Configuration and Deployment Guide for the Cassandra NoSQL Data Store on Intel Architecture

About this Guide

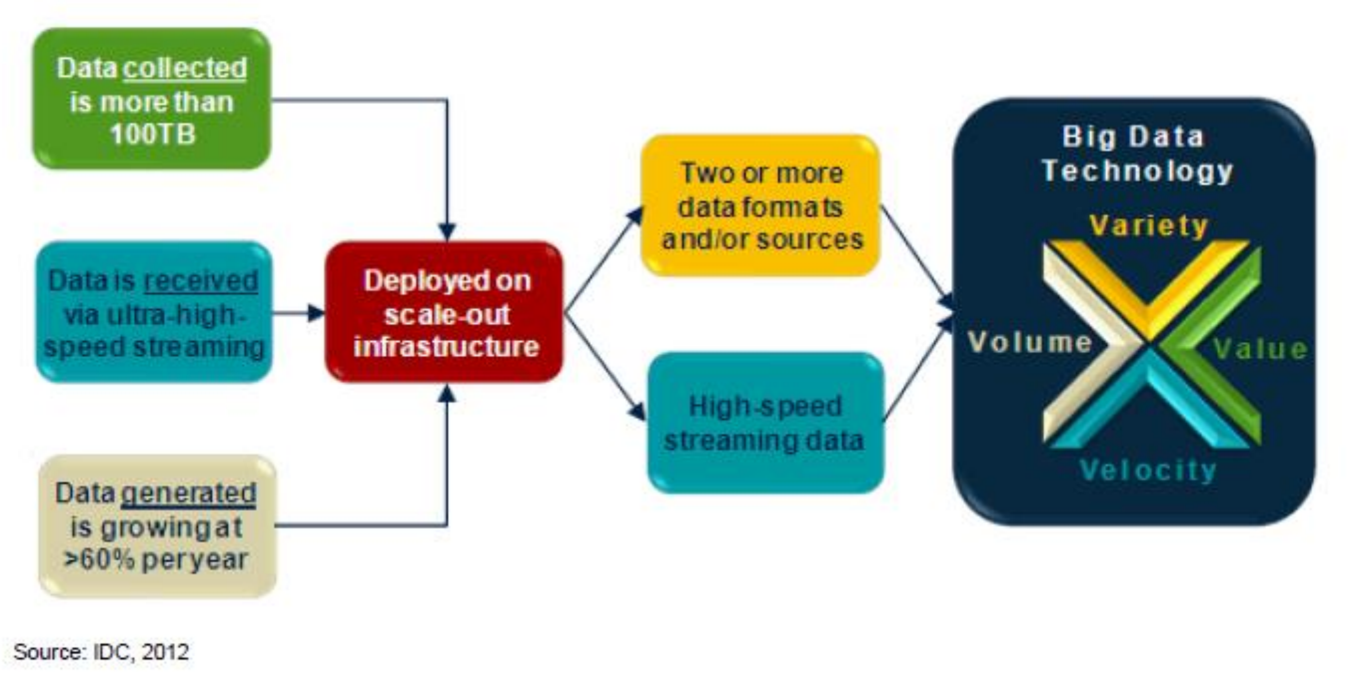
This *Configuration and Deployment Guide* explores one of the leading Not Only Structured Query Language (NoSQL) databases, Cassandra, on Intel® Architecture. The configuration guidelines address use cases with both Intel Xeon® processor- and Atom™ processor-based servers that take into account differing business scenarios, performance requirements and Total Cost of Ownership (TCO) objectives.

Table of Contents

1.0 Introduction – Driving Forces for NoSQL

The driving force behind the development of NoSQL databases is the need to rapidly store and manage ever larger, dynamically changing "Big Data" information sets. According to the Open Data Center Alliance (ODCA), Big Data can be defined as "massive amounts of data, the size and variety of which are beyond the processing capabilities of traditional data management tools to capture, manage and analyze in a timely manner". The key challenges include capture, curation, storage, search, sharing, transfer, analysis and visualization.

Big Data is often characterized by the three "V"s of Volume, Variety, and Velocity; along with a fourth "V" referring to the Value that can be derived from the effective use of Big Data informational assets.


Data collected is more than 100TB
Data is received via ultra-high-speed streaming
Data generated is growing at >60% per year
Deployed on scale-out infrastructure
Two or more data formats and/or sources
High-speed streaming data
Big Data Technology
Variety
Volume
Value
Velocity
Source: IDC, 2012

IDC's market forecast shows that the worldwide Big Data technology and services market will grow at a 31.7% compound annual growth rate (CAGR) – about seven times the rate of the overall information and communication technology market – with Big Data revenues reaching $23.8 billion in 2016.

NoSQL emerged as a means to satisfy new data format demands, in which data is no longer predominantly system generated, but rather user content emanating from multiple sources.

This new phenomenon presents challenges for existing database systems, as the user generated data does not abide by any structure or model. With the popularity of social media outlets such as Facebook and Twitter, not only is the data growing at unprecedented rates, but it also needs to be shared with greater frequency by a growing user base.

Big Data workloads vary from compute-intensive applications, such as real-time analytics, to I/O-intensive operations, such as responding to fast-trending social media peak usage patterns and so-called "hot-key" requests (e.g., a surge of requests for a "hot" video clip, picture or other piece of popular content).

Depending on your workloads and business requirements, the appropriate type of NoSQL implementation and the type of servers to run it on will vary. This guide examines configuration and deployment practices for Cassandra both in heavy-workload/rigorous SLA environments and in lighter-workload/looser-SLA environments.

2.0 NoSQL Database Overview

2.1 NoSQL vs. Relational Databases

NoSQL databases provide data storage and retrieval methods that use looser consistency models than traditional relational database management system (RDBMS) approaches. Primary reasons for the move to NoSQL include simplicity of design, improved horizontal scaling, and finer control over data availability. NoSQL architectures use distributed data stores to optimize scalability along with simple write, append and retrieval methods for fast performance, even with very large and diverse data sets.

NoSQL solutions are not designed to provide the same guarantees of Atomicity, Consistency, Isolation and Durability (ACID) that characterize RDBMSs. By leveraging the concept of eventual consistency, distributed databases can deal with huge volumes, velocity and variety of data while delivering very fast transactional performance. If no new updates are made for a given data item, eventually all accesses to that item will return the last updated value.

Depending on the specifics of the NoSQL architecture, this loosening of ACID consistency enables the database to provide high performance while scaling large data sets across distributed systems.

2.2 Differences in NoSQL Architectures

There are different approaches to classifying NoSQL databases, but the most widely used classification is by data model, such as Document, Key-Value/Object, Column or Graph.

The first category of NoSQL databases is built as collections of "documents". In general, these data stores assume that documents encapsulate and encode data in standard formats, such as XML, YAML, or JSON, as well as binary forms like BSON, PDF or Microsoft Office documents.

Compared to RDBMSs, document stores resemble tables and documents resemble records. But they are different: every record in a RDBMS table has the same sequence of fields, while different documents have fields that are completely different.

Each document is addressed in the data store via a unique key that represents that document. Beyond the simple key-document lookup, the NoSQL database can offer an API or query language to allow retrieval of documents based on their contents, or it can retrieve information using MapReduce techniques.

Key–value/Object stores allow applications to store data in a schema-less way, thereby providing a very high degree of flexibility. Usually, the data is stored as objects, but key-value stores also can be based on data types (i.e., integers, Booleans, characters, floating-point numbers, or alphanumeric strings).

Column-oriented databases are different from traditional row-oriented databases because of how they store data. By storing a whole column together instead of a row, you can minimize disk access when selecting a few columns from a row containing many columns. This has advantages for data warehouses, customer relationship management (CRM) systems, library card catalogs, and other ad-hoc inquiry systems where aggregates are computed over large numbers of similar data items.

Graph-oriented NoSQL databases are optimized for data relationships best represented in graphical form, such as networks of interconnected elements with an undetermined number of relations among them. Examples could be social networks, public transport links, or other networking topologies.

The following table provides an overview of popular NoSQL databases as categorized by their primary data models. It is important to note that some NoSQL approaches use a combination of data models. Cassandra is a case in point—it can be classified as either a Column or Key-Value data store because it has attributes of both.

Data Model	Database	API - Language	Notes
Column Store	Cassandra	Many Thrift languages	MapReduce
	Hadoop/HBase	Java/any writer	MapReduce Java
	Hypertable	Thrift (Java, PHP, Perl, Python, Ruby)	JQL, native Thrift API
Document Store	MongoDB	BSON (Binary JSON)	Dynamic, object-based
	CouchDB	JSON	MapReduce
	MarkLogic Server	JSON, XML, Java	Full text, XPath, Geospatial
	Clusterpoint	XML, PHP, Java, .NET	Full text, XML, XPath
Key-Value Store	CouchBase Server	Memcached API+ protocol	
	Riak	JSON	MapReduce term matching
	MemcacheDB	Memcached protocol	
Graph Databases	InfiniteGraph	Java	
	AllegroGraph	SPARQL	
	Neo4j	Java	Compatible with Java, Ruby, Python, Groovy and others.

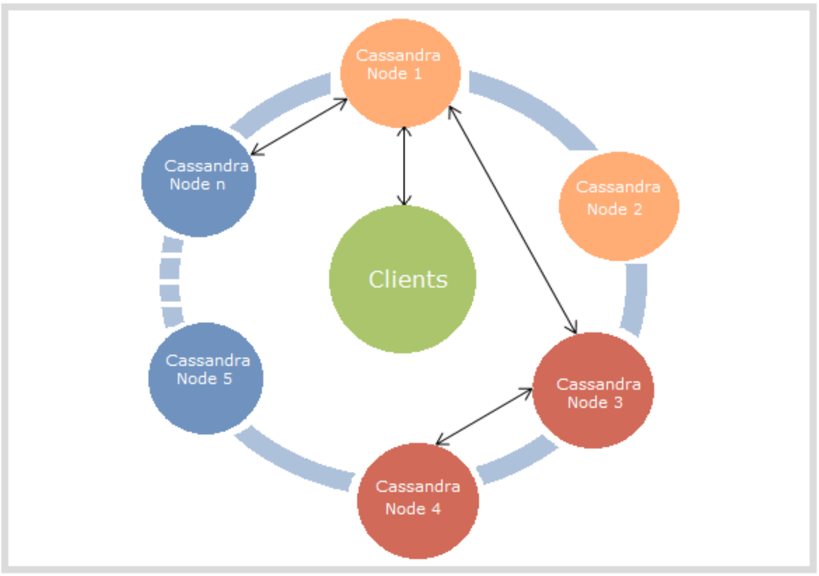
3.0 Cassandra

Cassandra is a scalable fully distributed NoSQL data storage system. First published by Facebook in 2008, Cassandra is licensed under the Apache License 2.0 for managing large amounts of loosely structured data. It aims to provide continuous availability with no single point of failure. Cassandra is essentially a hybrid between a key-value and a column-oriented tabular database. Each key in Cassandra corresponds to an object that groups values as columns, and columns are grouped together into sets called column families. Also, multiple column families can be grouped in super column families. Thus each key in Cassandra identifies a variable number of elements, which could be considered as multi-dimensional maps that are indexed by a single key.

A column family in Cassandra resembles a table in an RDBMS. Column families contain rows and columns. Each row is uniquely identified by a row key. Each row has multiple columns, each of which has a name, value, and a timestamp. Unlike a table in an RDBMS, different rows in the same column family do not have to share the same set of columns, and a column may be added to one or multiple rows at any time. In addition, applications can specify the sort order of columns within a Super Column or Simple Column family.

3.1 Cassandra Cluster Topology

Cassandra's topology relies on a peer-to-peer distributed architecture. All nodes participate in a "ring" or datastore cluster, and communicate with each other via a *gossip* protocol. Data, depending on the user preference is either orderly or randomly distributed across all nodes.



Additionally, data replication is available and is highly recommended to the user for fault-tolerance, as it allows for data durability in cases of node failures. The minimum replica set recommended is three. This means that should *n* nodes be found adequate for a given cluster/workload combination, *3n* nodes should be provisioned. The user would then enable Cassandra to replicate each data chunk to three nodes, allowing for any part of database to be accessible even under node failures. For greater fault-tolerance, a higher replica set number can be chosen.

3.2 Server Operating System Configuration and Tuning

Cassandra workloads can rapidly become CPU-bound, and as such, we recommend systems with as many CPU cores as possible. Operating system parameters also determine how well a Cassandra cluster will perform. Among those, specific system resource limits will need to be changed. Such settings can usually be found in /etc/security/limits.conf. Below are the values that should be adopted.

```
* soft nofile 32768

* hard nofile 32768

root soft nofile 32768

root hard nofile 32768

* soft memlock unlimited

* hard memlock unlimited

root soft memlock unlimited
```

root hard memlock unlimited

* soft as unlimited

* hard as unlimited

root soft as unlimited

root hard as unlimited

Additionally, you need to change kernel parameters controlling the maximum number of memory map areas used by Cassandra. This will enable the software to map more than 65536 areas, as it is by default.

sysctl -w vm.max_map_count=131072 To make this setting permanent,

vm.max_map_count = 131072 Should be added to: /etc/sysctl.conf

On RedHat Enterprise Linux, Oracle Enterprise Linux, and CentOS,Systems, system limits should be changed from 1024 to 10240 in /etc/security/limits.d/90-nproc.conf like so: soft nproc 10240

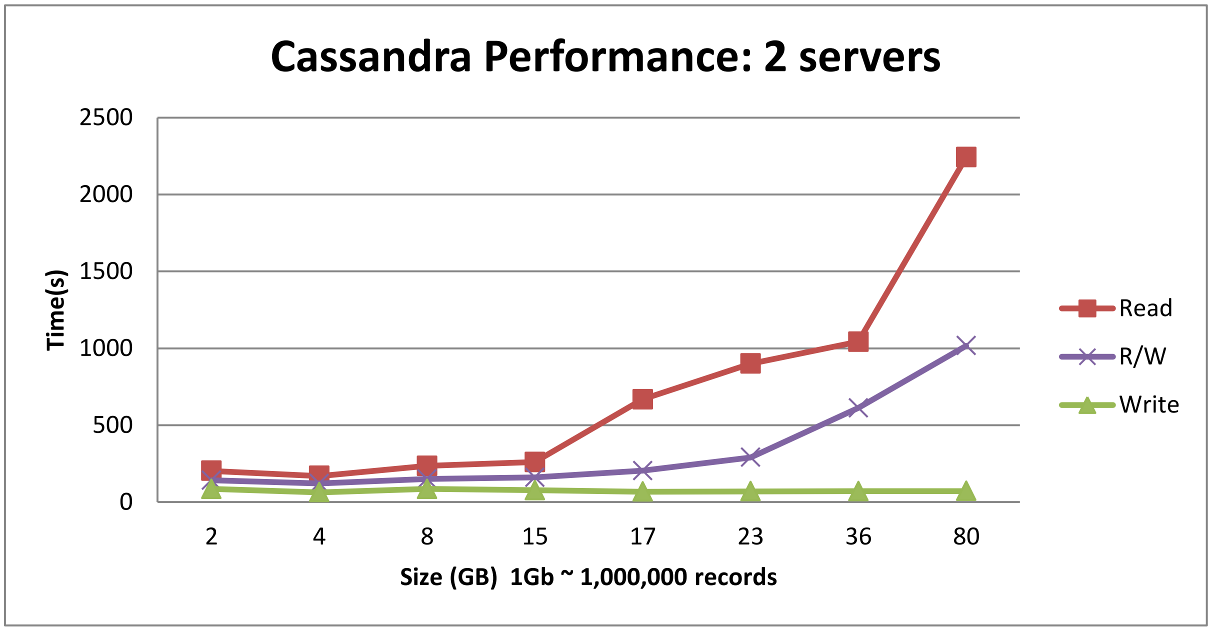
So that the Java Virtual Machine (JVM) does not endlessly swap, all paging spaces should be deactivated like so: sudo swapoff –all or as root swapoff –all This change can be made permanent by removing swap file entries from /etc/fstab

Tune the JVM Heap size on each node of the cluster, depending on the amount of memory on that specific node. Too big a Heap size can impair Cassandra's efficiency. Recommended settings are:

RAM	Heap Size
< 2GB	½ of RAM
2GB to 4GB	1GB
> 4GB	¼ of RAM, but not more than 8GB

3.3 Memory Size Selection

Cassandra relies on RAM for speed and efficiency for read-dominant workloads. More RAM allows Cassandra larger cache sizes, larger *memtables*, and thus fewer disk flushes. However, rather than using most of the available RAM on a system, Cassandra through the JVM will only make use of 8GB. If your workload is predominantly composed of read operations, to achieve best performance, you should allow for enough nodes to host the dataset in RAM. This considers that each node will have only 8GB of its RAM usable to Cassandra. As our testing shows, performance begins deteriorating past 15.7GB of data for 2 servers, thus about 8GB per server. As we point out in the table above this section, assigning more than 8GB of heap size to the JVM proves to be counterproductive in Cassandra's case.



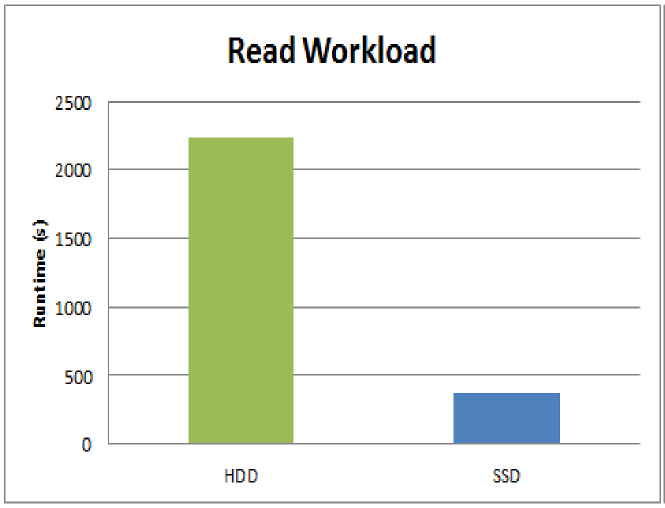
3.4 Storage Device Configuration and Tuning

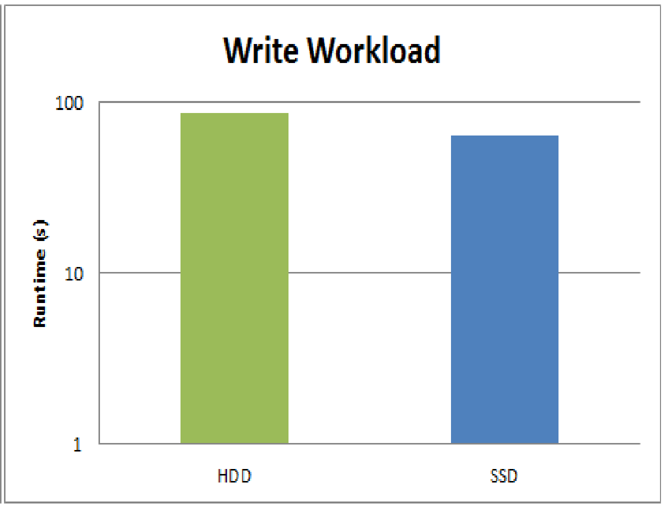
If the intended workload for a Cassandra deployment will not fit in RAM, secondary storage will be more frequently accessed. Opting for storage solutions offering faster I/O operations, such as RAID or SSDs, becomes crucial for out of memory read-dominant workloads, as shown in the figure below.

Read-dominant workloads show greater sensitivity to this condition than write-dominant ones, as read operations tap secondary storage far more than writes in Cassandra's case. As an example, our tests with various workloads showed SSDs to perform up to 6X faster than spinning disks for read-dominant workloads, as compared to 25% faster for write-dominant workloads. When dealing with in-memory workloads, the use of spinning drives is generally adequate, as the performance numbers observed between those and SSDs show very little variation for Intel Xeon and Intel Atom processor-based platforms, with the exception of writes on Intel Xeon-based systems. For write-dominant workloads, Intel Xeon processor-based platforms show up to a 24% performance increase with SSDs over spinning disks, when it comes to in-memory writes workloads.

Cassandra writes to the disk when appending content to its “commit log”, and when flushing memtables for durability. Storing “commit logs” on a separate disk from the data disks will benefit performance.

When selecting storage sizes for a given workload, we recommend provisioning for a minimum of 1.1 times the size of the workload to be processed.





3.5 Networking

As per our testing, the recommended bandwidth for Cassandra is a 1Gb internal network between the nodes. A faster network might be generally preferred, but in Cassandra’s case, 1Gb is sufficient.

3.6 Cassandra Use Cases

3.6.1 Intel Atom processor-based Cassandra Deployment

For each workload, we assume a 1Gb internal network between all Cassandra nodes. Assuming 32GB of RAM per Intel Atom processor-based microserver, and given that 8GB will be provisioned for Cassandra, below are possible cluster configurations along with resulting performance expectations. These scenarios assume a fault-tolerance cluster with a replication count of three. (The same scenarios can be considered without replication, by dividing the number of suggested nodes by three.)

For a maximum throughput of 800 read operations/second at the client, the table below should be followed to determine the number of servers. For a solely read-intensive workload, spinning drives would be sufficient. The total disk capacity across all servers should be greater or equal to 1.1 times the size of the workload. SSDs in this scenario do not add much performance, only 0.3% when the workload is fully hosted in memory. For write-intensive workloads, a throughput of 1400 operations/second at the client can be achieved with the use of spinning disks. As writes occur in memory, the use of SSDs offers only 1.5% more performance to our workload. The total disk capacity across all servers in this case also should be greater or equal to 1.1 times the size of the workload.

Workload Size	# of Servers
100GB	39
1TB	378

At the cost of performance, an alternate configuration choice is possible with the use of SSDs and a reduced number of total participating servers. This also could be the case when not enough servers are available to accommodate a given workload. The table below highlights such a case for our two scenarios (100GB, and 1TB of user data).

This configuration choice provides 100 read operations/second versus the 800 read operations/second previously noted because the data is no longer directly fetched from RAM. SSDs in this case will provide 600 read operations per second compared to 100 operations with spinning disks. For write-intensive applications in the configuration below, we can expect 1200 write operations per second with spinning drives, and a 25% improvement over spinning drives with SSDs.

Workload Size	# of Servers
100GB	3
1TB	3

3.6.2 Intel Xeon processor-based Cassandra Deployment

Because Cassandra effectively uses only 8GB of heap space, despite the larger overall memory capacity, the number of servers recommended is the same as in the Atom processor-based scenario. However, Cassandra still benefits from larger memory systems, as larger pages can be cached.

For each workload, we assume a 1Gb internal network between all Cassandra nodes. Considering 512GB of RAM per server, and given that only 8GB of heap space will be used by Cassandra, below are possible Intel Xeon processor-based performance expectations:

In this scenario we assume a fault-tolerance cluster with a replication count of three. If this same scenario is considered without replication, the number of suggested nodes should be divided by three. For a maximum throughput of 1200 read operations/second at the client, the table below should be followed to determine the number of servers. For a solely read-intensive workload, spinning drives would be sufficient, as SSDs only provide a 1.5% performance boost. The total disk capacity across all servers should be greater than or equal to 1.1 times the size of the workload. For write-intensive workloads, a throughput of 2900 operations/second at the client can be achieved with the use of spinning disks, and a 24% higher throughput with SSDs. The total disk capacity across all servers in this case also should be greater than or equal to 1.1 times the size of the workload.

Workload Size	# of Servers
100GB	39
1TB	378

At the cost of performance, an alternate configuration is possible with the use of SSDs and a reduced amount of total participating servers. This also can be the case when not enough servers are available to accommodate a workload. The table below highlights such a case for our two scenarios (100GB, and 1TB of user data).

This configuration will provide 100 read operations/second versus the 1200 read operations/second previously noted, because the data is no longer directly fetched from RAM. SSDs in this case would provide 600 read operations per second compared to 100 operations with spinning disks. For write-intensive applications on the other hand, we can expect 2800 write operations per second with spinning drive, and 3500 operations per second with SSDs.

Workload Size	# of Servers
100GB	3
1TB	3

4.0 Summary

NoSQL provides important solutions for effective data management across a growing range of scenarios including compute-intensive applications, such as real-time Big Data analytics, as well as I/O-intensive requirements, such as serving hot-trending social media peak usage patterns.

The flexibility of NoSQL has also driven its usage in realms beyond Big Data. Because NoSQL is basically about data storage and management, it provides advantages for applications such as entry dedicated hosting, static web serving, simple content delivery and memory caching.

Cassandra has emerged as a leading open-source NoSQL data store that offers a high level of flexibility, scalability and performance for addressing these challenges.

The proper hardware configuration for optimizing performance and cost of ownership for Cassandra is not a one-size-fits-all proposition. Key factors that must be considered include:

- Size of the workload and future growth predictions
- Performance requirements and criticality vs. cost of ownership considerations
- Nature of workload: read-intensive, write-intensive, or both
- Overall data center considerations: space, power, interoperability

As detailed in this guide, the deployment of Cassandra can be optimized to achieve a targeted balance of these key factors using either Intel Xeon processor-based or Intel Atom processor-based solutions, depending on the specific requirements of the application.

The use of Intel Architecture based solutions for Cassandra deployments also takes advantage of the extensive Intel ecosystem, reduces overall complexity and improves TCO by providing a common hardware and software architecture that can be optimized for performance, cost and maintainability throughout the data center.

5.0 Additional Resources

Open Data Center Alliance (ODCA) Big Data Consumer Guide
http://www.opendatacenteralliance.org/docs/Big_Data_Consumer_Guide_Rev1.0.pdf

The Apache Cassandra Project
http://cassandra.apache.org/

Apache Cassandra Quick Doc
http://planetcassandra.org/Content/Learn/Docs/QuickDoc.pdf