection without explicit match set creation using a reservoir sorting algorithm. The goal of the methodology was to provide a balance between exploitation of learning and exploration. Hence, rule weighting in their proposed scheme is based solely on the predicted reward, which corresponds to the fitness of each rule. The use of genetic operators to modify the rules (as used in full-fledged extended classifier systems) was proposed for a future investigation. Synthesis of the LCT in a Xilinx Virtex-II Pro FPGA revealed less than 5 percent overhead compared to the standard Leon 3 processor core. To test the LCT they implemented a simulation in software (not in the FPGA) of a networking application that transferred packets from main memory to Ethernet MAC for transmission. Initially, all tasks were scheduled in one core (of the Leon 3 cores). The objective function to be minimized using the LCT consisted of a weighted sum of deltas in frequency, utilization, and workload for all the cores. They used a token ring structure to furnish the autonomic system's interconnect needs. Sharing global data between autonomic elements was shown to be advantageous in their simulations. Going forward, the authors identified challenges in scalability, reliability, and LCT timing. For example, actuator actions take time to percolate through the system at various rates. So, when many autonomic elements are simultaneously generating actions, hierarchical interconnect structures simplify the learning process at the cost of possibly converging to local minima. This suggested a hierarchical

topology with islands of autonomic elements using local token ring structures connected to form archipelagos and so on. The authors noted that constraints are necessary to ensure system reliability and fault tolerance. These constraints include guards on directly adjustable parameters (such as voltage and frequency) and hard and soft constraints on indirectly adjustable parameters (such as utilization and temperature). The authors also distinguished between application-aware constraints (such as packet forwarding rate and video frame processing rate) and application-independent constraints. To cope with constraints, LCT proposes relative actions, preventing large jumps in parameters. Margins are still necessary between soft and hard constraints. Finally, the authors noted that the application of overlapping rules helps to reduce lookup latency, but its effect on learning behavior and system stability needs to be further studied.

Yoon, et al.[44] described an adaptive granularity memory system targeting improvements in throughput and power efficiency. The proposed system combines compiler, OS, cache hierarchy, and memory controller (for example, sub-ranking) tools to optimize the channel according to the granularity (locality) of the data. The authors reported gains in throughput and power efficiency of 44 percent and 46 percent, respectively, with respect to standard double data rate memory systems optimized for coarse grained access.

Yao, et al.[42] described an adaptive depth pipeline technique called pipeline stage unification (PSU) that according to the authors, provides lower power under light loads, is fast switching (order of nanoseconds) and uses a low cost method to categorize program behavior at high granularity. Based on processor IPC to categorize loads, the method achieved 13.5 percent EDP reduction compared to the same processor without PSU. For comparison, the authors noted that voltage changes in DVFS incur execution delays of tens of microseconds. PSU bypasses inactive parts of pipeline registers and use shallow pipelines under light loads (up to 5 times shallower compared to baseline with 20 stages). PSU latency can be broken down into a pipeline flush and a frequency scaling, which together translate into tens of cycles. It was observed that optimal depth (in terms of energy and performance) varies per application; hence, PSU uses an adaptive depth with three stage unification degrees. Based on the analyses presented by the authors, CPI (inverse of IPC) is a linear function of pipeline stages ($CPI = CPI0 + beta \times n$), with beta dependent on the application. In a similar way, power consumption can also be expressed as a function of pipeline stages. The analysis conducted by the authors showed that for large beta, shallower pipelines lead to improved EDP. Since CPI0 in their formulation is approximately constant across applications, relative beta for an application can be approximated as CPI/n, or (1/IPC)/n, which was the method used in their PSU implementation to categorize applications.

"PSU bypasses inactive parts of pipeline registers and use shallow pipelines under light loads."

"Optimal depth (in terms of energy and performance) varies per application; hence, PSU uses an adaptive depth."

Ma, Wang, and Wang[26] proposed including L2 cache in partitioning a CMP chip level power budget based on workload characteristics, using a power capping algorithm. In their proposed scheme, performance contributed by each heterogeneous component needs to be measured dynamically, so that

DVFS (per core) or cache resizing (which changes the number of active cache ways/lines/blocks) can be applied to enforce per component budget. According to the authors, CMP performance improves as function of power budget increase, for either core or cache, approximately as a linear or piecewise linear function. Hence, CMP performance in the short term can be approximated as the sum of the performance contributions from each component. They proposed using a recursive least squares estimator to calculate a performance contribution parameter vector for all the components. The resulting linear programming optimization problem to be solved was to maximize the sum of the performance contributions subject to the power budget of the whole chip. Power consumption could then be controlled using a proportional-integral-derivative controller.

Temperature Control

Huang, et al.[17] argued in favor of considering temperature aware architectures all the way from the early design stages to end of life of the devices. They pointed out that, compared to power aware architectures, temperature depends on power density, is a nonlinear function of time (versus instantaneous), is most important when utilization is high, affects reliability (power swings due to power management), and generates hot spots. Temperature models are needed to account for effects such as leakage, thermal dissipation, and microarchitecture versus transistor level impact. Therefore, the choice of cooling solution affects the optimum processor architecture. The authors noted some of the challenges ahead. In particular, with respect to DTM, DVFS, and (core) hopping, there are well-established techniques. However, variability and accuracy of measurements are key issues that may negate the benefit of these techniques. The sizing of guard bands is another key issue, as it must account for many different failure mechanisms, such as timing errors, soft errors (thermal noise), excessive leakage, aging, and so on. Better modeling across multiple spatial and temporal scales is essential, according to the authors.

"Temperature depends on power density, is a nonlinear function of time (versus instantaneous), is most important when utilization is high."

Coskun, et al.[7] investigated thermally-aware scheduling in MP-SoC. They proposed an OS-level solution that includes temperature measurements to decrease hot spots and temperature variations at reduced performance cost. The authors found that spatial temperature variations across the chip result in performance mismatches that lead to performance or logic failures. Timing failures result, for example, from NBTI and HCI. Circuit delays and voltage drop result from increasing circuit resistance associated with increasing temperature. The authors reported that raising temperature by 20°C resulted in a 5–6 percent increase in Elmore delay in interconnects, causing clock skew problems.

"Spatial temperature variations across the chip result in performance mismatches that lead to performance or logic failures."

Mesa-Martinez, et al.[28] characterized various SPEC200x (http://www.spec.org) applications based on several thermal metrics: 1) timing (maximum temperature and thermal gradient may cause throttling and skew/timing violations, respectively), 2) reliability (using RAMP to compute MTTF and average failures in time per block (architectural unit) weighted by area and added over all units), 3) energy (leakage based on the BSIM3 model described

in the 1998 M98/51 technical report that can be found in http://www.eecs.berkeley.edu/Pubs/TechRpts), 4) thermal time constant and 5) instructions per cycle (average and maximum).

Coskun, et al.[8] proposed a simulation framework that captures architectural level effects over tens of seconds or longer while also capturing thermal interactions among cores from scheduling policies. They found that different DTM techniques that provide nearly identical performance, power and peak temperature can differ by 2X in expected lifetime. Their model includes electromigration, TDDB, and thermal cycling. Lifetime based on this model was affected mostly by accounting (or not) for asymmetric thermal characteristics of cores (such as core location) and frequency of migration.

Reda, Cochran, and Nowroz[34] proposed a hard sensor allocation algorithm to determine the sensor locations where hot spots can be tracked accurately given a budget number of sensors. They further proposed a soft sensing technique that combined measurements from hard sensors in an optimal way to estimate temperature at any desired location.

Future Directions

Some general trends may be extracted from the various proposals reviewed in the previous section. Reliability and resiliency fall clearly in the domain of self-organizing systems. Self-test and self-healing are often cited as underlying control mechanisms. Survivability in the presence of variability, aging, and perturbations while maximizing lifetime throughput may be considered a main objective. Circuits and microarchitectural elements play a key role. Challenges have been identified in integrating circuit level and microarchitectural solutions for error detection and correction, fault isolation, and task scheduling under severe area, power, and performance constraints. Evolving models for aging and trust are critical to many of the proposed solutions.

"Survivability in the presence of variability, aging, and perturbations while maximizing lifetime throughput may be considered a main objective."

Adaptive techniques are more commonly used for power, performance, and temperature control. However, the methods used are often the same ones (for example, DVFS, task scheduling) that are used for managing reliability and resiliency. In general, there is no silver bullet and tradeoffs must be made between reliability, power, performance, and temperature. The fact that there is not a single, static definition of performance complicates matters further. At a system level, we find increasingly heterogeneous solutions and strong interactions between the hardware, the operating system, and the running applications. Indeed, SoCs may be considered vertical systems. Constraints are necessary as the response times associated with typical control mechanisms (such as DVFS) are relatively slow, on the order of tens of microseconds. Scalability, reliability, and learning present important challenges.

"There is no silver bullet and tradeoffs must be made between reliability, power, performance, and temperature."

With respect to temperature control, the choice of cooling solution has a large impact on power (for example, leakage), performance, reliability, and ergonomics. Maximum temperature (hot spot) and temperature gradients are

both important and difficult to measure in systems. Temperature sensors are generally not coincident with hot spot locations. Thermal models are therefore needed but time-dependent component variability and sensor/actuator accuracy must be accounted for. Guard bands are required to compensate for uncertainty in measurements and controls.

"There are several open challenges in the use of self-organization to cope with the increasing complexity of SoC."

There are several open challenges in the use of self-organization to cope with the increasing complexity of SoC. Current approaches aim at increasing the adaptability, robustness, and survivability of systems. However, this comes at a certain cost, in the sense that additional components are required to deploy a self-organizing SoC.

For example, a naive attempt to increase robustness is with redundancy of components. Thus, if one component fails, duplicate copies can maintain the functionality. However, multiplication of entire circuits or chips also multiplies their cost, instead of exploiting multiple resources in parallel. More economical measures to increase robustness are being developed.

Many approaches are inspired in living systems, since these exhibit the desired properties of engineered systems[40].

A list of some of the open challenges follows:

1. A formalization of the effect of different approaches in desired properties of adaptive and self-organizing SoC.
2. Standardized performance measures for comparing different proposals.
3. Minimization of additional modules while increasing robustness.
4. Adaptation to different and changing physical properties of SoC from fabrication differences, from changes in temperature, and from aging.
5. General methodologies for designing self-organizing SoC, to be compared using 2.

"Self organizing SoC can contribute to the increase of reliability, performance, survivability, and robustness, while offering a reduction of power consumption."

Not only challenges can be envisaged, but also several opportunities. Self-organizing SoC can contribute to the increase of reliability, performance, survivability, and robustness, while offering a reduction of power consumption, errors, and design time.

Conclusions

This article presented an overview of the literature related to self-organizing SoC. This literature was categorized in terms of reliability, resilience, power/performance optimization, and temperature control. Each of these areas is relevant for producing self-organizing SoC, and should be considered in parallel.

Building on the state of the art, a list of open challenges and several opportunities were mentioned. Future research is self-organizing SoC is promising to develop hardware of increased capacities.

References

[1] Barbagallo, D., Di Nitto, E., Dubois, D.J., and Mirandola, R. (2010). A Bio-inspired Algorithm for Energy Optimization in a Self-organizing Data Center, D. Weyns et al. (Eds.): SOAR 2009, LNCS 6090, pp. 127–151, 2010.

[2] Borkar, S. and Chien, A.A. (2011). The Future of Microprocessors, Communications of the ACM, May 2011, vol. 54, no. 5, pp. 67–77.

[3] Bull, D., Das, S.,Shivashankar, K., Dasika, G.S., Flautner, K., and Blaauw, D. (2011). A Power-Efficient 32 bit ARM Processor Using Timing-Error Detection and Correction for Transient-Error Tolerance and Adaptation to PVT Variation, IEEE Journal of Solid-State Circuits, Vol. 46, No. 1, January 2011, pp. 18–31.

[4] Chen, M., Wang, X., and Li, X. (2011). Coordinating Processor and Main Memory for Efficient Server Power Control, ICS'11, May 31-June 4, 2011, Tuscon, Arizona, USA.

[5] Collet, J. H., Zajac, P., Psarakis, M., and Gizopoulos, D. (2011). Chip Self-Organization and Fault Tolerance in Massively Defective Multicore Arrays, IEEE Transactions On Dependable and Secure Computing, Vol. 8, NO. 2, March–April 2011.

[6] Constantinescu, C., Parulkar, I., Harper, R., and Michalak, S. (2008). Silent Data Corruption—Myth or Reality? International Conference on Dependable Systems & Networks: Anchorage, Alaska, June 24–27 2008, pp. 108–109.

[7] Coskun, A.K., Rosing, T.S., and Whisnant, K. (2007). Temperature Aware Task Scheduling in MPSoCs, 2007.

[8] Coskun, A.K., Strong, R., Tullsen, D.M., and Simunic Rosing, T. (2009). Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors, Sigmetrics/Performance '09, June 15–19, 2009, Seattle, WA, USA.

[9] de Kruijf, M., Nomura, S., and Sankaralingam, K. (2010). Relax: An Architectural Framework for Software Recovery of Hardware Faults, 37th International Symposium on Computer Architecture (ISCA '10).

[10] Dobson, S., Sterritt, R., Nixon, P., and Hinchey, M. (2010). Fueling the vision of autonomic computing, Computer, Jan. 2010, pp. 35–41.

[11] Dong, X. and Xie, Y. (2011). AdaMS: Adaptive MLC/SLC Phase-Change Memory Design for File Storage, IEEE, pp. 31–36.

[12] Feng, S., Gupta, S., Ansari, A., and Mahlke, S. (2010). Maestro: Orchestrating Lifetime Reliability in Chip Multiprocessors, Y.N. Patt et al. (Eds.): HiPEAC 2010, LNCS 5952, pp. 186–200.

[13] Gershenson, C. (2007). Design and Control of Self-organizing Systems. CopIt Arxives, Mexico. ISBN 978-0-9831172-3-0 http://tinyurl.com/DCSOS2007.

[14] Gherman, V., Massas, J., Evain, S., Chevobbe, S., and Bonhomme, Y. (2011). Error Prediction based on Concurrent Self-test and Reduced Slack Time, DATE11, 2011.

[15] Gupta, S., Feng, S., Ansari, A. and Mahlke, S. (2011). StageNet: A Reconfigurable Fabric for Constructing Dependable CMPs, IEEE Transactions on Computers, Vol. 60, No. 1, January 2011, pp. 5–19.

[16] Hashimoto, M. (2011). Run-Time Adaptive Performance Compensation using On-chip Sensors, 3C-3, pp. 285–290.

[17] Huang, W., Stany, M.R., Allen-Ware, M., Carter, J.B., Chengx, E., and Skadron, K. (2011). Temperature-Aware Architecture: Lessons and Opportunities, preprint.

[18] Hudec, J. (2011). An Efficient Technique for Processor Automatic Functional Test Generation based on Evolutionary Strategies, Proceedings of the ITI 2011 33rd Int. Conf. on Information Technology Interfaces, June 27–30, 2011, Cavtat, Croatia.

[19] Kephart, J.O. and Chess, D.M. (2003). The Vision of Autonomic Computing, Computer, Jan. 2003, pp. 41–50.

[20] Khalid, A., Abdul Haye M., Jahan Khan, M., and Shamail, S. (2009). Survey of Frameworks, Architectures and Techniques in Autonomic Computing, 2009 Fifth International Conference on Autonomic and Autonomous Systems, pp. 220–225.

[21] Khan, O. and Kundu, S. (2009). A Self-Adaptive System Architecture to Address Transistor Aging, DATE09, 2009.

[22] Kramer, D., Buchty, R., and Karl, W. (2011). Monitoring and Self-awareness for Heterogeneous, Adaptive Computing Systems. C, Müller-Schloer et al. (eds.), Organic Computing—A Paradigm Shift for Complex Systems, pp. 163-177.

[23] Kumar, S. V., Kim, C. H., and Sapatnekar, S. S. (2011). Adaptive Techniques for Overcoming Performance Degradation Due to Aging in CMOS Circuits, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 19, No. 4, April 2011, pp. 603–614.

[24] Li, Y., Kim, Y.M., Mintarno, E., Gardner, D., and Mitra, S. (2009). Overcoming early-life failure and aging challenges for robust system design. Accepted for publication in IEEE Design and Test of Computers.

[25] Li, Y., Mutlu, O., Gardner, D. S., and Mitra, S. (2010). Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips, 2010 28th IEEE VLSI Test Symposium, pp. 232–237.

[26] Ma, K., Wang, X., and Wang, Y. (2011). DPPC: Dynamic Power Partitioning and Capping in Chip Multiprocessors, 2011.

[27] Ma, K., Li, X., Chen, M., and Wang, X. (2011). Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications, ISCA'11, June 4–8, 2011, San Jose, California, USA.

[28] Mesa-Martínez, F.J., Ardestani, E.K., and Renau, J. (2010). Characterizing Processor Thermal Behavior, ASPLOS'10, March 13–17, 2010, Pittsburgh, Pennsylvania, USA.

[29] Peters, S.J., Norman, A., Shykind, D., and White, M.T. (2005). A methodology to determine how much electrical margin is enough to ship a product, DTTC 2005.

[30] Pionteck, T. and Brockmann, W. (2010). A Concept of a Trust Management Architecture to Increase the Robustness of Nano Age Devices, 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 142–147.

[31] Polack, F.A.C. (2010). Self-organisation for Survival in Complex Computer Architectures, D. Weyns et al. (Eds.): SOAR 2009, LNCS 6090, pp. 66–83, 2010.

[32] Puschini, D., Clermidy, F., Benoit, P., Sassatelli, G., and Torres, L. (2008). Temperature-Aware Distributed Run-Time Optimization on MP-SoC using Game Theory, IEEE Computer Society Annual Symposium on VLSI, pp. 375–380.

[33] Ravotto, D., Sánchez, E., and Sonza Reorda, M. (2010). A Hardware Accelerated Framework for the Generation of Design Validation Programs for SMT Processors, 2010 IEEE.

[34] Reda, S., Cochran, R.J. and Nowroz, A.N. (2011). Improved Thermal Tracking for Processors Using Hard and Soft Sensor Allocation Techniques, IEEE Transactions on Computers, Vol. 60, No. 6, June 2011.

[35] Sander, B., Bernauer, A., and Rosenstiel, W. (2010). Design and Run-time Reliability at the Electronic System Level, IPSJ Transactions on System LSI Design Methodology Vol. 3 1–21 (Aug. 2010), pp. 1–21.

[36] Schmeck, H., Müller-Schloer, C. Çakar, E., Mnif, M., and Richter, U. (2010). Adaptivity and selforganization in organic computing systems. ACM Trans. Autonom. Adapt. Syst. 5, 3, Article 10 (September 2010), 32 pages.

[37] Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M. and Richter, U. (2011). Adaptivity and Self-organisation in Organic Computing Systems, C. Müller-Schloer et al. (eds.), Organic Computing—A Paradigm Shift for Complex Systems, 5–37.

[38] Soto, J., Moreno, J. M., and Cabestany, J. (2011). Description of a Fault Tolerance System Implemented in a Hardware Architecture with Self-adaptive Capabilities, J. Cabestany, I. Rojas, and G. Joya (Eds.): IWANN 2011, Part II, LNCS 6692, pp. 557–564, 2011.

[39] Srinivasan, J. Adve, S.V., Bose, P. and Rivers, J.A. (2004). The Case for Lifetime Reliability-Aware Microprocessors, 31st International Symposium on Computer Architecture (ISCA-04), June 2004.

[40] Steuer R, Waldherr S, Sourjik V, Kollmann M (2011) Robust Signal Processing in Living Cells. PLoS Comput Biol 7(11): e1002218. doi:10.1371/journal.pcbi.1002218

[41] Wang, J. and Calhoun, B.H. (2007). Canary Replica Feedback for Near-DRV Standby VDD Scaling in a 90nm SRAM, IEEE 2007 Custom Intergrated Circuits Conference (CICC), pp. 29–32.

[42] Yao J., Miwa S., Shimada H., and Tomita, S. (2011). A fine-grained runtime power/performance optimization method for processors with adaptive pipeline depth, Journal of Computer Science and Technology 26(2): pp. 292–301. Mar. 2011.

[43] Yi, H., Yoneda, T., Inoue, M., Fujiwara, H., Sato, Y., and Kajihara, S. (2010). Aging Test Strategy and Adaptive Test Scheduling for SoC Failure Prediction.

[44] Yoon, D.H., Jeong, M.K. and Erez, M. (2011). Adaptive Granularity Memory Systems: A Tradeoff between Storage Efficiency and Throughput, Proceedings of ISCA'11 June 4–8, San Jose, CA, USA.

[45] Zeppenfeld, J., Bouajila, A., Herkersdorf, A., and Stechele, W. (2010). Towards Scalability and Reliability of Autonomic Systems on Chip, 2010 13th IEEE International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing Workshops, pp. 73–80.

[46] Zhang, Y., Li, H., and Li, X. (2010). Software-Based Self-Testing of Processors Using Expanded Instructions, 19th IEEE Asian Test Symposium, 2010.

[47] Shekhar Borkar and Andrew A. Chien, The Future of Microprocessors, Communications of the ACM, May 2011, vol. 54, no. 5, pp. 67–77.

[48] Feng, S., Gupta, S., Ansari, A. and Mahlke, S. (2010). Maestro: Orchestrating Lifetime Reliability in Chip Multiprocessors, Y.N. Patt et al. (Eds.): HiPEAC 2010, LNCS 5952, pp. 186–200.

[49] Khan, O. and Kundu, S. (2009). A Self-Adaptive System Architecture to Address Transistor Aging, DATE09, 2009.

[50] Pionteck, T. and Brockmann, W. (2010). A Concept of a Trust Management Architecture to Increase the Robustness of Nano Age Devices, 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 142–147.

[51] Kramer, D., Buchty, R. and Karl, W. (2011). Monitoring and Self-awareness for Heterogeneous, Adaptive Computing Systems. C, Müller-Schloer et al. (eds.), Organic Computing - A Paradigm Shift for Complex Systems, pp. 163–177.

[52] Hashimoto, M. (2011). Run-time adaptive performance compensation using on-chip sensors, *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific* , pp. 285–290, 25–28 Jan. 2011

[53] Johannes Zeppenfeld, Abdelmajid Bouajila, Andreas Herkersdorf, and Walter Stechele (2010). Towards Scalability and Reliability of Autonomic Systems on Chip. In *Proceedings of the 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops* (ISORCW '10).

Author Biographies

Rafael de la Guardia is a senior hardware engineer in Intel Labs. Rafael joined Intel in 2005, working in the area of adaptive systems. He received a PhD in engineering from the National Autonomous University of México (UNAM). His email is Rafael.de.la.guardia@intel.com.

Carlos Gershenson is a full-time researcher and head of the computer science department at the Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas of the Universidad Nacional Autónoma de México. He received a PhD *summa cum laude* from the Vrije Universiteit Brussel in 2007 and has been working on the application of self-organization to engineering. His Web page is http://turing.iimas.unam.mx/~cgg/ and he can be reached at cgg@unam.mx.

WORKLOAD CONSOLIDATION IN VIRTUALIZED COMPUTING SYSTEMS VIA HIERARCHICAL CONTROL

Contributors

Rui Wang
Drexel University

Nagarajan Kandasamy
Drexel University

We develop a hierarchical control framework for workload consolidation in virtualized environments. The hierarchy uses the concept of receding horizon control and comprises two levels: fully distributed controllers that independently optimize the CPU share provided to virtual machines (VMs) under their control, so that the system-wide CPU capacity is appropriately tuned to the incoming workload intensity; and a supervisory controller that reduces power consumption during periods of light workload by consolidating the workload on to fewer VMs and shutting down extra servers. We validate the framework on a heterogeneous cluster supporting three online services, showing that the system adapts quickly to dynamic workload changes and saves, on average, 20 percent in power-consumption costs over a three-hour period when compared to a system operating without dynamic control. The framework is quite scalable and accommodates the dynamic addition/removal of system components while maintaining overall system performance.

Introduction

A preliminary version of this article appeared as a paper in the *International Conference on Autonomic Computing (ICAC)* titled "A distributed control framework for performance management of virtualized computing environments."

Virtualization technology enables on-demand computing where resources such as CPU, memory, and disk space are allocated to applications as needed, based on the currently prevailing workload demand, rather than statically, based simply on the peak workload demand. By dynamically provisioning virtual machines (VMs) and turning servers on/off appropriately, data center operators can maintain the desired quality of service (QoS) while achieving higher server utilization and lower power consumption.

"Operators can maintain desired QoS while lowering power consumption by dynamically provisioning VMs and servers to match the incoming workload."

"Routine system management tasks can be formulated as optimal control problems and solved online."

A promising method of automating system management tasks is to formulate them as online control problems in terms of cost/performance metrics[1][2]. Most proposed control architectures, however, are centralized designs aimed at managing the performance of a standalone server or a small-scale system comprising a few servers. Significant challenges must still be addressed to achieve real-time control of a large-scale computing system with multiple interacting components. For an optimization scheme to be of practical value in such a distributed setting, it must tackle the "curse" of dimensionality: the number of available tuning options is quite large and the corresponding search space grows exponentially with each new variable, making centralized controller designs intractable. Fortunately, control theory provides techniques that can reduce the computational burden of managing large-scale systems. Concepts

from approximation theory can be used to make relevant approximations when constructing dynamical models to predict system behavior and when optimizing the control variables issued to the system[3]. Another method is to structure controllers in decentralized fashion wherein the overall problem is decomposed into a set of simpler sub-problems and solved cooperatively by multiple controllers[4].

"The workload consolidation problem is decomposed into simpler subproblems and solved by multiple, decentralized controllers."

We develop and validate a control architecture for workload consolidation in virtualized computing environments using concepts from hierarchical and distributed control. Considering a heterogeneous cluster hosting multiple enterprise applications on VMs and processing a time-varying workload, the problem of interest is: (1) to maintain the applications' response times under their respective thresholds (the QoS metric) by dynamically tuning the CPU shares provided to VMs so that the system-wide CPU capacity can handle the incoming workload intensity; and (2) to reduce power consumption by shutting down unneeded servers during light workload periods. This problem is decomposed via a control hierarchy as follows:

- Controllers, implemented locally within each server, dynamically solve their respective subproblems of optimizing CPU shares to VMs under their control in a cooperative fashion such that the cluster as a whole offers the processing capacity needed to handle the workload intensity. Local controllers (LCs) are developed as non-communicating agents wherein each controller infers the actions of others in the cluster without explicitly exchanging messages. These fully distributed LCs comprise the L0 level of the control hierarchy. Each LC uses receding horizon (RH) control, a form of predictive control where the idea is to solve an optimal control problem over a given prediction horizon and then continuously extend this horizon forward[8].
- Since LCs at the L0 level tune the CPU share of VMs to match the incoming workload, servers have spare processing capacity available during periods of light workload. A supervisory controller (SC), placed on top of the L0 level, uses this opportunity to increase server utilization and reduce energy consumption by consolidating the workload on to fewer VMs/servers and shutting down unneeded servers. The SC comprises the L1 level of the hierarchy. The control laws governing the SC are simplified to provide approximate solutions that the LCs can refine further; the SC predicts the future workload and decides only which servers to operate such that the cluster possesses enough processing capacity to satisfy this workload, leaving the LCs to fine-tune the CPU shares provided to individual VMs.

"Receding horizon control is used to solve the optimal control problem over a given prediction horizon."

We validate the control framework on a heterogeneous cluster hosting three benchmark applications: Trade6, RUBBoS, and RUBiS. Experimental results demonstrate that the cluster, when subject to our workload traces and managed using the proposed approach, saves on average 20 percent in power-consumption costs over a three-hour period when compared to a system operating without dynamic control. The framework also shows excellent scalability and allows for the dynamic addition/removal of servers during system operation while maintaining the overall QoS.

The article is organized as follows. The next section, "Preliminaries," introduces the testbed and the proposed control architecture. We develop the L0 and L1 levels in subsequent two sections, "The L0 Control Level" and "The L1 Control Level." Next, in the section "Experimental Results, the framework is validated, and we summarize our findings in the section "Conclusions."

Preliminaries

Figure 1 illustrates our system architecture, supporting three services, termed Gold, Silver, and Bronze. The Gold service is enabled by Trade6, a stock-trading benchmark. Trade6 resides in the IBM WebSphere* application server, which is hosted by VMs in the application tier. The Silver service is enabled by RUBBoS, a bulletin board application. The Bronze service is enabled by RUBiS, an auction site. The Silver and Bronze services each reside in the Apache application server hosted by VMs. We focus on resource provisioning within the application tier only, since this tier typically requires more CPUs than the database tier[5].

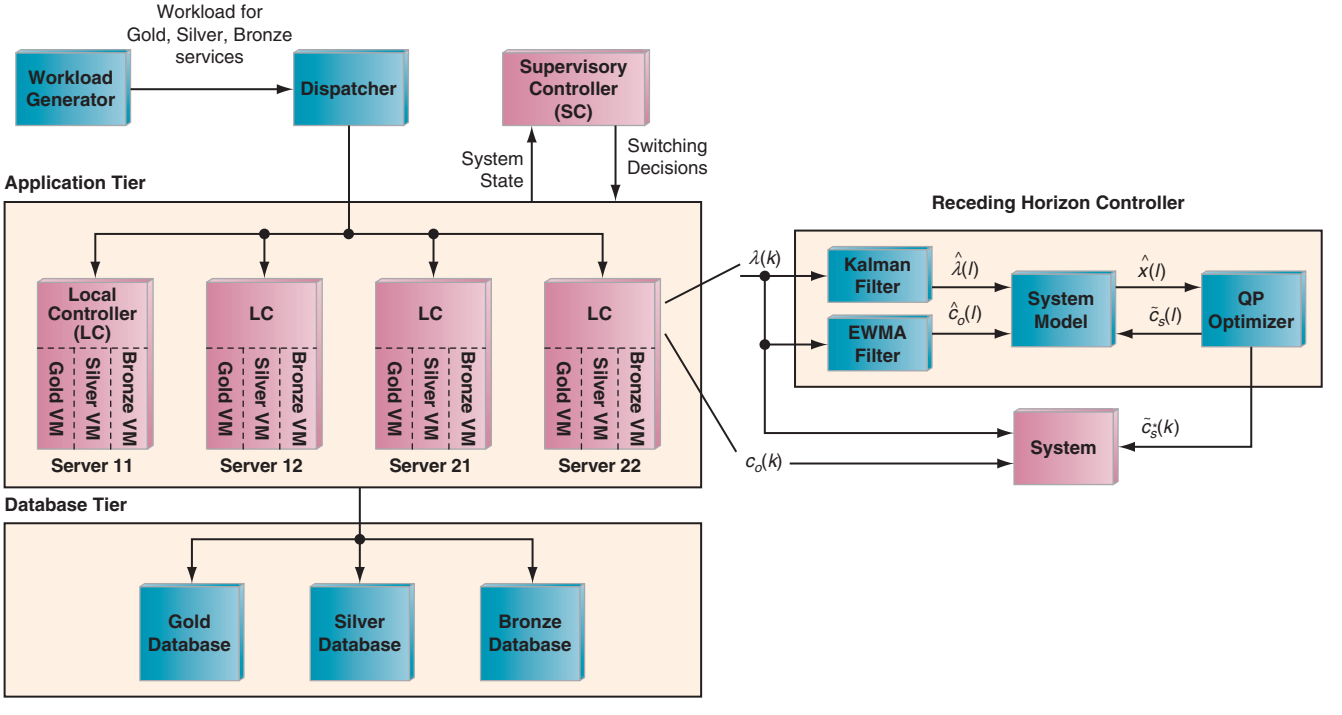


Figure 1: The system architecture and the schematic of the local controller implemented on each server (Source: Drexel University, 2012)

The application tier comprises four heterogeneous servers, each hosting three VMs. Virtualization is enabled by VMware's ESX Server* and each VM is dedicated to supporting one of the three services. VMs residing on different servers but supporting the same service form a *virtual computing cluster*. The local controller (LC) on each server allocates CPU share (in GHz) to its VMs via the ESX server API. Referring to Figure 1, Server11 and Server12 are Dell PowerEdge* 2950 machines with a total CPU capacity of 14 GHz; the per-VM CPU share ranges from 1.5 GHz to 5 GHz. Server21 and Server22 are PowerEdge 1950 machines

with a total CPU capacity of 11 GHz; the per-VM CPU ranges from 1.5 GHz to 4 GHz. Servers in the database tier run SUSE Linux with DB2 or MySQL as the database. Each database is dedicated to a single service.

A workload generator sends a mix of requests to the applications, requiring both database reads/writes. Incoming requests to an application are dispatched to VMs within the corresponding virtual computing cluster in a weighted round-robin fashion with the weights proportional to CPU share, since a VM's CPU share reflects processing capacity. At the start of each control step, an LC transmits its recent CPU-share decision to the dispatcher.

Figure 1 also shows our hierarchical control solution comprising two levels:

- *L0 level:* A fully distributed control structure where LCs on each server independently optimize the CPU share provided to VMs under their control to handle the incoming workload guarantee response-time requirements.
- *L1 level:* A supervisory controller (SC) that consolidates the workload and dictates which servers to turn on/off based on the system state and estimates of future workload intensity.

The SC and LCs cooperate to manage the cluster's power consumption while satisfying QoS requirements. Since LCs tune the CPU share of VMs based on the workload intensity, servers have spare CPU capacity available during periods of light workload. The SC uses this knowledge to shut down servers not needed and consolidate the workload on to fewer servers. The SC operates on a longer timescale than LCs. The proposed scheme is highly scalable. First, the number of servers in the application tier can be increased without affecting the control structure. Secondly, an upper-level controller with essentially the same logic as the SC can be added on top to manage multiple L1-level server clusters by switching them on/off. This can be eventually extended to multiple levels for larger systems.

The L0 Control Level

Design of the L0 level uses this well-known property: given multiple subsystems whose local cost functions are quadratic and whose dynamics and operating constraints are uncoupled, having each subsystem independently optimize its local cost function can potentially achieve the global optimal[6][7]. The performance management problem considered here falls in this category. To improve the scalability of our framework, the LCs are developed as non-communicating agents wherein the CPU capacity of other servers is inferred independently by each LC. Another issue is, if the LCs operate synchronously, they would observe the same external environment and system state, and take the same actions, causing the system to oscillate. Therefore, the LCs in our framework operate in an asynchronous fashion.

"Having each controller independently optimize its local cost function achieves the global optimal."

The scheme adopted by each LC, shown in Figure 1, follows receding horizon (RH) control. At time step k, the global request rate for all services, λ, and the aggregate CPU capacity of other VMs in the virtual clusters c_o, are estimated over the prediction horizon h, and used by the system model to forecast future

system states $\hat{x}$. The optimizer then finds an optimal sequence of control actions, $\{\tilde{c}_s(l) \mid l \in [k, k+h-1]\}$, representing CPU shares to VMs on this server in the next h steps. Then, only the first control action, $\tilde{c}_s^*(k)$, is applied to the system and the rest are discarded. The process is repeated at step $k+1$.

For a given application, the workload arrival rate λ for the coming time step k is estimated by a Kalman filter[9] as $\hat{\lambda}(k)$. For a VM being offered a CPU share c (in GHz), we define the request rate it can handle while satisfying the desired response time as $m \cdot c$, where m is a mapping factor. (The method used to obtain m is detailed in the section "Experimental Results.")

From an LC's perspective, the total CPU share offered by a virtual cluster for an application comprises two terms: the local VM's CPU share c_s, and c_o, the aggregate CPU share of other VMs in the virtual cluster. The LC needs to estimate c_o at step k without communicating with other LCs. Let $\lambda_s(k-1)$ and $\lambda_o(k-1)$ denote the request rate to the local VM and other VMs during the previous time step $k-1$, respectively, and let $c_s(k-1)$ and $c_o(k-1)$ be the corresponding CPU shares. Then, the following condition holds:

$$\begin{cases} \lambda(k-1) = \lambda_s(k-1) + \lambda_o(k-1), \\ \dfrac{\lambda_s(k-1)}{c_s(k-1)} = \dfrac{\lambda_o(k-1)}{c_o(k-1)}. \end{cases}$$

The LC is aware of $\lambda(k-1)$, $\lambda_s(k-1)$, and $c_s(k-1)$, and so $c_o(k-1)$ can be computed using the above condition. An EWMA filter then estimates $\hat{c}_o(k)$ in the coming step k as

$$\hat{c}_o(k) = \eta \cdot c_o(k-1) + (1-\eta) \cdot \hat{c}_o(k-1), \tag{1}$$

where η is a smoothing factor.

For the LC, the dynamics at time k of a virtual cluster supporting an application can be represented as

$$\hat{x}(k+1) = x(k) + T_s \cdot [\hat{\lambda}(k) - m \cdot \hat{c}_o(k) - m \cdot c_s(k)], \tag{2}$$

where T_s is the sampling period and $\hat{x}$ is the state of the application representing the accumulated error between $\hat{\lambda}$ and $m \cdot \hat{c}_o + m \cdot c_s$. So, the LC aims to drive $\hat{x}(k+1)$ to 0 by tuning c_s so that the response time is satisfied while minimizing the corresponding CPU share.

Based on equation 2, the LC constructs a model that includes all the applications hosted on the server as

$$\hat{\mathbf{x}}(k+1) = \mathbf{A} \cdot \mathbf{x}(k) + \mathbf{B} \cdot \mathbf{u}(k), \tag{3}$$

where

$$\mathbf{A} = \mathbf{I}, \quad \mathbf{B} = \mathbf{T}_s \cdot [\mathbf{I}, -\mathrm{diag}(m_i), -\mathrm{diag}(m_i)], \quad \mathbf{u}(k) = [\hat{\lambda}(k), \hat{\mathbf{c}}_\mathbf{o}(k), \mathbf{c}_\mathbf{s}(k)]^T,$$

the subscript $i = g, s, b$ denotes the Gold, Silver, and Bronze applications, respectively, m_i denotes the corresponding mapping factor, and T denotes transpose. To ensure

QoS while reducing the CPU share, the LC poses this problem as a quadratic programming (QP) problem, maintaining both vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ near their set points $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{u}}$, respectively, along the one-step prediction horizon:

$$\min \sum_{l=k}^{k+1} [(\mathbf{x}(l) - \bar{\mathbf{x}}(l))^T \mathbf{P}(\mathbf{x}(l) - \bar{\mathbf{x}}(l))] + (\mathbf{u}(k) - \bar{\mathbf{u}}(k))^T \mathbf{Q}(\mathbf{u}(k) - \bar{\mathbf{u}}(k)) \qquad (4)$$

where

$$\mathbf{P} = p \cdot \mathrm{diag}(p_i), \qquad \mathbf{Q} = q \cdot \mathbf{I}.$$

The variables p and q are weights reflecting the tradeoff between application performance and CPU share, and p_i reflects the relative priority of application i. The optimizer must also consider constraints on the upper and lower bounds on a VM's CPU share, and the fact that the sum of CPU shares offered to VMs within the host must not exceed the total CPU capacity. Solving equation 4, the LC obtains the optimal CPU share vector $\tilde{c}_s^*(k)$ and applies it to VMs in the server.

The L1 Control Level

Powering up servers, instantiating the VMs, and finally launching applications, incurs significant dead time, typically ranging from five to ten minutes. So, if the SC needs a usable server during step $k + 1$, it must power on that host in advance; that is, at the start of step k. Such actions require predictive control where the decision must be made in anticipation of future workload intensity. The SC uses two-step RH control and its sampling time is set as 15 minutes so as to make timely but not overly frequent switching decisions. Figure 2(a) shows the overall scheme comprising these steps:

"Powering up servers and instantiating VMs requires making control decisions in anticipation of future workload intensity."

- With $\hat{\lambda}(l)$ and the system model, a QP optimizer finds a sequence of control vectors, $\tilde{C}(l)$, representing the aggregate CPU share needed by the virtual clusters along the prediction horizon.

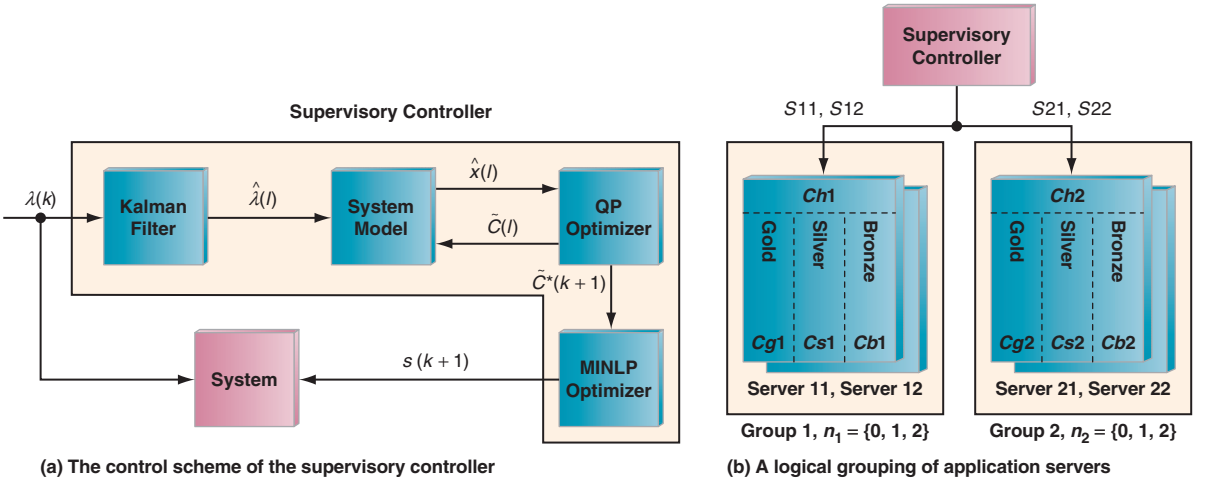


Figure 2: The control scheme of the supervisory controller. To improve scalability, servers are logically grouped based on their processing capabilities.
(Source: Drexel University, 2012)

- The second control vector in $\tilde{C}(l)$, denoted as $\tilde{C}^*(K+1)$, acts as a constraint to a mixed-integer nonlinear programming (MINLP) problem that determines the set of servers to power on/off.

The first step adopts the method detailed in the previous section. From the SC's viewpoint, the dynamics of a virtual computing cluster supporting one application at step k is

$$\hat{x}(k+1) = x(k) + T_s \cdot [\hat{\lambda}(k) - m \cdot C(k)], \tag{5}$$

where T_s is the sampling period of the SC, C is the aggregate CPU share of the whole virtual cluster, and $\hat{x}$ represents the accumulated error between $\hat{\lambda}$ and $m \cdot C$.

Thus the system model that includes all applications is

$$\hat{\mathbf{x}}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{6}$$

where

$$\mathbf{A} = \mathbf{I}, \quad \mathbf{B} = T_s \cdot [\mathbf{I}, -\mathrm{diag}(m_i)], \quad \mathbf{u}(k) = [\hat{\lambda}(k), \mathbf{C}(k)]^T .$$

The SC aims to guarantee the QoS and minimize the system wide CPU capacity. So it keeps $\boldsymbol{x}$ and $\boldsymbol{u}$ near the set points $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{u}}$, respectively, along the two-step prediction horizon:

$$\min \quad \sum_{l=k}^{k+2} [(\mathbf{x}(l) - \bar{\mathbf{x}}(l))^T \mathbf{P}(\mathbf{x}(l) - \bar{\mathbf{x}}(l))] + \sum_{l=k}^{k+1} [(\mathbf{u}(l) - \bar{\mathbf{u}}(l))^T \mathbf{Q}(\mathbf{u}(l) - \bar{\mathbf{u}}(l))] \tag{7}$$

The aggregate CPU share provided to service i should obey a lower and upper bound (as determined by the capacity of the cluster), and the total CPU share to all the services should not exceed the cluster's maximum capacity. Solving equation 7, the QP optimizer gets $\tilde{C}^*(K+1)$, the desired aggregate CPU share for the cluster at step $k+1$.

A MINLP optimizer then guarantees $\tilde{C}^*(K+1)$ by determining the optimal set of servers to power on/off. To improve scalability, we logically group the heterogeneous application servers based on their CPU capabilities as shown in Figure 2(b) and solve this problem in two steps. We first determine the number of servers to keep operational in each group by optimizing:

$$\min_{n} \quad \sum_{j} (c_{hj} \cdot n_j) \tag{8}$$

subject to

$$\begin{cases} n_j \in \{0,1,\ldots,N_j\} \\ c_{lj} \le c_{ij} \le c_{uj} \\ \sum_i c_{ij} \le c_{hj} \\ \sum_j (n_j \cdot c_{ij}) \ge \tilde{C}_i^*(\boldsymbol{k}+1) \end{cases} \tag{9}$$

Here, the subscript i denotes applications, and j = 1, 2 denotes Group 1 and Group 2, respectively. N_j is the maximum number of hosts in group j, while n_j is the number of hosts to keep operational, and c_{hj} is the maximum CPU capacity of each host. Since power consumption depends on operating frequency, $\sum j(c_{hj} \cdot n_j)$ approximates the power consumed by all operational hosts. A feasible CPU share to a VM supporting application i and residing in Group j is denoted

as c_{ij} while c_{lj} and c_{uj} are the lower and upper bounds. (Note that c_{ij} is simply a value assumed by the SC to ensure a feasible solution. The optimal CPU is determined by the LCs in a finer timescale.) The last constraint in equation 9 ensures that the aggregate CPU share offered by the operational VMs in each virtual cluster is no less than the desired $\tilde{C}^*(K+1)$. After n_j is obtained, the optimizer decides the status of individual hosts in each group by optimizing:

$$\min_{s} \quad \sum_{k} (s_{jk} - \bar{s}_{jk})^2 \tag{10}$$

subject to

$$\begin{cases} s_{jk} \in \{0,1\} \\ \sum_k s_{jk} = n_j \end{cases} \tag{11}$$

Here, $\bar{s}_{jk}$ and s_{jk} represent the current and next state of the kth host in Group j: s_{jk}= 0/1 denotes that the host is off/on and so $(s_{jk} - \bar{s}_{jk})^2$ captures the corresponding switching cost. After s_{jk} is obtained, the SC applies it to the hosts.

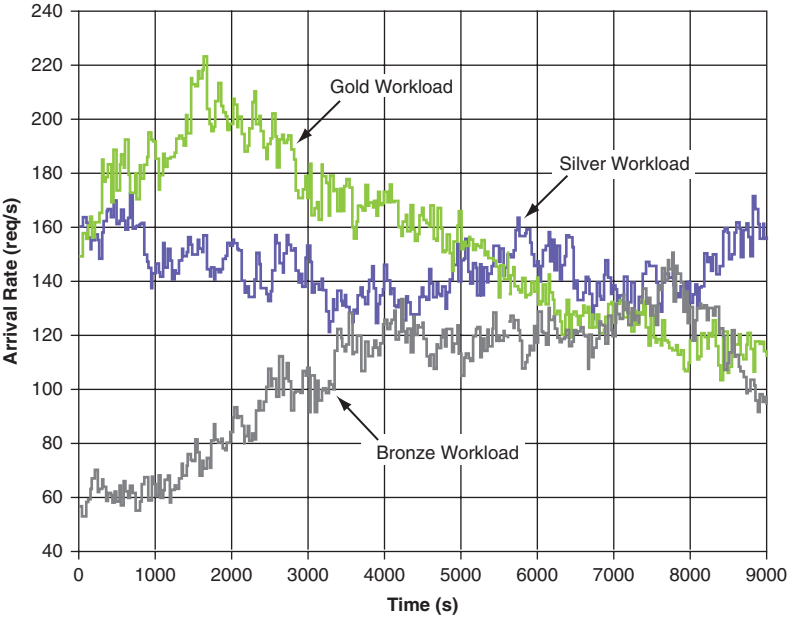
Experimental Results

The framework developed in the previous two sections has been validated on the testbed shown in Figure 1. The LCs in our experiments have a sampling period of two minutes and their starting times are each staggered by 30 seconds for asynchronous execution. The SC has a sampling period of 15 minutes and implements a policy to avoid frequent switching activity: once a server is turned on, it is held operational for at least four time steps (one hour); and if a server is turned off, it remains powered down for at least two time steps. In equation 1, η is set to 0.1 to focus more on the past observations. In equations 4 and 7, p_i are set to 5, 3, and 1, respectively, to prioritize the three services, and $p = 2$ and $q = 1$, giving greater priority to satisfying the response time over assigning lower CPU shares. Finally, both $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{u}}$ are set to 0.

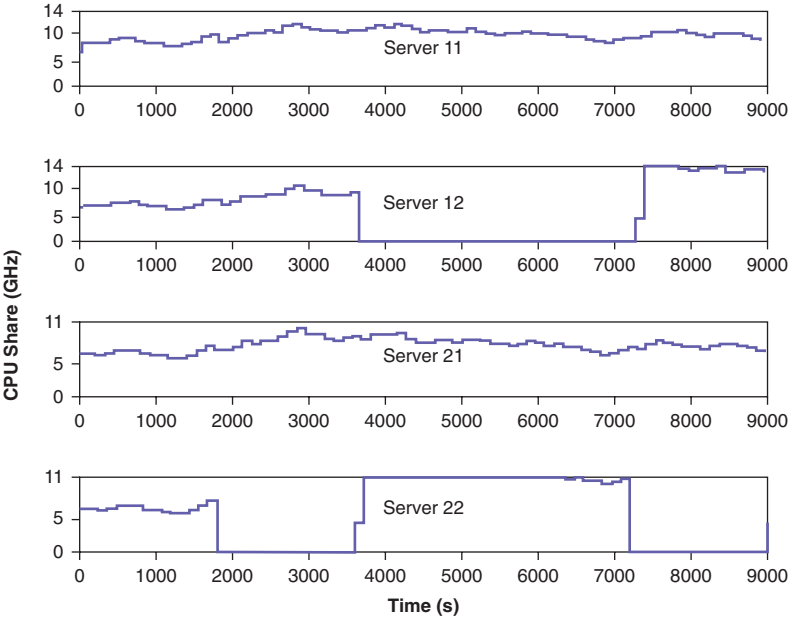
Profiling experiments were performed to obtain the mapping factor m_i used in equations 3 and 6. Consider the response times achieved by a VM hosting the Gold service as a function of CPU share and request arrival rate. If the VM is assigned a fixed CPU share of 3 GHz, then for an arrival rate under 65 req/s we achieve a relatively steady response time below 200 ms. Once the rate exceeds 65 req/s, the response time jumps dramatically, indicating an unstable system. So, a 3 GHz VM can accommodate approximately 65 req/s. This procedure is repeated with different CPU shares. Since a stable response time of below 200 ms is achievable in our experiments, it is set as the QoS for Gold service. The Silver and Bronze services show similar profiles and their QoS goals are set as 250 ms and 300 ms. By analyzing the request rates accommodated by a VM as a function of CPU share, the factor is obtained as 20.8, 14.5, and 12.8 for each of the three services, respectively.

We drive the testbed using the dynamic workload shown in Figure 3(a). The workload mix for the three services has a 50:50, 80:20, and 90:10 ratio of database reads to writes, respectively. (Experiments with other workload traces are qualitatively similar.) Figure 3(b) shows the switching behavior of the servers as dictated by the SC. About 1800 secs into the run, the SC estimates that

three servers are sufficient to process the incoming workload and powers down Server22. At 3600 secs, the supervisory further fine-tunes the cluster's CPU capacity to match the workload intensity by powering up Server22 (with a CPU capacity of 11 GHz) and turning off Server12 (with a slightly higher CPU capacity of 14 GHz).

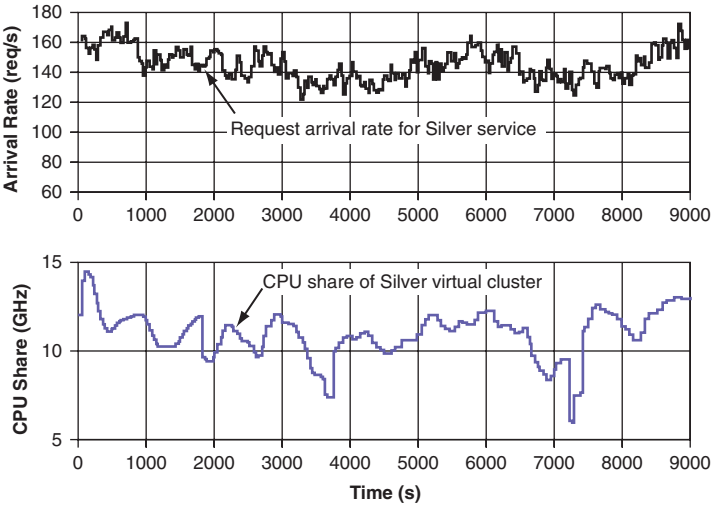


(a) Number of requests to each service, plotted at 30-second granularity.

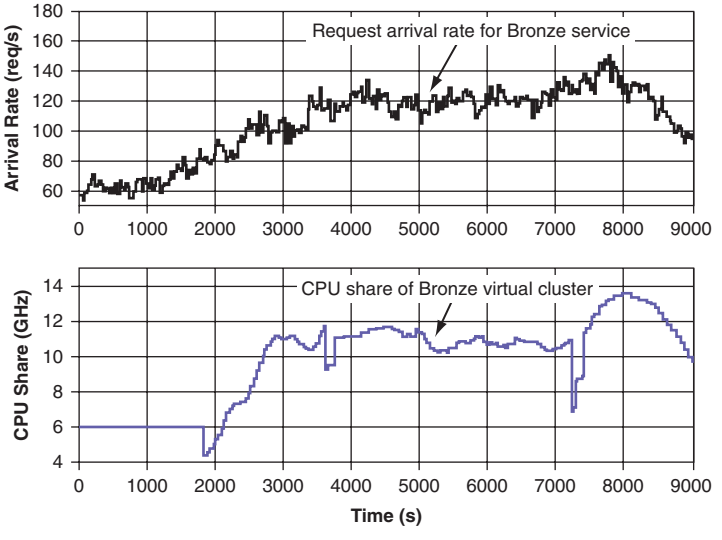


(b) Switching behavior of the cluster.

Figure 3: (a) The workload supplied to the cluster **(b)** switching behavior of the servers as commanded by the SC
(Source: Drexel University, 2012)



(a) CPU share provided to the Silver cluster.

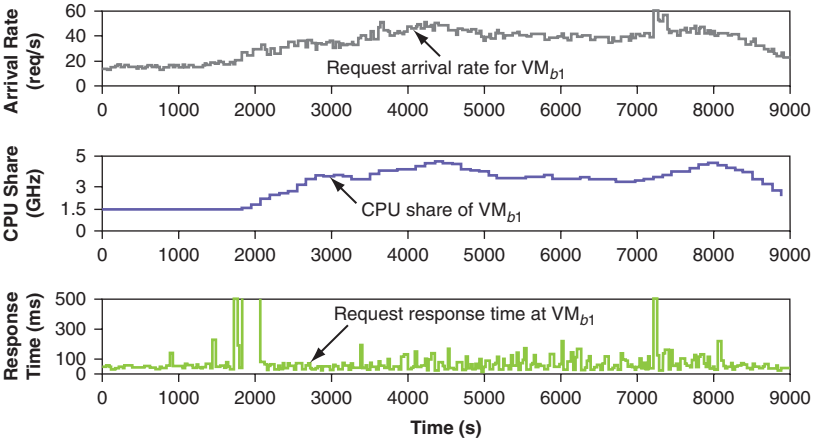


(b) CPU share provided to the Bronze cluster.

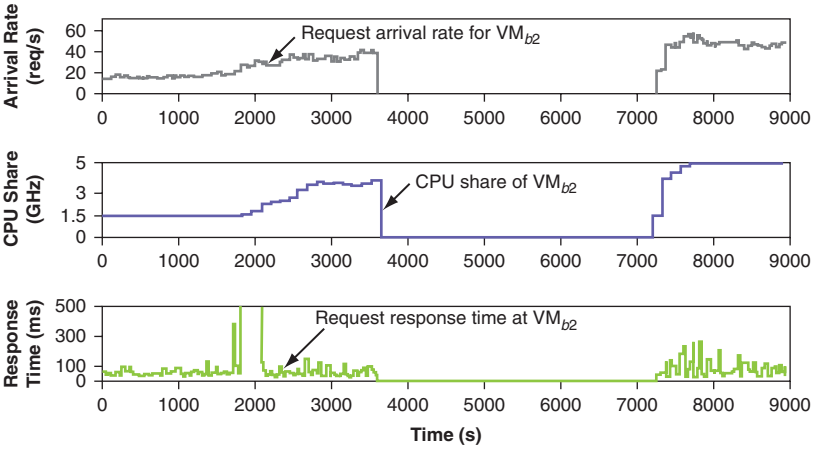
Figure 4: The aggregate CPU share provided by LCs to Silver and Bronze virtual clusters in response to the workload (Source: Drexel University, 2012)

Figure 4 shows the aggregate CPU share provided to the Silver and Bronze virtual clusters in response to the workload traces. (The Gold cluster behaves similarly.) The plots show that even when the workload intensity is highly variable and bursty, each virtual cluster's CPU capacity tracks this variability well. Note that the CPU capacity dips briefly when servers are turned off by the SC but then bounces back quickly (within two LC control steps) since other LCs appropriately tune the CPU shares of their VMs.

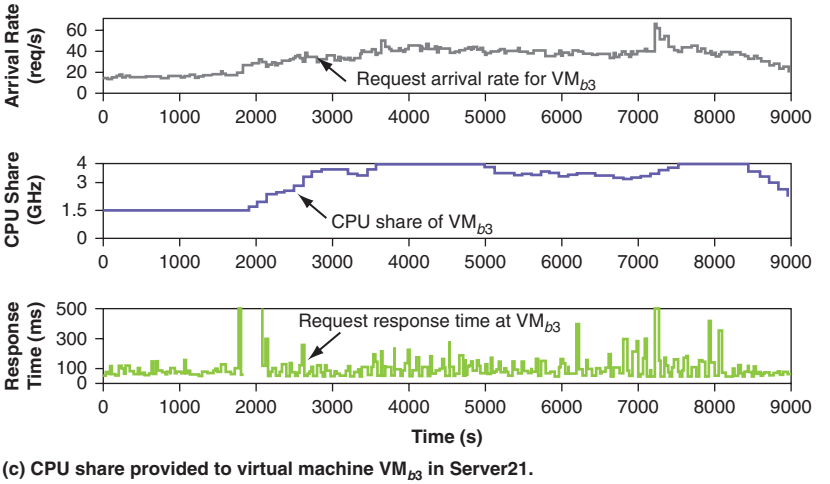
Next, we characterize the performance of individual VMs in the Bronze virtual cluster. We denote a VM supporting the Bronze service and residing in the jth server ($j \in (1, 2, 3, 4)$) as VM_{bj}, and the corresponding LC as LC_k. Referring to Figure 5, we note that though the Bronze workload trends upward, a total CPU



(a) CPU share provided to virtual machine VM_{b1} in Server11.

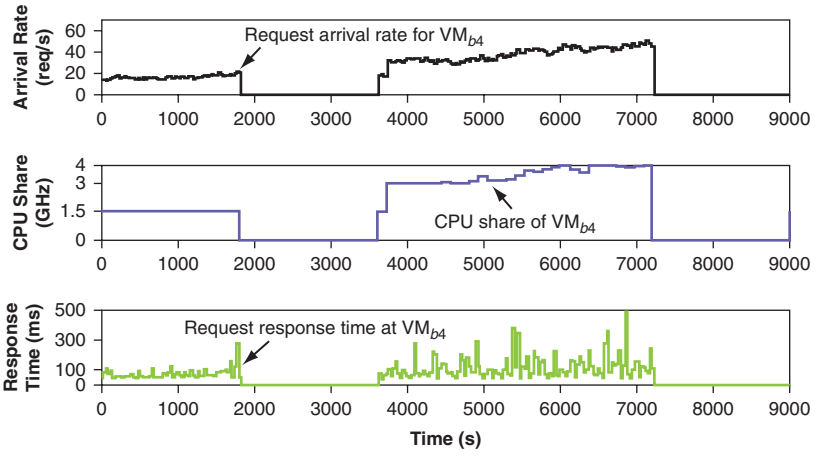


(b) CPU share provided to virtual machine VM_{b2} in Server12.



(c) CPU share provided to virtual machine VM_{b3} in Server21.

Figure 5: The performance of VMs in the Bronze virtual cluster (Source: Drexel University, 2012)



(d) CPU share provided to virtual machine VM$_{b4}$ in Server22.

Figure 5: *(continued)*

capacity of less than 6 GHz is sufficient until 1800 sec. So, each LC provides the VM with the minimum 1.5 GHz CPU. At 1800 sec, VM_{b4} leaves the system as Server22 is switched off by the SC causing a dip in the total CPU capacity, and VM_{b4}'s workload is distributed to other VMs resulting in transient spikes in response times. Meanwhile, LC_1 estimates that the arrival rate is trending upward and infers that the current CPU capacity of the Bronze cluster is insufficient. So, it increases VM_{b1}'s CPU share properly; LCs managing VM_{b2} and VM_{b3} behave similarly as well. VM switching activities also happen at 3600 and 7200 seconds, and are handled appropriately by the LCs as expected. During the entire experiment, the LCs cooperate very well and the response time is maintained mostly under 300 ms with very few QoS violations, as shown in Table 1.

Control type	Services	QoS violations	Power savings
Practical	Gold	23 (1.91%)	506 Watth (20%)
	Silver	15 (1.25%)	
	Bronze	41 (3.41%)	
Oracle	Gold	58 (4.82%)	759 Watth (30%)
	Silver	35 (2.91%)	
	Bronze	30 (2.49%)	

Table 1: Performance of practical and oracle controllers.
(Source: Drexel University, 2012)

Our final tests were aimed at comparing the practical controller implementation against an "oracle" with perfect knowledge of other controllers' actions and future workload arrivals at both the L1 and L0 levels. As Table 1 shows, the practical controllers maintain a very low percentage of QoS violations, and save, on average, 20 percent in power consumption at the application tier when compared to a system operating without dynamic control where servers are always powered on. (The power savings achieved, of course, also depend

"The control scheme saves 20 percent in power consumption costs while incurring few QoS violations."

on the specific workload traces.) When compared with the oracle, the practical ones achieve higher QoS satisfaction but less power savings. This is because the Kalman filters slightly overestimate the workload to cover the variability, making the LCs more generous when assigning CPU shares and the SC more conservative when turning off servers. The oracle does not use prediction and is more accurate when allocating CPU shares and switching servers, saving about 30 percent in power consumption.

"Both local and supervisory controllers have low computational cost and can be scaled up to manage large systems."

Our scheme is an attractive option for controlling large systems. Since each LC is only responsible for VMs residing in a single server, the corresponding execution time is only 0.01 seconds; for a sampling period of 120 seconds, the control overhead is 0.008 percent. Moreover, the execution time of the SC as a function of cluster size is small as well: an SC managing 80 servers incurs a control overhead of 1.949 percent for a sampling time of 900 seconds and only 6 percent when managing 100 servers.

Conclusions

We have developed a two-level framework to manage the performance/power of virtualized computing environments using concepts from hierarchical and distributed control. We have validated the control framework on a cluster of heterogeneous servers hosting multiple applications. Experimental results show that, the structure adapts quickly to dynamic workload changes, guarantees QoS most of the time, and saves, on average, 20 percent in power consumption costs. The framework is quite flexible in that it can scale up to multiple levels and tolerate the dynamic addition/removal of system components while maintaining the overall system performance.

References

[1] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[2] J. Hellerstein, S. Singhal, and Q. Wang, "Research challenges in control engineering of computing systems," IEEE Trans. Network & Service Mgmt., vol. 6, no. 4, pp. 206–211, Dec. 2009.

[3] K. Narendra and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," IEEE Trans. Neural Networks, vol. 8, no. 3, pp. 475–485, May 1997.

[4] N. Sandell, P. Varaiya, M. Athans, and M. Safonov, "Survey of decentralized control methods for large scale systems," IEEE Trans. Automatic Control, vol. 23, no. 2, pp. 108–128, April 1978.

[5] T. Atwood, "Right architecture for the right workload: The application tier," Sun Microsystems Report, Enterprise Systems Products, Tech. Rep., Jul. 2004.

[6] A. Guez, I. Rusnak, and I. B. Kana, "Multiple objectives optimization approach to adaptive and learning control," Int'l Journal Control, vol. 56, no. 2, pp. 469–482, September 1992.

[7] W. B. Dunbar and R. M. Murray, "Distributed receding horizon control for multi-vehicle formation stabilization," Automatica, vol. 42, no. 4, pp. 549–558, 2006.

[8] J. M. Maciejowski, *Predictive Control with Constraints.* London: Prentice Hall, 2002.

[9] S. G. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting: methods and applications,* 3rd ed., S. C. Wheelwright (Editor), Wiley, 1998.

Author Biographies

Rui Wang is a PhD candidate in the Department of Electrical and Computer Engineering at Drexel University. His research focuses on the application of control theory to the performance management of virtualized computing systems. He received his BS degree in Automation from the University of Science and Technology Beijing, China.

Nagarajan Kandasamy is an Associate Professor in the Electrical and Computer Engineering Department at Drexel University. He holds a PhD degree from the University of Michigan and his research interests include dependable computing, embedded systems, self-managing systems, and computer architecture. Dr. Kandasamy is a recipient of the NSF CAREER award in 2007 as well as best paper awards in the IEEE Conference on Autonomic Computing in 2006 and 2008. He is a senior member of the IEEE.