JVM Performance Study
Comparing Java HotSpot®
and Azul Zing® Using
Apache Cassandra™
Legal Notices

Apache Cassandra™, Spark™ and Solr™ and their respective logos are trademarks or registered trademarks of the Apache Software Foundation in the US and/or other countries.

Azul Systems®, Zing® and the Azul logo are trademarks or registered trademarks of Azul Systems, Inc.

Linux® is a registered trademark of Linus Torvalds. CentOS is the property of the CentOS project.

Oracle®, Java™, and HotSpot™ are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

Intel® and Intel® Xeon® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

GitHub® is a registered trademark of GitHub, Inc.

Other marks are the property of their respective owners and are used here only for identification purposes

©Copyright 2015 Azul Systems, Inc. All rights reserved.
Table of Contents

Legal Notices.................................................................................................................................................. 2
1 Executive Summary..................................................................................................................................... 4
2 Background ............................................................................................................................................. 5
3 Testing Environment ............................................................................................................................. 5
4 Testing Methodology ............................................................................................................................ 7
5 Results .................................................................................................................................................. 8
6 Conclusion ............................................................................................................................................ 10
7 References ........................................................................................................................................... 11
8 Appendix ............................................................................................................................................ 12
1 Executive Summary

Big data technologies are transforming enterprise operations by making it possible to process massive datasets and deliver new and innovative solutions, such as web personalization, product recommendations, catalogue lookups, real-time analytics, fraud detection and drug discovery. But ensuring consistent, low latency performance isn’t a guarantee for all Java-based big data applications. Careful attention to product choices, runtime components and deployment topologies are essential to maximizing the values of these new big data solutions, including Apache Cassandra and supporting components such as Spark and Solr.

This benchmark study compares the response time performance of two different Java Virtual Machines (JVMs), namely Azul Zing and Java HotSpot, while running Apache Cassandra at different throughput levels. All benchmark results were derived using the cassandra-stress framework, a utility for load testing and benchmarking a Cassandra cluster, and the jHiccup Java Virtual Machine (JVM) measurement tool.

The testing methodology was designed to measure JVM response time consistency based on percentiles (e.g. 99%, 99.9%, 99.99%, etc.) and Cassandra application max outliers at different throughput rates. The benchmark was configured with three Cassandra data nodes and one load server. The jHiccup measurement tool was used to capture and graph the response time distribution of the two JVMs for all benchmark runs.

The results of the benchmark show that response time distributions and runtime consistency vary dramatically between the two JVMs. For Java HotSpot, which employs a stop-the-world young generation collector, response time variance ranged between 1 millisecond (msec) to a maximum of 667 msec. In contrast the Azul Zing runtime showed response time consistency at all throughput rates (e.g. 10K to 60K OPS). When comparing Cassandra max outliers at different throughput rates, Zing deployments were 14x to 51x better than similarly configured Java HotSpot deployments. This difference in response time profile of the two JVMs suggests that for Cassandra deployments that require consistent low latency (i.e. have implicit or explicit SLAs), only the Azul Zing JVM can ensure runtime consistency that will not stall the application and contribute to Cassandra response time variations.
This paper describes the testing environment, testing methodology and resulting response time profiles of the two JVMs while supporting an Apache Cassandra application. Companies deploying or running big data applications, including Cassandra, can use this information to ensure they are using the correct JVM to meet their specific business requirements or they can leverage this benchmark to design their own testing scenarios.

## Background

Cassandra is a NoSQL database that can scale elastically and enables sub-second response times with linear scalability. DataStax Enterprise (DSE), built on Apache Cassandra, is a distributed database for online applications that require fast performance with no downtime. Because Apache Cassandra, Spark, Solr and other DSE components are written in Java, they require a Java Virtual Machine (JVM) for runtime deployment. Since not all JVMs are the same and employ different garbage collection algorithms (e.g. Concurrent Mark Sweep, C4, etc.), the benchmark was configured to capture JVM response time variances as well as application max outliers at different throughput rates (i.e. operations per second or OPS).

For this performance study the cassandra-stress 2.1 load tool, a Java-based stress testing utility for load testing and benchmarking Cassandra clusters, was used to provide a reproducible way to simulate a specific transaction load and measure the performance of the individual Cassandra nodes, including max response times. To ensure accurate runtime measurements of the two JVMs, including the contributions of Java garbage collection (GC) pauses, jHiccup was added to the cassandra-stress benchmarking framework and used to capture and graph JVM response time profiles by percentiles (e.g. 99%, 99.9%, 99.99%, etc.). Designed to only measure JVM responsiveness, jHiccup charts shows “the best possible response time” the application could have experienced at given percentile (e.g. 99%). jHiccup is not an end-to-end performance measurement tool and does not capture the additional overhead of the Cassandra application or related transaction logic.

## Testing Environment

The test environment consisted of four nearly identical Iron Systems® servers:
- Cassandra servers A, B, and C (i.e. nodes 1, 2 and 3)
- Cassandra-stress load server

Each Cassandra server had 4 Intel® Xeon® processors with 512 GB of memory, running CentOS 6.0 and DataStax Enterprise version 4.5.3 that includes Cassandra version 2.0. The three Cassandra machines and the load server were directly interconnected using Solarflare Communications Solarstorm SFC9020 10GbE network cards. The exact configurations are listed below:

<table>
<thead>
<tr>
<th>Machine Configuration</th>
<th>Cassandra Server A (node 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Iron Systems</td>
</tr>
<tr>
<td>Processors (x 4)</td>
<td>Intel® Xeon® CPU E7-4820 @ 2.00GHz</td>
</tr>
<tr>
<td>Memory (x 32)</td>
<td>16GB RDIMM, 1066MHz, Low Volt, Dual Rank</td>
</tr>
<tr>
<td>Networking</td>
<td>1 x Solarflare Communications SFC9020</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 6.0</td>
</tr>
</tbody>
</table>
Machine Configuration  Cassandra Server B (node 2)
Manufacturer                      Iron Systems
Processors (x 4)                  Intel® Xeon® CPU E5-4620 0 @2.20GHz
Memory (x 32)                     16GB RDIMM, 1333MHz, Low Volt, Dual Rank
Networking                        1 x Solarflare Communications SFC9020
OS                                 CentOS 6.0

Machine Configuration  Cassandra Server C (node 3)
Manufacturer                      Iron Systems
Processors (x 4)                  Intel® Xeon® CPU E5-4620 0 @2.20GHz
Memory (x 32)                     16GB RDIMM, 1333MHz, Low Volt, Dual Rank
Networking                        1 x Solarflare Communications SFC9020
OS                                 CentOS 6.0

As recommended to maximize performance, Cassandra was configured with a Replication Factor of 3 and with the Quorum set to 3. Cassandra-stress 2.1 was configured with a master process on the load server and issued requests directly to the Cassandra cluster. jHiccup version 2.0.2 was used to capture JVM response times across all 3 Cassandra nodes and graph the response time distribution up to the 99.999th percentile.

Both Java HotSpot and Azul Zing were configured on the Cassandra servers to use 16 GB heaps. The Java HotSpot JVM was configured to use the CMS collector and as recommended to maximize performance was configured using the following flags:

-XX:--Xms16G -Xmx16G -Xmn8G -Xss256k
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
-XX:+CMSParallelRemarkEnabled
-XX:SurvivorRatio=8
-XX:MaxTenuringThreshold=1
-XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly
-XX:+UseTLAB
-XX:+CMSParallelInitialMarkEnabled
-XX:+CMSEdenChunksRecordAlways
-XX:+UseCondCardMark
-XX:+PrintGCDetails -verbose:gc
-Djava.net.preferIPv4Stack=true
-Dcom.sun.management.jmxremote.port=$JMX_PORT
-Dcom.sun.management.jmxremote.rmi.port=$JMX_PORT
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
-Xloggc:/var/log/cassandra/gc.log
-XX:+PrintGCApplicationStoppedTime
-XX:--Xloggc=/var/log/cassandra/gc.log
-XX:+PrintGCDetails -verbose:gc
-javagen:
javaagent:/home/bsilva/jHiccup-2.0.2/jHiccup.jar=""-d,30000"

The Azul Zing runtime employed few runtime flags and used the default Azul C4 pauseless collector. For both Zing and Java HotSpot, the CentOS operating system was configured to use LargePages.

-XX:+UseLargePages
-javagen:
javaagent:/home/bsilva/jHiccup-2.0.2/jHiccup.jar=""-d,30000""
4 Testing Methodology

The cassandra-stress tool is a Java-based stress testing utility for basic benchmarking and load testing a Cassandra cluster. The load generator attempts to produce a load of “xx,xxx” transactions per second, of a given transaction type mix, and measure the observed transaction latencies. This tool is commonly used to support capacity planning decisions.

Since the objective of this performance study was to measure JVM response times and max application outliers at a given throughput, performance runs for Zing and HotSpot ranged from 10K to 60K operations per second (OPS). Cassandra was configured for 3 nodes with a Replication Factor of 3 and with the Quorum set to 3 across three physical servers.

The Cassandra transactional mixed for each run was set using the following string (only the TARGET_OPS was varied from run to run):

```
$cassandra-stress user
profile=$STRESS_DIRECTORY/tools/bin/stress/stress320.yaml
ops\{insert=33,read=66,delete=1\} n=$N cl=QUORUM -rate threads=80
limit=${TARGET_OPS}/s -mode native cql3 -node
file=$STRESS_DIRECTORY/tools/bin/stress/nodes -log $LOG_PARAMS
```

The reported latencies from the cassandra-stress tool suffer from Coordinated Omission (see pages 30-46), as do many other load generators. As such, only the “op rate”, “partition rate”, “row rate” and “latency max” were saved for each run:

```
Results:
op rate : 10003
partition rate : 5323
row rate : 5323
latency max : 376.7
```

To compensate for this measurement and for latency percentiles calculation errors, JVM results were captured and charted using the open source jHiccup agent which attached to each Cassandra process on each node.

A sample jHiccup chart for Java HotSpot is shown below:
5 Results

By aggregating the different Cassandra application max outliers at different OPS for both the Azul Zing and Java HotSpot JVMs, we can accurately compare response times variances of the two Java runtimes (e.g. at 30K OPS Cassandra on Zing had a max response time of 11.8 msec vs. HotSpot deployment which had a 594.3 msec max).

If we look at the individual jHiccup charts for Zing and HotSpot (e.g. 30K OPS), we again see different JVM response time profiles. While Java HotSpot generally performed well at the 95th percentile, it starts to encounter response variances at 99th and higher percentiles which was directly associated with its young generation, stop-the-world CMS (Concurrent Mark Sweep) collector. If we look specifically at the jHiccup histogram for Java HotSpot at 30K OPS, we see that over this relatively short 10 minute benchmark run JVM encountered 9 GC spikes over 200 msec; one of which exceeded 360 msec. Meanwhile, Azul Zing never
exceeded 1.6 msec throughout the entire run for the same throughput rate and transactional mix.

If we examine the Java HotSpot GC Stopped Time histogram (figure 5) over the length of the run, we’ll see several spikes related to garbage collection pauses (similar to what was reported by jHiccup in figure 3). At higher operations per second, HotSpot pauses increased and at 60K OPS, which was near saturation for HotSpot, the max Cassandra outlier exceeded 640 msec, while the related JVM max latency was over 600 msec as captured by the jHiccup tool (figure 4).

Figure 3 – Java HotSpot and Azul Zing jHiccup Charts for Node #3 at 30K OPS

Figure 4 – Java HotSpot jHiccup Charts for Nodes 1, 2 and 3 at 60K OPS
6 Conclusion

This benchmark study demonstrates that application performance and runtime consistency are affected by the choice of JVM used with Cassandra. Since different JVMs employ different garbage collection (GC) algorithms, application response time consistency can vary based on which collector is used (i.e. not all Java runtimes are the same). For applications that have real-time use cases (e.g. fraud detection) or have explicit or implicit SLAs, careful attention to application characteristics, such as heap size, live set size, objection allocation rates, and mutation rates are important factors when choosing a JVM and garbage collector that can meet your specific deployment requirements.

When metrics such as “Sustained Throughput” (i.e. a throughput rate which never exceeds a specific response time SLA), and time-to-production are important, Azul Zing can provide better Java runtime metrics with less JVM tuning and reduce devastating application pauses. For big data solutions such as Cassandra and use cases that require runtime consistency or can benefit from larger Java heaps (e.g. Solr, Spark), Azul Zing provides greater runtime consistency and a viable alternative to Java HotSpot. When Cassandra is deployed with strict SLAs, e.g. 50 – 500 milliseconds, only Azul Zing can guarantee deployment success.
7 References

DataStax Enterprise
http://www.datastax.com/what-we-offer/products-services/datastax-enterprise

Azul Systems C4 – Continuously Concurrent Compacting Collector (ISMM paper)
http://www.azulsystems.com/products/zing/c4-java-garbage-collector-wp

Understanding Java Garbage Collection White Paper
http://www.azulsystems.com/dev/resources/wp/understanding_java_gc

Blog post: If you’re not measuring and/or plotting the Max, what are you hiding (from)?
http://latencytipoftheday.blogspot.com/2014/06/latencytipoftheday-if-you-are-not.html

The cassandra-stress tool
http://www.datastax.com/documentation/cassandra/2.0/cassandra/tools/toolsCStress_t.html

JHiccup open source performance measurement tool
http://www.azulsystems.com/jhiccup

Azul Inspector
http://www.azulsystems.com/dev_resources/azul_inspector

Azul Systems Zing FAQ
http://www.azulsystems.com/products/zing/faq

Contact

Azul Systems

Phone: +1.650.230.6500
Email: info@azulsystems.com
Web: www.azulsystems.com
Twitter: @AzulSystems
## 8 Appendix

### 8.1 Test Durations and Settings

<table>
<thead>
<tr>
<th>Cassandra Cluster Configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>3</td>
</tr>
<tr>
<td>Replication</td>
<td>3</td>
</tr>
<tr>
<td>Quorum Setting</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cassandra-stress Configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Duration</td>
<td>10 minutes (600 seconds)</td>
</tr>
<tr>
<td>Operations/sec</td>
<td>10,000-60,000</td>
</tr>
<tr>
<td>Transaction mix</td>
<td>33% insert, 66% read, 1% delete</td>
</tr>
</tbody>
</table>
8.2 Hiccup Charts for All Runs

Java HotSpot 10K OPS (actual 10,003 OPS)

Results:
op rate : 10003
partition rate : 5323
row rate : 5323
latency max : 376.7 (msec)
Total operation time : 00:09:59
END
Azul Zing 10K OPS (actual 10,003)

Results:
- op rate : 10003
- partition rate : 5318
- row rate : 5318
- latency max : 8.8 (msec)

Total operation time : 00:09:59

END
Azul Zing 15K OPS (actual 15,007 OPS)

Results:
- op rate : 15007
- partition rate : 7878
- row rate : 7878
- latency max : 7.8 (msec)
- Total operation time : 00:05:33
Azul Zing 15K OPS (node 1) by Time Interval

Max per Interval 99% 99.9% 99.99% Max

Azul Zing 15K OPS (node 2) by Time Interval

Max per Interval 99% 99.9% 99.99% Max

Azul Zing 15K OPS (node 1) by Percentile Distribution

Max=1.11

Azul Zing 15K OPS (node 2) by Percentile Distribution

Max=2.15

Azul Zing 15K OPS (node 3) by Time Interval

Max per Interval 99% 99.9% 99.99% Max

Azul Zing 15K OPS (node 3) by Percentile Distribution

Max=1.67
Java HotSpot 20K OPS (actual 20,004 OPS)

Results:
op rate : 20004
partition rate : 11551
row rate : 11551
latency max : 454.1 (msec)
Total operation time : 00:09:59
END
Azul Zing 25K OPS (actually 25,002 OPS)

Results:
- op rate : 25002
- partition rate : 15040
- row rate : 15040
- latency max : 11.4 (msec)

Total operation time : 00:09:59
END
Azul Zing 30K OPS (actual 30,011 OPS)

Results:

- op rate : 30011
- partition rate : 18876
- row rate : 18876
- latency max : 11.8 (msec)

Total operation time : 00:09:59

END
Java HotSpot 30K OPS (actual 30,004 OPS)

Results:
- op rate : 30014
- partition rate : 18870
- row rate : 18870
- latency max : 594.3 (mec)

Total operation time : 00:09:59
END
Azul Zing 40K OPS (actual 40,003 OPS)

Results:
- op rate : 40003
- partition rate : 26998
- row rate : 26998
- latency max : 20.8 (msec)

Total operation time : 00:09:59

END
Java HotSpot 40K OPS (actual 40,006 OPS)

Results:
- op rate : 40006
- partition rate : 27002
- row rate : 27002
- latency max : 586.8 (msec)

Total operation time : 00:09:59
END
Azul Zing 50K OPS (actual 50,006 OPS)

Results:
- op rate : 50006
- partition rate : 35656
- row rate : 35656
- latency max : 27.5 (msec)

Total operation time : 00:09:59
END
Java HotSpot 50K OPS (actual 50,007 OPS)

Results:
op rate : 50007
partition rate : 35649
row rate : 35649
latency max : 667.6 (msec)
Total operation time : 00:09:59
END
Azul Zing 60K OPS (actual 60,0019 OPS)

Results:
- \text{op rate} : 60019
- \text{partition rate} : 44613
- \text{row rate} : 44613
- \text{latency max} : 45.8 \text{ (msec)}

Total operation time : 00:09:59
END
Java HotSpot 60K OPS (actual 59,995 OPS)

Results:
- op rate : 59995
- partition rate : 44585
- row rate : 44585
- latency max : 643.3 (msec)

Total operation time : 00:10:00
END