JVM Performance Study Comparing Java HotSpot® to Azul Zing® Using Red Hat® JBoss® Data Grid

Legal Notices

Table of Contents

Legal Notices 2

1 Executive Summary 4

2 Background 5

3 Testing Environment 5

4 Testing Methodology 7

5 Results 9

6 Conclusion 10

7 References 11

8 Appendix 12

1 Executive Summary

While new In-memory Computing (IMC) techniques hold the promise of better scalability and performance, ensuring consistent, low latency performance isn't a guarantee for all Java applications. Careful attention to product choices, runtime components and deployment topologies are essential to maximizing the values of these new IMC solutions, including In-memory Data Grids (IMDGs).

This benchmark study compares the response time performance of two different JVMs, namely Azul Zing and Java HotSpot, while running Red Hat JBoss Data Grid at different node sizes. All benchmark results were derived using the open source RadarGun framework and the jHiccup measurement tool.

The testing methodology was designed to measure response time consistency and throughput of the data grid nodes at different JVM heap sizes. The data grid was configured for distributed mode with one data node and one backup node replicated across two physical servers (4 nodes total). The jHiccup measurement tool was used to capture and graph the response time distribution of the two JVMs during all benchmark runs.

The results of the benchmark clearly show that response time distributions and runtime consistency vary widely between the two JVMs. For Java HotSpot, which employs a stop-the-world young generation collector, response time variance was more than 250x across individual runs (i.e. was not measurable using standard deviation). In contrast the Azul Zing runtime showed highly predictable response time consistency across the entire distribution. At 5 "9s" (i.e. 99.999%) Zing had a maximum response time of only 23 milliseconds when the grid was deployed using a 200 GB node. In addition, Java HotSpot, which reported response time outliers exceeding multiple seconds at 100 GB, was not able to complete the benchmark run at 200 GB because of system saturation due to consecutive garbage collections.

The profound difference in response time profile of the two JVMs suggests that for Java deployments that require low latency SLAs, large individual instances (e.g. 20+ GB heaps), or higher sustained throughput, only the Zing JVM can keep up with object allocation rates without stalling the application. For data grid deployments, this could mean 1) lower node response times (<50 msec), 2) greater resistance to node failures due to heavy load and 3) larger node capacity with support for Java heaps up to 340 GB.

This paper describes the testing environment, testing methodology and resulting response time profiles of two different JVMs while running JBoss Data Grid. Companies deploying or running Java applications including IMDGs can use this information to make more informed JVM choices to meet their specific business use case or they can leverage this benchmark to design their own testing scenarios.

2 Background

Red Hat JBoss Data Grid, based on the Infinispan open source community project, is a distributed in-memory grid, supporting local, replicated, invalidation and distributed modes, While JBoss Data Grid can be configured in a number of different topologies, it is typically deployed as a data tier to provide faster read/write behavior and data replication while offloading the database or persistent data store. As an in-memory schema-less key/value store, JBoss Data Grid can provide a simple and flexible means to storing different objects without a fixed data model.

Because the JBoss Data Grid is a Java application, it requires 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), the benchmark was configured to capture JVM response time variances for different Java heap sizes, live set sizes, objection allocation rates, mutation rates and other application-specific characteristics.

For this performance study, the RadarGun benchmarking framework coupled with the jHiccup measurement tool was used to simulate load and measure response times of the JVMs. Originally designed to compare the performance of different data grid products, RadarGun provides a reproducible way to simulate a specific transaction load and measure the performance of the individual data grid nodes. To ensure accurate runtime measurements, even during long garbage collection (GC) pauses, jHiccup was added to the benchmark. Designed to measure only JVM responsiveness, jHiccup shows “the best possible response time” the application or data grid could have experienced during the benchmark run. jHiccup is not an end-to-end performance measurement tool and does not capture the additional overhead of the application or data grid transaction logic.

3 Testing Environment

The test environment consisted of two nearly identical Iron Systems® servers; Node Servers A and B. Each server had 4 Intel® Xeon® processors with 512 GB of memory, running Red Hat Enterprise Linux 6 and JBoss Data Grid version 6.2. The two machines were directly interconnected using Solarflare Communications Solarstorm SFC9020 10GbE network cards. The exact configurations are listed below:

Machine Configuration	**Node Server A**
Manufacturer	Iron Systems
Processors (x 4)	Intel® Xeon® CPU E7-4820 @ 2.00GHz
Memory (x 32)	16GB RDIMM, 1066MHz, Low Volt, Dual Rank
Networking	1 x Solarflare Communications SFC9020
OS	Red Hat Enterprise Linux Server release 6.2

Machine Configuration	Node Server B
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	Red Hat Enterprise Linux Server release 6.2

The data grid was configured on the two servers in distributed mode with 1 backup or copy node replicated on the other system (i.e. 4 nodes in total). The RadarGun benchmark was configured with a master process on Node Server A which communicated directly with all 4 nodes via an in-process client which in turn issued requests to individual data grid nodes. jHiccup version 1.3.7 was used to launch the RadarGun benchmark, capturing JVM response times across all 4 grid nodes and then graphing the response time distribution up to the 99.999th percentile.

Both servers where configured to support both JVMs. The Java HotSpot JVM was configured to use the CMS collector with the following flags:

```
-XX:NewRatio=3
-XX:+UseConcMarkSweepGC
-XX:+UseParNewGC
-XX:MaxPermSize=512m
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+UseLargePages
```

The Azul Zing runtime did not employ any runtime flags and used the default C4 collector. For both Zing and Java HotSpot, the RHEL operating system was configured to use LargePages:

```
 -XX:+UseLargePages
```

Java Configuration	JVM Version
Azul Zing	Azul Systems Zing java version "1.7.0-zing_5.8.0.0" Zing Runtime Environment for Java Applications (build 1.7.0-zing_5.8.0.0-b4) Zing 64-Bit Tiered VM (build 1.7.0-zing_5.8.0.0-b15-product-azlinuxM-X86_64, mixed mode)
Java HotSpot	Java HotSpot java version "1.7.0_05" Java SE Runtime Environment (build 1.7.0_05-b06) Java HotSpot 64-Bit Server VM (build 23.1-b03, mixed mode)

4 Testing Methodology

The benchmark was designed to measure JVM response times and throughput of JBoss Data Grid at multiple JVM heap sizes. Performance runs at 20, 40, 60, 100 and 200 GB heaps were repeated on both the Azul Zing and Java HotSpot JVMs. JBoss Data Grid was configured in distributed mode with just a single copy of the data and a backup replicated across two physical servers.

In order to simulate real world conditions, the individual grid nodes and corresponding JVM memory heaps, were 'primed' by the RadarGun benchmark for each test and before jHiccup measurements were taken. Initial data loads were approximately 40% of the maximum heap size for each run. In order to approximate "typical" Java HotSpot response times and maximum outliers during this short duration benchmark (i.e. 20-30 minutes), the default NewRatio=3 was used to avoid full garbage collection pauses during the measurement period[1]. This ratio resulted in 25% of the heap allocated to the young generation and 75% dedicated to the old generation.

In addition to priming the data nodes, the benchmark utilized multiple object sizes (i.e. 15% 1K, 35% 2K and 50% 4K) to better simulate real world scenarios. For the first 3 Java heap sizes (20, 40, and 60 GB), the transactional mix of the RadarGun benchmark was set to a ratio of 67% read, 32% write, 1% remove. However, in order to accelerate the priming of the data notes to 40% for the 100 and 200 GB runs, the transactional mix was set to 39% read, 60% write, and 1% remove.

Because the RadarGun benchmark was primarily designed to measure the throughput of individual grid nodes for a given load, the open source jHiccup tool was added to capture and graph the responsiveness of the JVMs during the benchmark runs. The tool captured the JVM response times for all 4 nodes and then combined the distributions into a single graph or "Hiccup Chart".

The graphs shown below are the individual Hiccup Charts for the Java HotSpot test at 60 GB:

[1] The test was configured to simulate a "well-tuned" Java HotSpot JVM, that is, to delay or prevent a full garbage collection (with resulting application pauses) as long as possible. This is how an application would normally be configured in production. Without this additional tuning, Java HotSpot maximum response times were likely to be orders of magnitude larger than they were in our test.

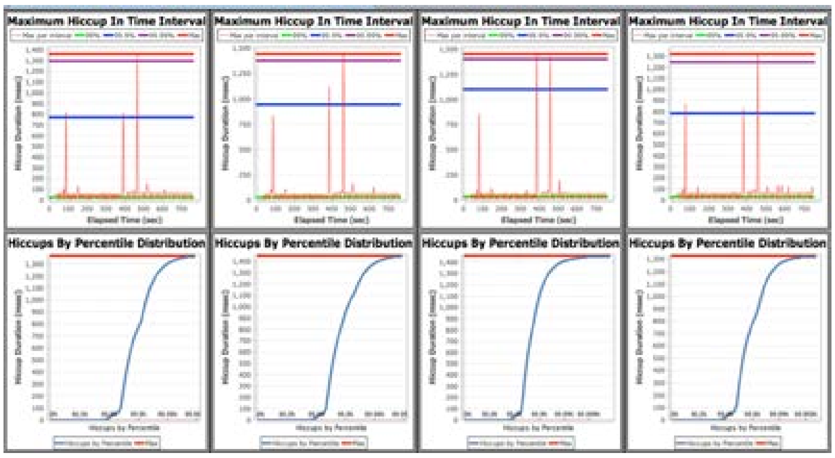


Figure 1 – Individual Hiccup Chart for Java HotSpot using 60 GB heaps

The benchmark ran the same transaction loads on each JVM at different Java heap sizes (20, 40, 60, 100 and 200 GB). Both JVM response times and RadarGun throughput metrics were captured for each run. Below is the RadarGun configuration for the 20 GB heap test.

RadarGun Benchmark Configuration – 20GB heap test	
Benchmark Duration	15 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	50,000
Transaction mix	67% read, 32% write, 1% remove
Number of threads	30

5 Results

By overlaying the different Hiccup Charts for all the Azul Zing and Java HotSpot runs, we can clearly see that response time consistency is vastly different for the two JVMs. While Java HotSpot performed well approximately 90% or the time (or 9 out of every 10 transactions), it starts to encounter response time delays at the 95th percentile directly associated with its young generation CMS (Concurrent Mark Sweep) collector. Pauses for Java HotSpot at 20 GB approach 500 milliseconds (or ½ second) at the 99.9th percentile. At 40 GB Java HotSpot approaches 1 second pauses, and at 60 GB, the JVM stalls approach 2 seconds. On even larger heaps Java HotSpot outliers start to dramatically increase (i.e. 5 seconds at 100 GB) and the JVM could not complete the benchmark at 200 GB because of severe thrashing.

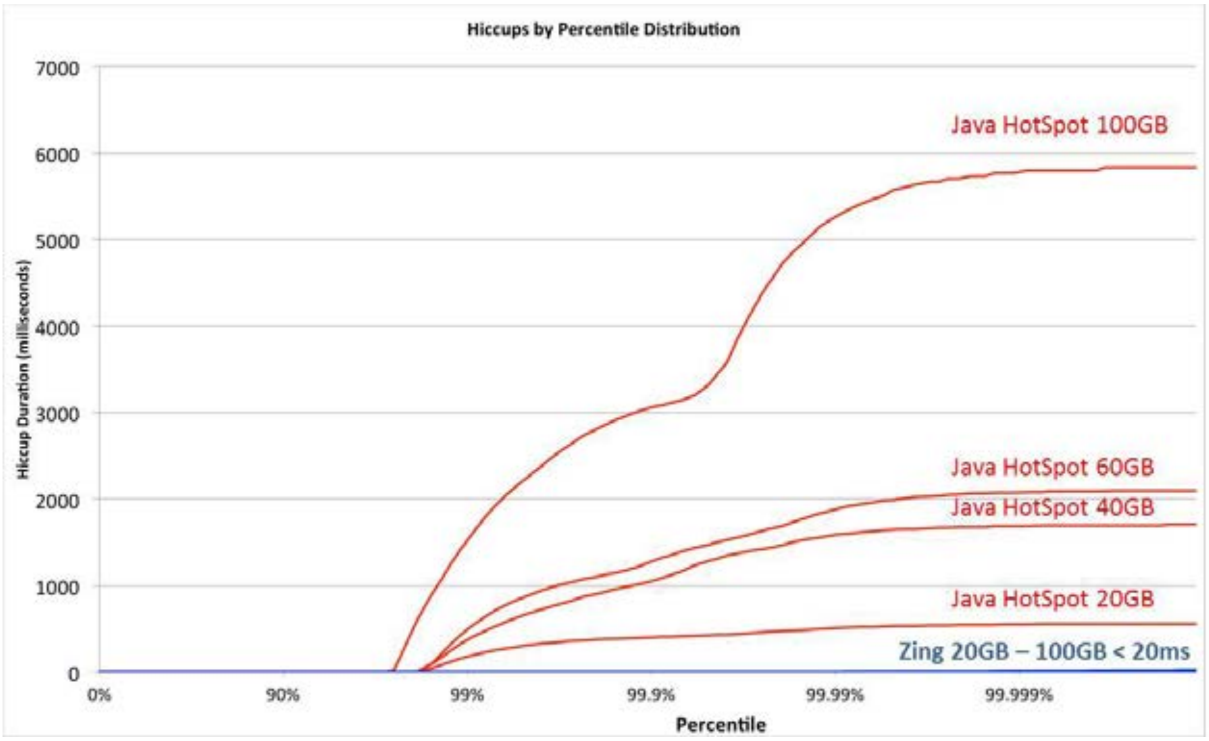


Figure 2 – Aggregated Hiccup Chart Showing Pauses in Milliseconds

From the graph in figure 2 Azul Zing shows highly consistent response time behavior across the entire benchmark spectrum. Whether at 20 GB or 200 GB, Zing response times were less than 30 milliseconds, with only a few outliers in the tails of the distributions as shown in Figure 3:

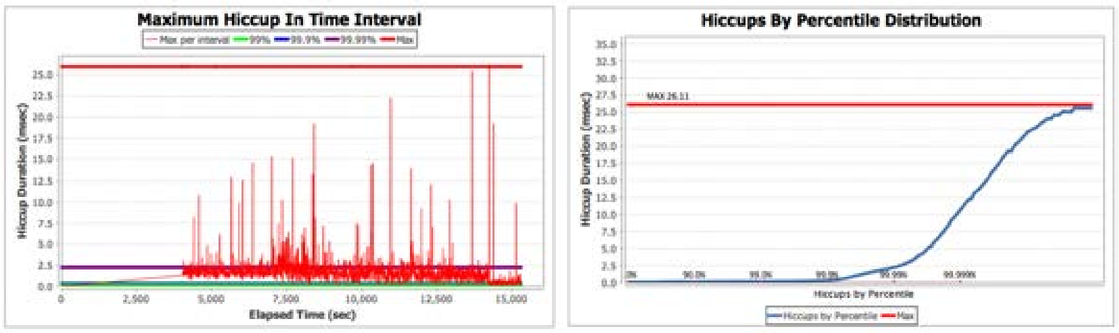


Figure 3 – Hiccup Chart for Azul Zing using 200 GB heap

When comparing the performance and consistency of the two JVMs, Azul Zing showed 22x better max outliers than Java HotSpot at 20 GB and more than a 250x advantage at 100 GB.

While response time profiles of the two JVMs where very different, the throughput of the date grid as measured by RadarGun were much closer, with only a slight advantage to Zing (about 1 – 3% across the nodes) over the short duration of this benchmark (i.e. 20-30 minutes).

6 Conclusion

This benchmark study demonstrates that application performance and runtime consistency can be greatly affected by the choice of JVM. Not all JVMs are the same and employ different GC algorithms that can impact application response time profiles. For applications that have explicit or even 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 that can meet deployment requirements.

When metrics such as “Sustained Throughput” (i.e. a throughput value which never exceeds a specific response time SLA), and time-to-production are important, Azul Zing provides better runtime metrics with less JVM tuning and heap resizing in order to avoid potentially devastating pauses.

For In-Memory Computing (IMC) solutions such as in-memory data grids (IMGDs), that benefit from larger Java heaps, Azul Zing provides a viable alternative with greater runtime consistency. When IMDGs are deployed with strict SLAs below 50 - 500 milliseconds, only Azul Zing will guarantee success. For IMDG configurations where node timeouts needs to be very low (e.g. under 50 milliseconds), only the Zing runtime can guarantee response times that will not cause nodes to go offline.

7 References

Red Hat JBoss Data Grid
http://www.redhat.com/products/jbossenterprisemiddleware/data-grid/

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

RadarGun Framework Tutorial on Github
https://github.com/radargun/radargun/wiki/Five-Minute-Tutorial

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	**Red Hat**
Phone: +1.650.230.6500	**Phone:** +1.888.REDHAT1
Email: info@azulsystems.com	**WebForm:** www.redhat.com/contact/sales.html
www.azulsystems.com	www.redhat.com

8 Appendix

8.1 Test Durations and Settings

RadarGun Benchmark Configuration – 20GB heap test	
Benchmark Duration	15 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	50,000
Transaction mix	67% read, 32% write, 1% remove
Number of threads	30

RadarGun Benchmark Configuration – 40GB heap test	
Benchmark Duration	15 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	50,000
Transaction mix	67% read, 32% write, 1% remove
Number of threads	60

RadarGun Benchmark Configuration – 60GB heap test	
Benchmark Duration	35 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	50,000
Transaction mix	67% read, 32% write, 1% remove
Number of threads	80

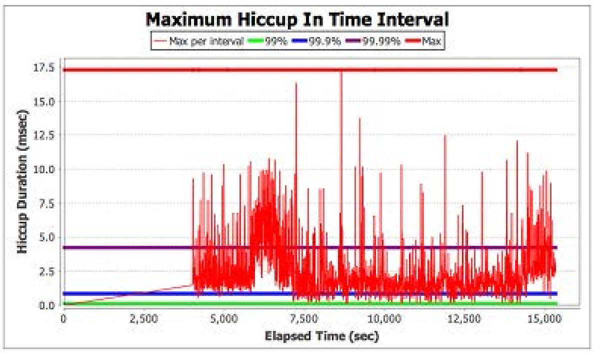
RadarGun Benchmark Configuration– 100GB heap test	
Benchmark Duration	60 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	100,000
Transaction mix	39% read, 60% write, 1% remove
Number of threads	100

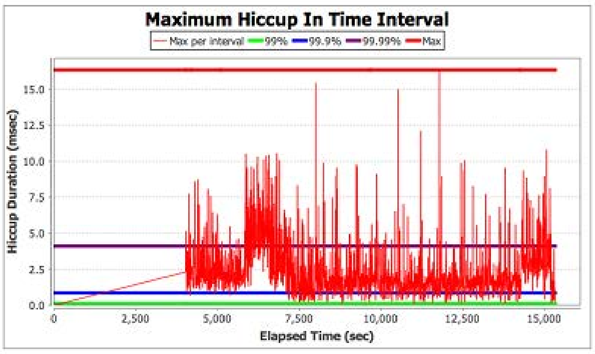
RadarGun Benchmark Configuration– 200GB heap test	
Benchmark Duration	120 minute runs
Message Payload Sizes	15% 1024 bytes, 35% 2048 bytes, 50% 4096 bytes
Mode of Data Grid Operation	Distributed
Heap Priming	35 – 45%
Transactions	200,000
Transaction mix	39% read, 60% write, 1% remove
Number of threads	100

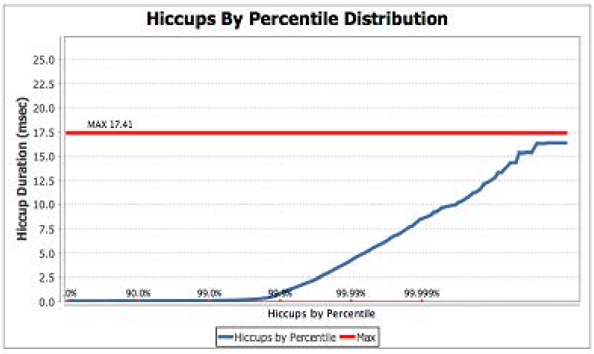
8.2 Hiccup Charts for All Runs

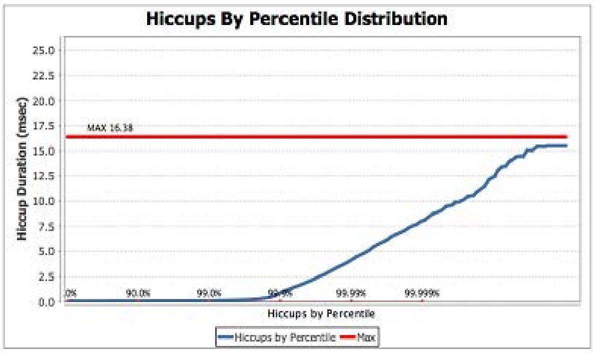
Note that the 4 graphs in each section represent 4 nodes, each with its own set of data.

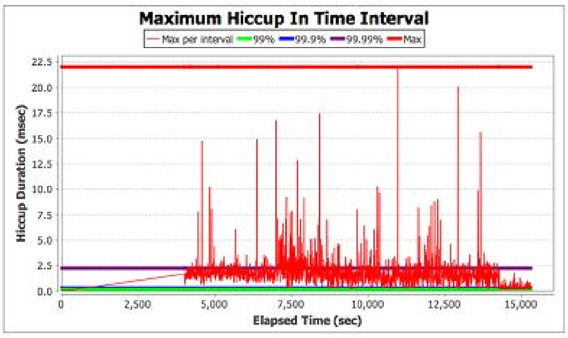
Zing 200 GB

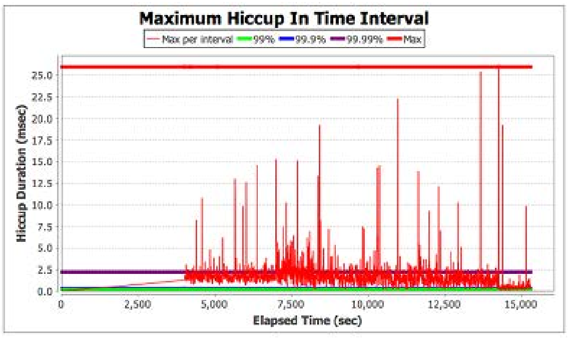


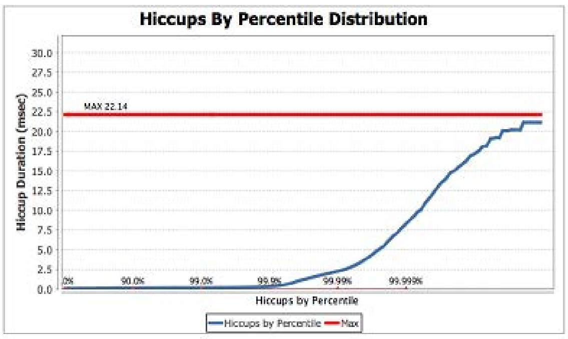


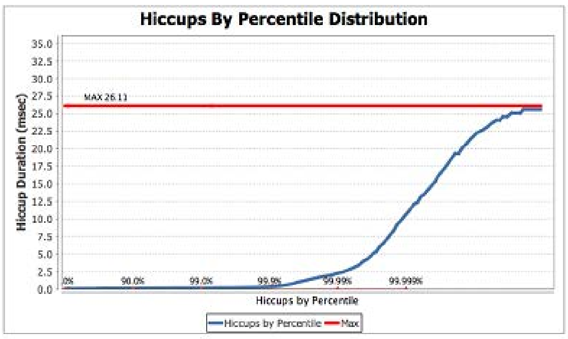




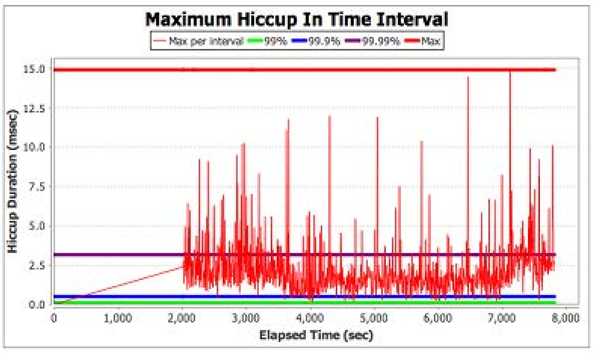


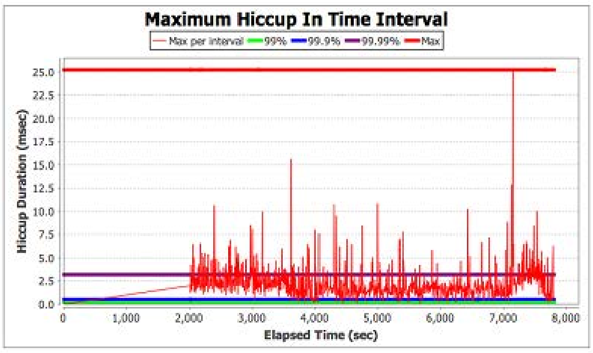


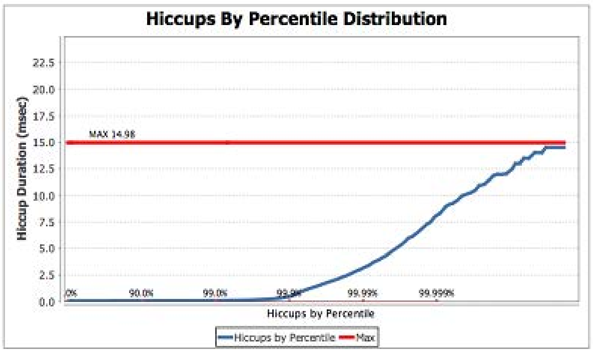


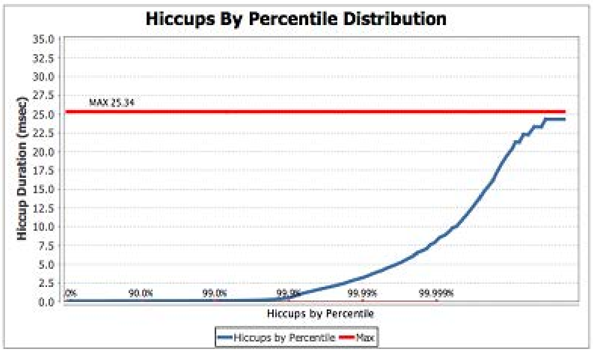


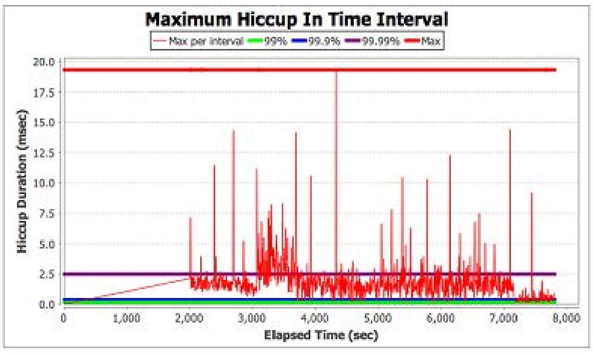
Zing 100 GB

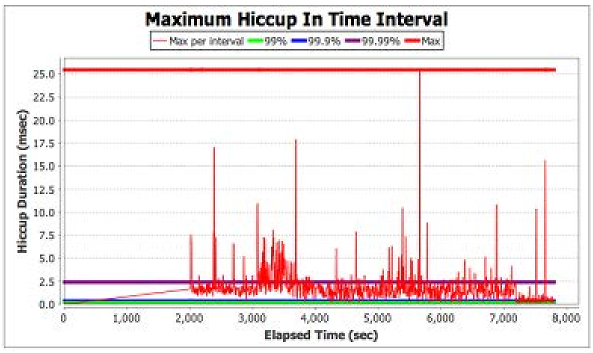

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

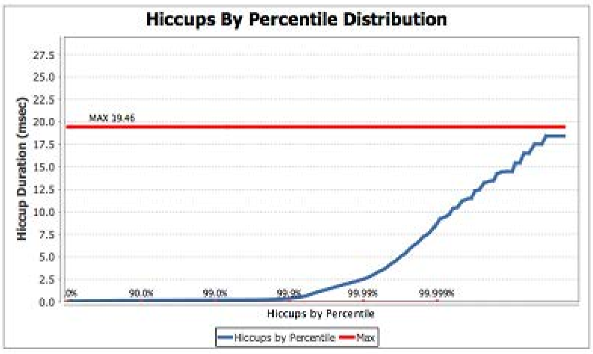

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

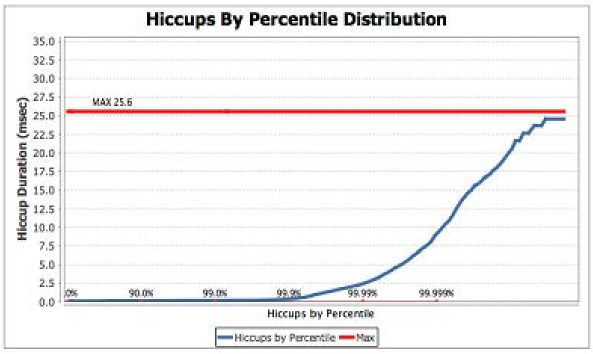

Hiccups By Percentile Distribution
MAX 14.98
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max


Hiccups By Percentile Distribution
MAX 25.34
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

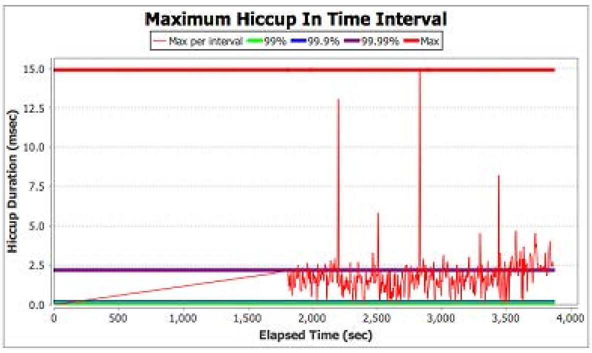

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

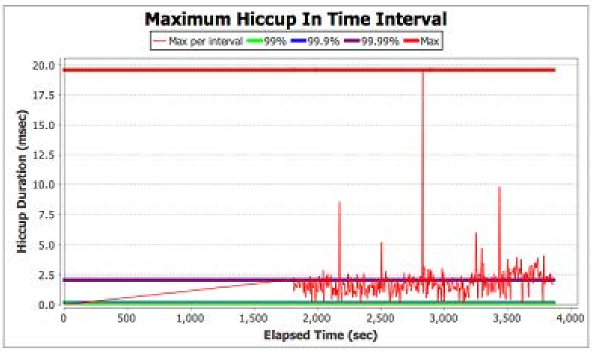

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

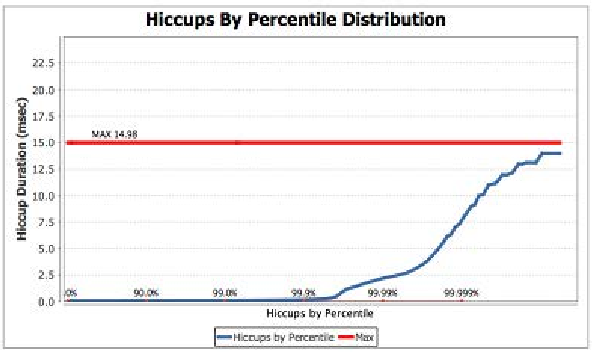

Hiccups By Percentile Distribution
MAX 19.46
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

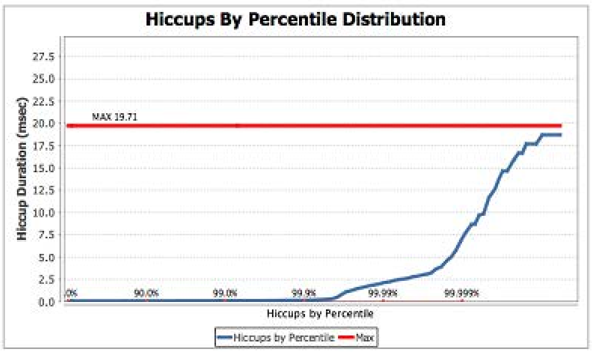

Hiccups By Percentile Distribution
MAX 25.6
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

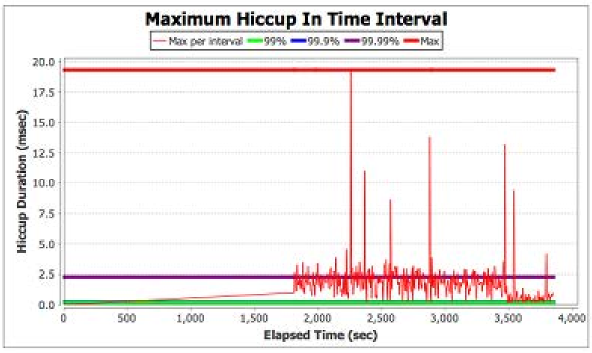
Zing 60 GB

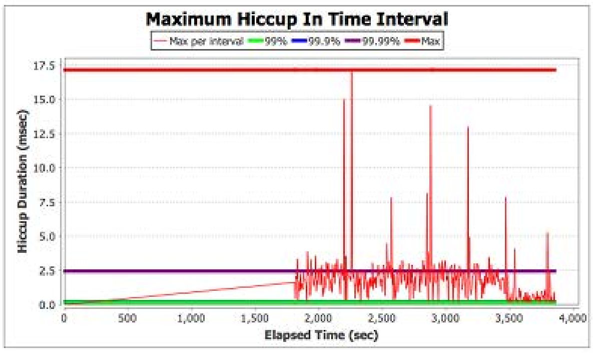

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

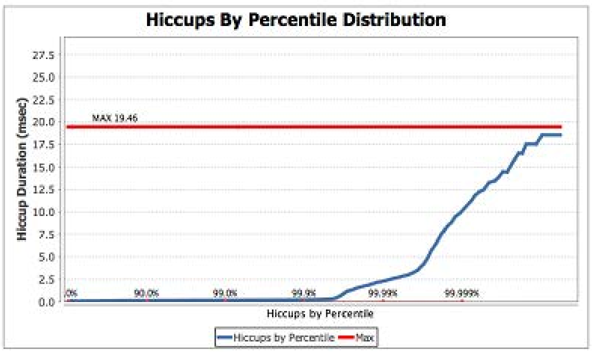

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

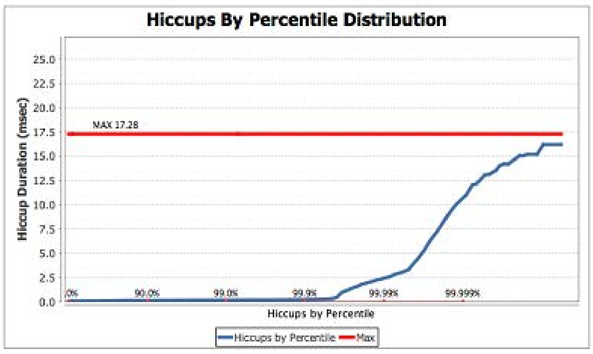

Hiccups By Percentile Distribution
MAX 14.98
Hiccup Duration (msec)
0%
90.0%
99.0%
99.9%
99.99%
99.999%
Hiccups by Percentile
Hiccups by Percentile
Max


Hiccups By Percentile Distribution
MAX 19.71
Hiccup Duration (msec)
0%
90.0%
99.0%
99.9%
99.99%
99.999%
Hiccups by Percentile
Hiccups by Percentile
Max

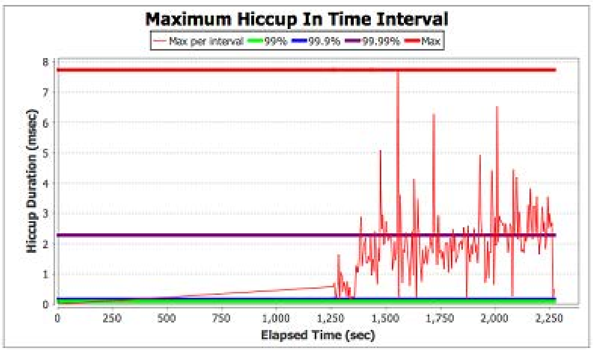

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

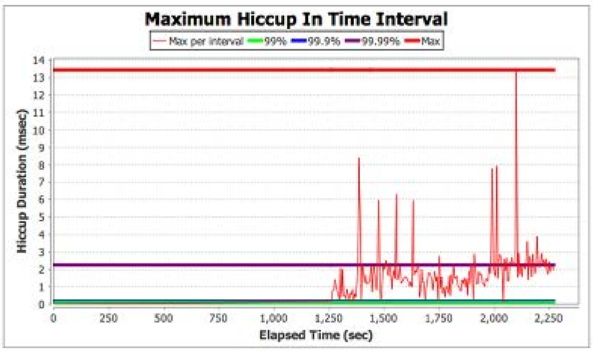

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

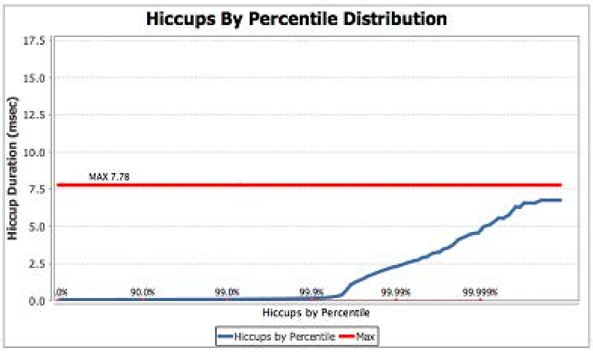

Hiccups By Percentile Distribution
MAX 19.46
Hiccup Duration (msec)
0%
90.0%
99.0%
99.9%
99.99%
99.999%
Hiccups by Percentile
Hiccups by Percentile
Max

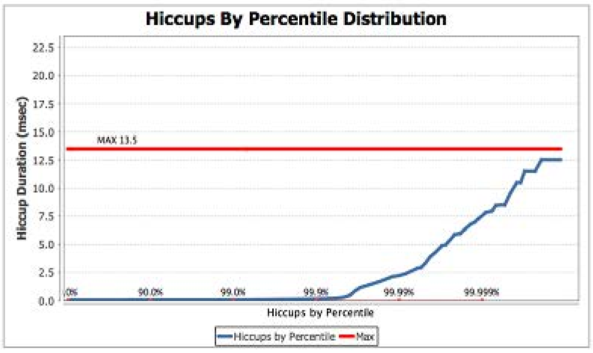

Hiccups By Percentile Distribution
MAX 17.28
Hiccup Duration (msec)
0%
90.0%
99.0%
99.9%
99.99%
99.999%
Hiccups by Percentile
Hiccups by Percentile
Max

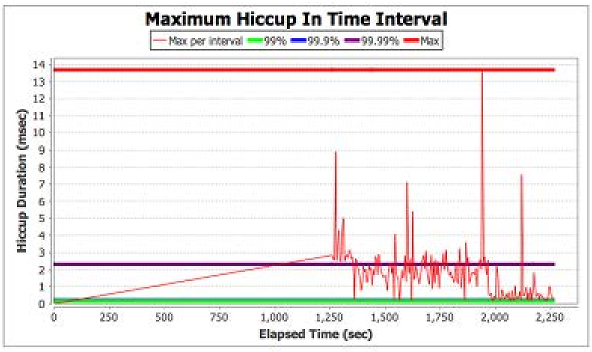
Zing 40 GB

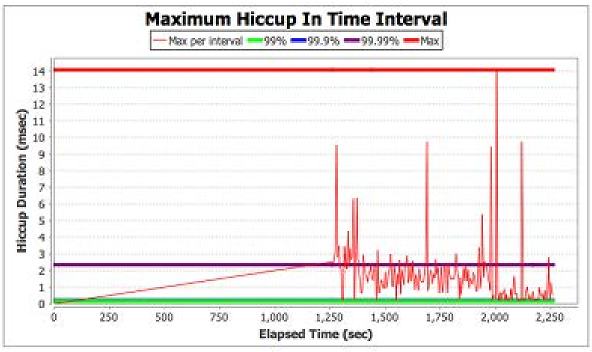

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

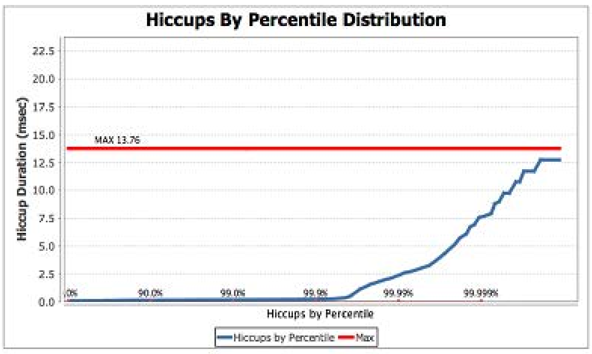

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

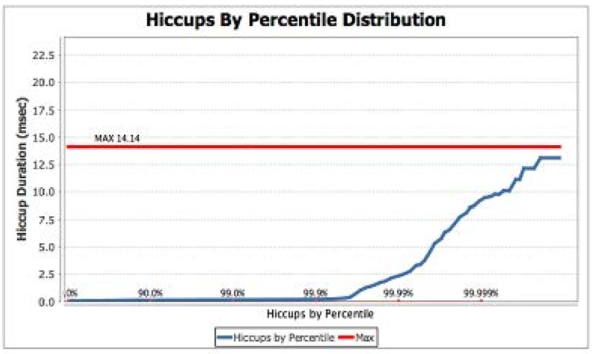

Hiccups By Percentile Distribution
MAX 7.78
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max


Hiccups By Percentile Distribution
MAX 13.5
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

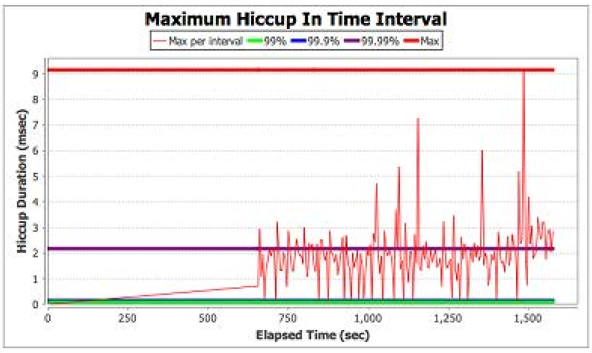

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

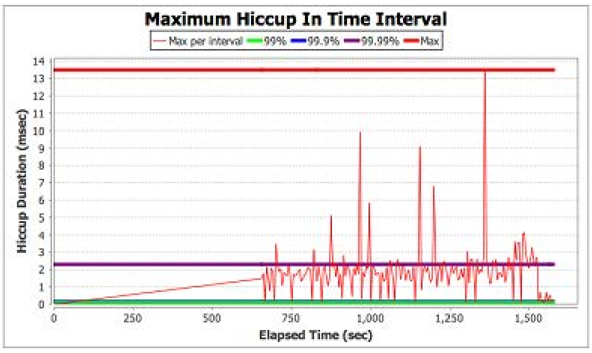

Maximum Hiccup In Time Interval
Max per interval
99%
99.9%
99.99%
Max
Hiccup Duration (msec)
Elapsed Time (sec)

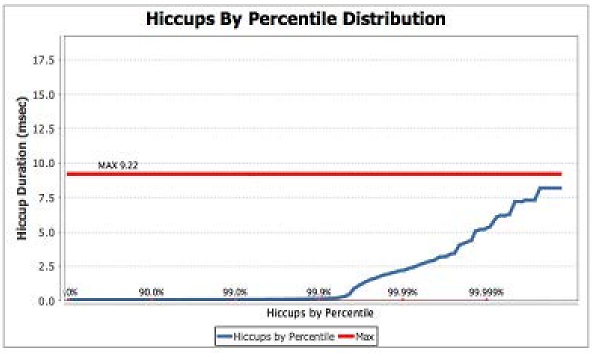

Hiccups By Percentile Distribution
MAX 13.76
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

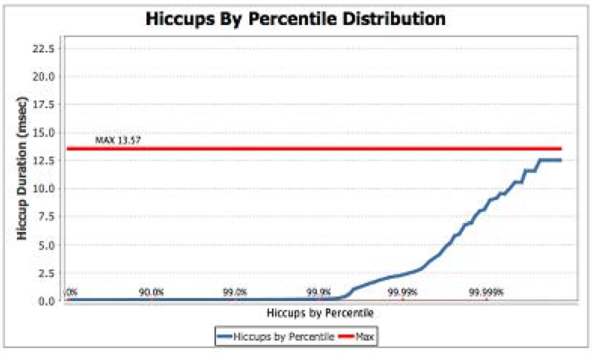

Hiccups By Percentile Distribution
MAX 14.14
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max

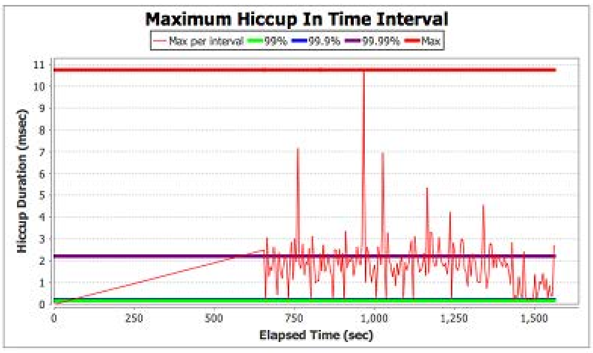
Zing 20 GB

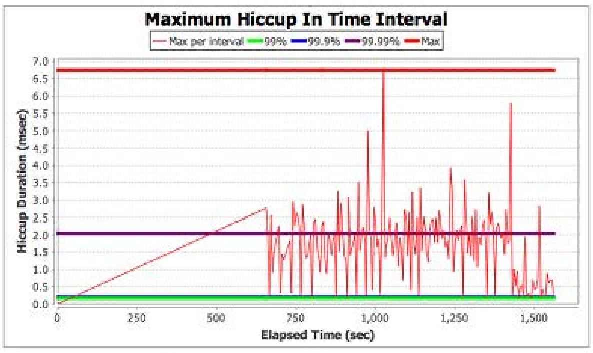

Maximum Hiccup In Time Interval
Max per interval 99% 99.9% 99.99% Max
Hiccup Duration (msec)
Elapsed Time (sec)

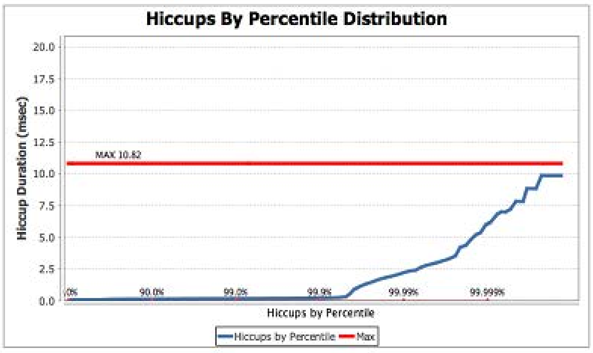

Maximum Hiccup In Time Interval
Max per interval 99% 99.9% 99.99% Max
Hiccup Duration (msec)
Elapsed Time (sec)

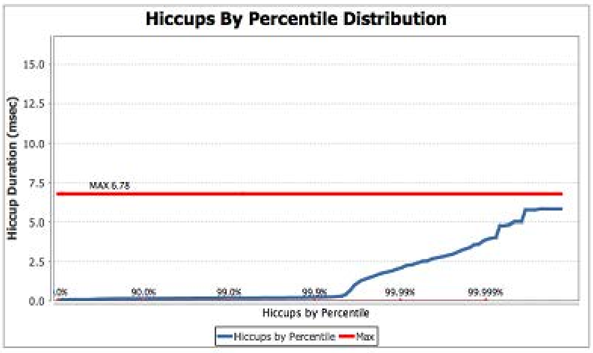

Hiccups By Percentile Distribution
MAX 9.22
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile Max


Hiccups By Percentile Distribution
MAX 13.57
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile Max

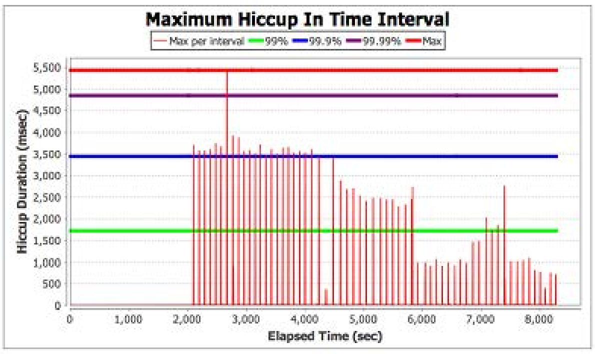

Maximum Hiccup In Time Interval
Max per interval 99% 99.9% 99.99% Max
Hiccup Duration (msec)
Elapsed Time (sec)

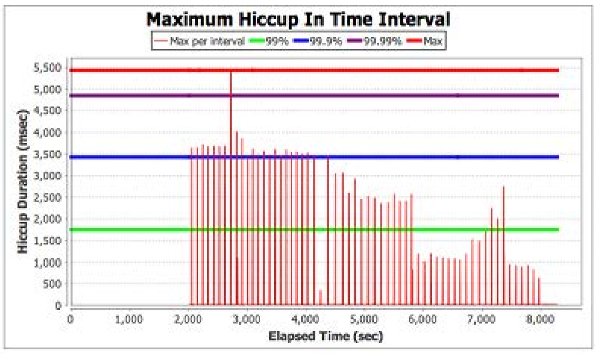

Maximum Hiccup In Time Interval
Max per interval 99% 99.9% 99.99% Max
Hiccup Duration (msec)
Elapsed Time (sec)

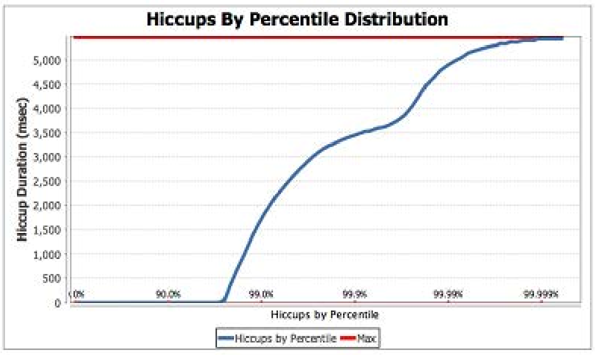

Hiccups By Percentile Distribution
MAX 10.82
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile Max

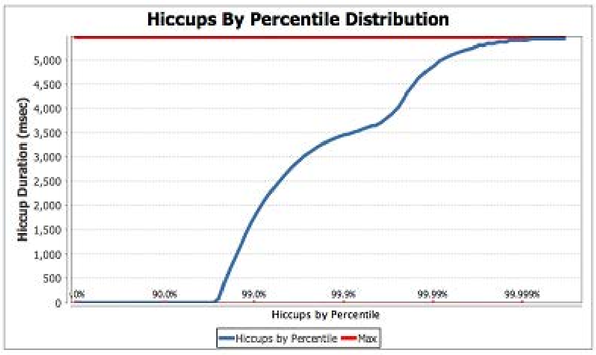

Hiccups By Percentile Distribution
MAX 6.78
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile Max

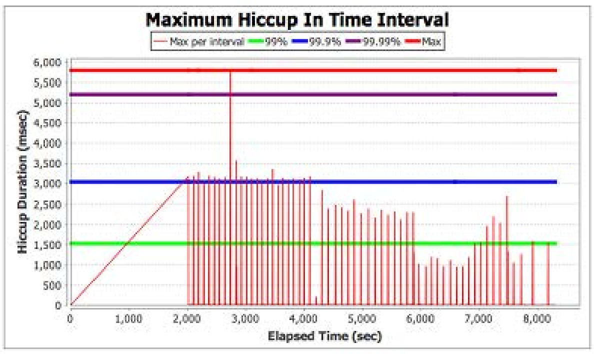
Java HotSpot 100 GB

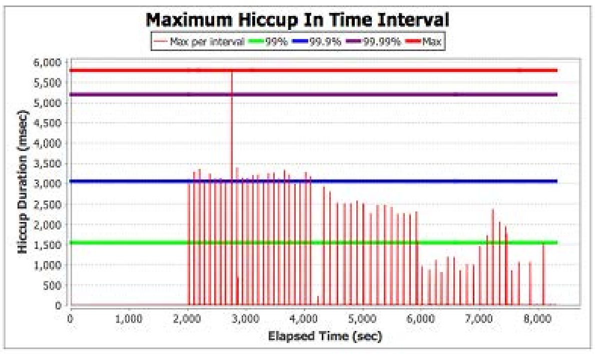


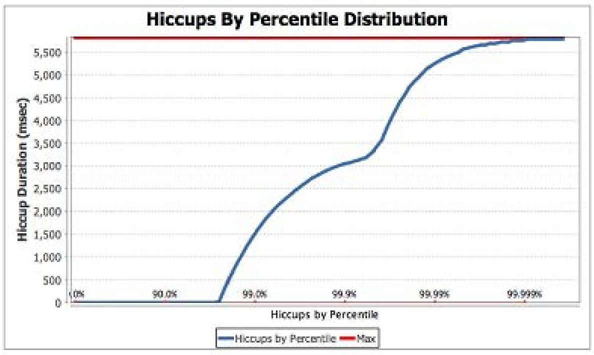


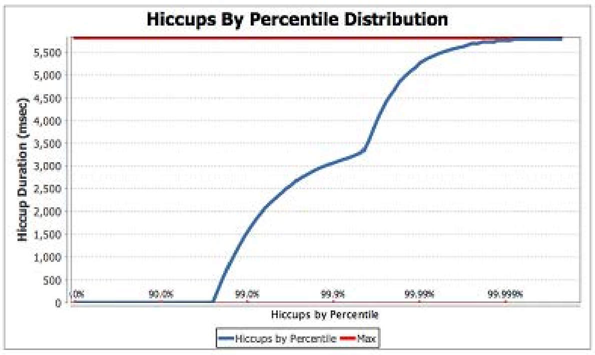




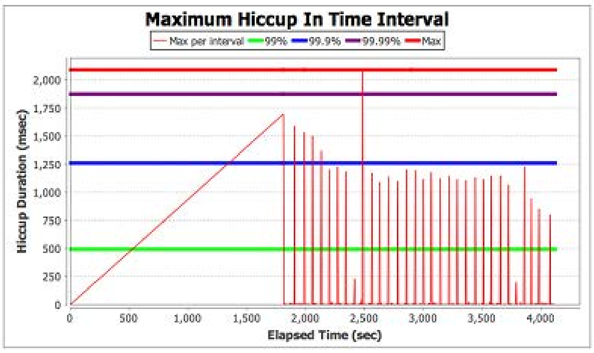


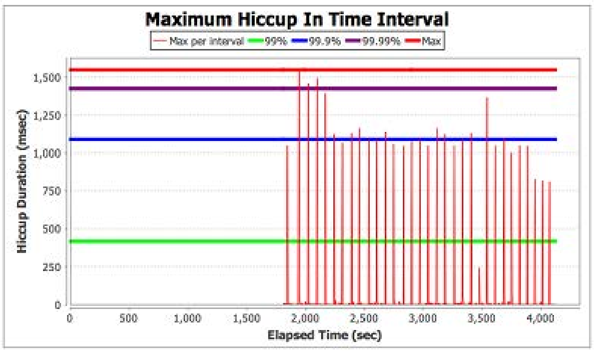


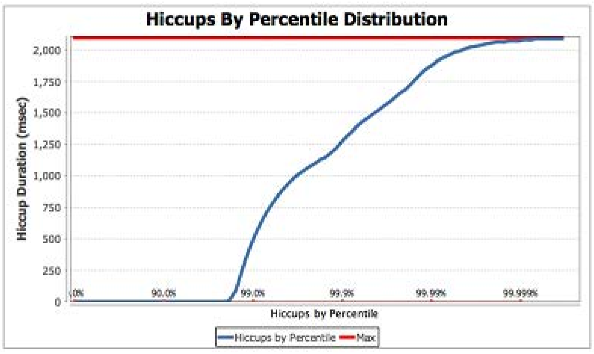


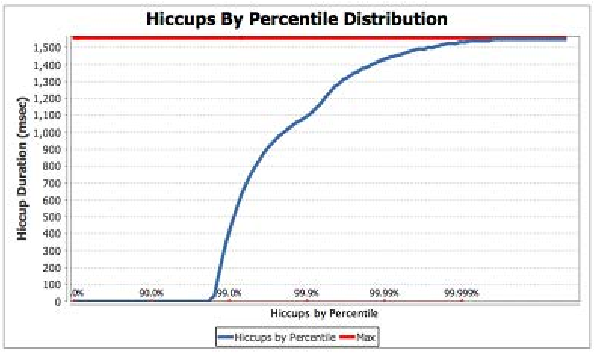


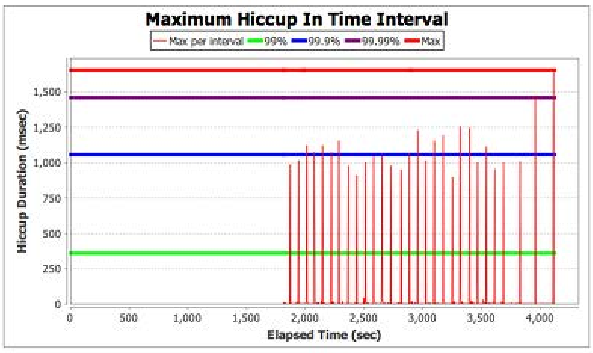
Java HotSpot 60 GB

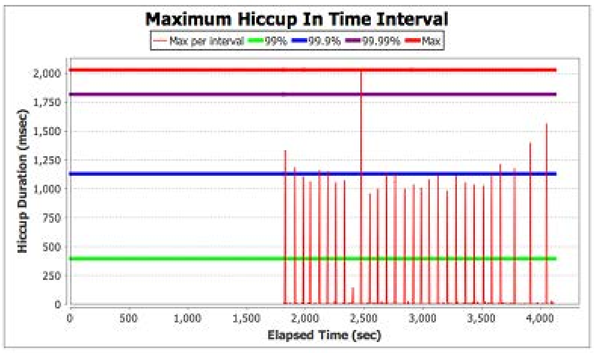


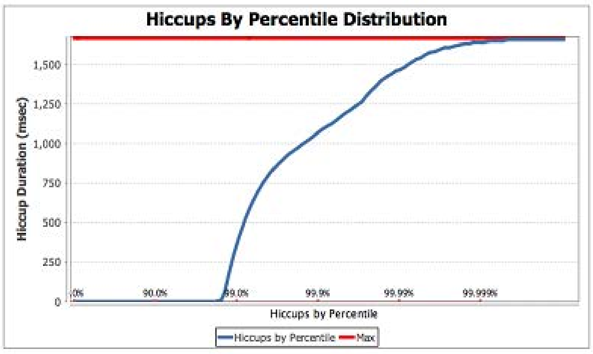


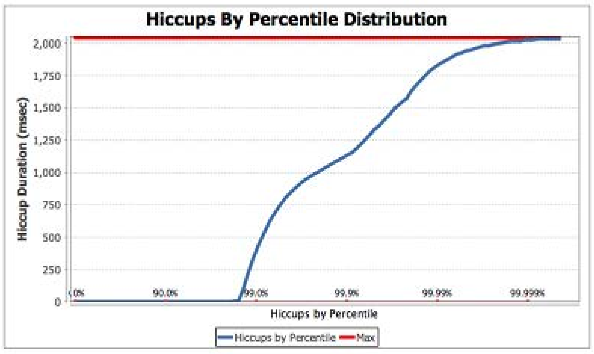




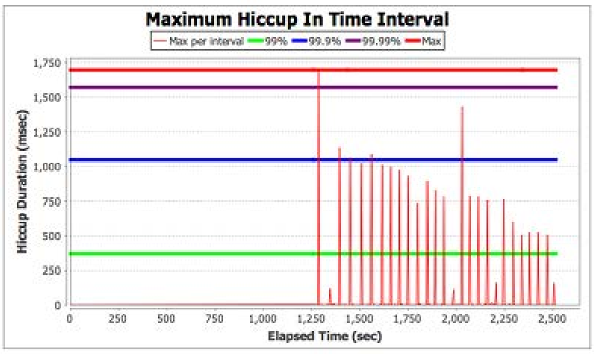


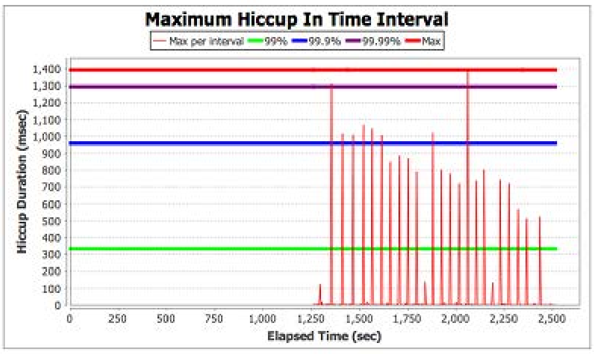


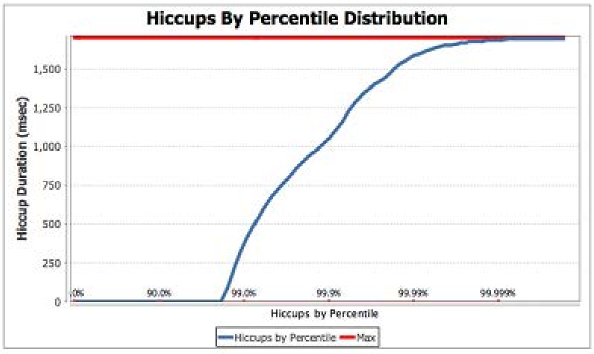


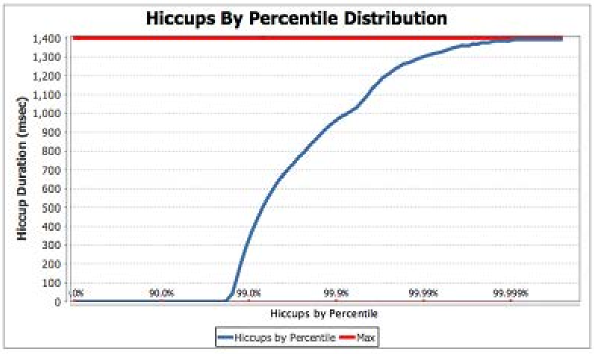


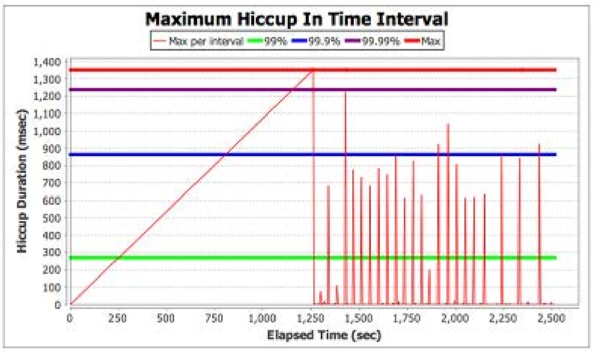
Java HotSpot 40 GB

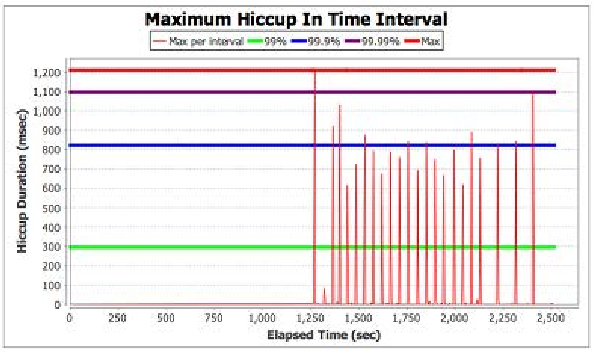


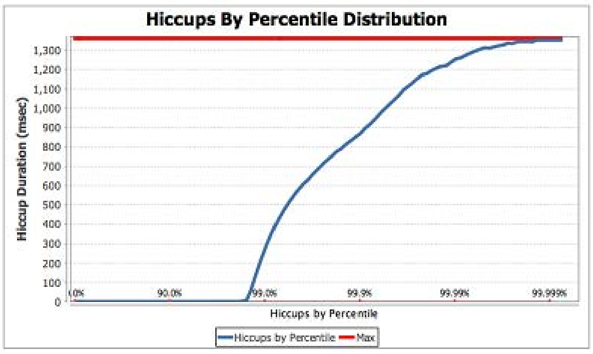


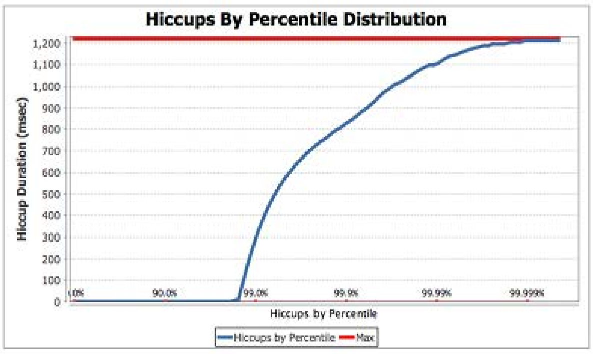




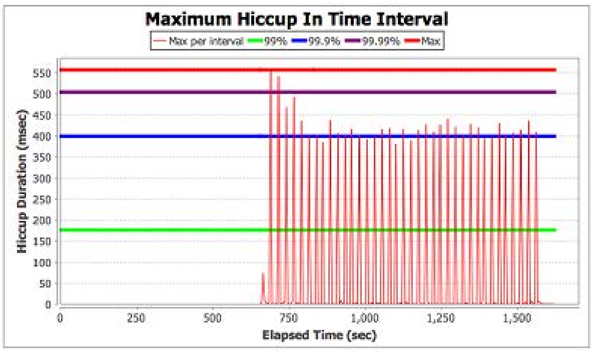


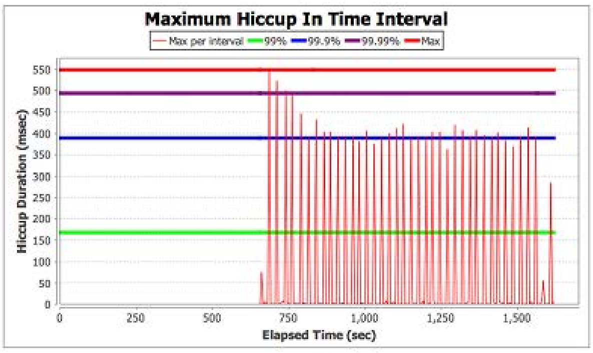


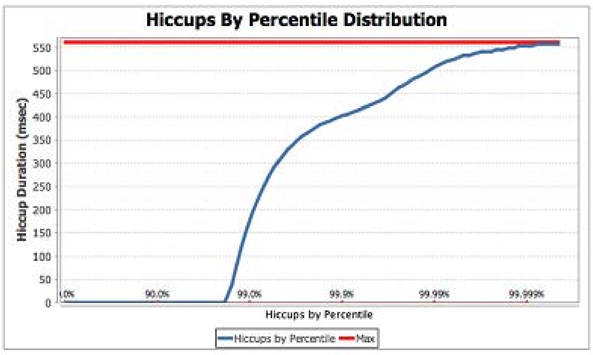


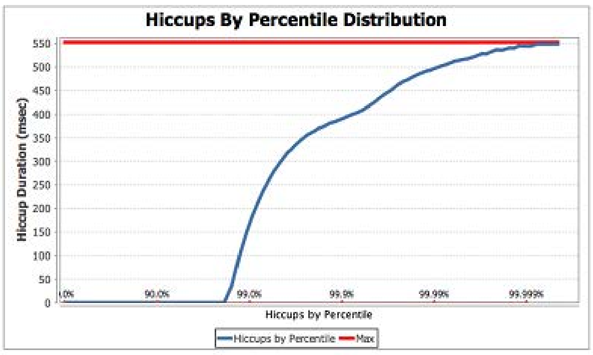


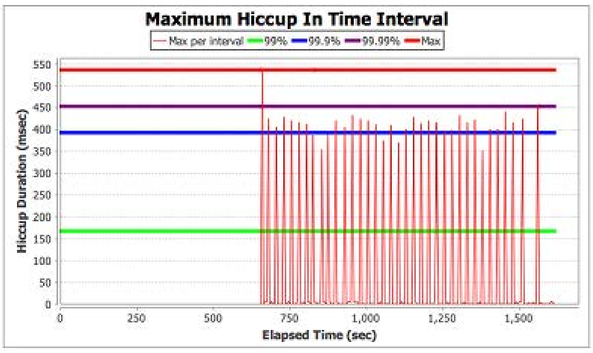
Java HotSpot 20 GB

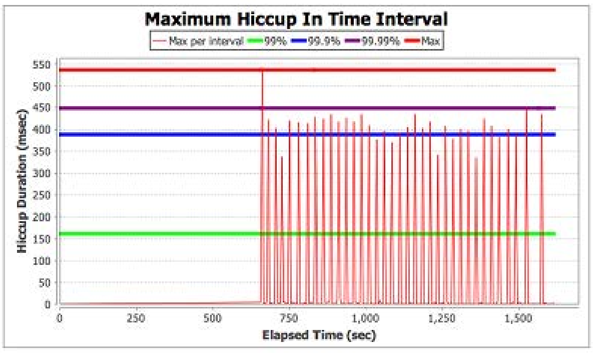

Maximum Hiccup In Time Interval
Max per interval
Max
Hiccup Duration (msec)
Elapsed Time (sec)

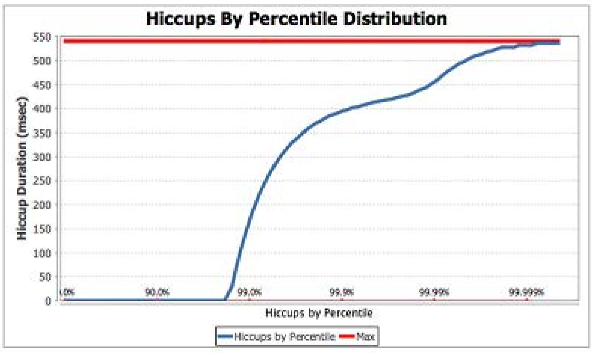

Maximum Hiccup In Time Interval
Max per interval
Max
Hiccup Duration (msec)
Elapsed Time (sec)

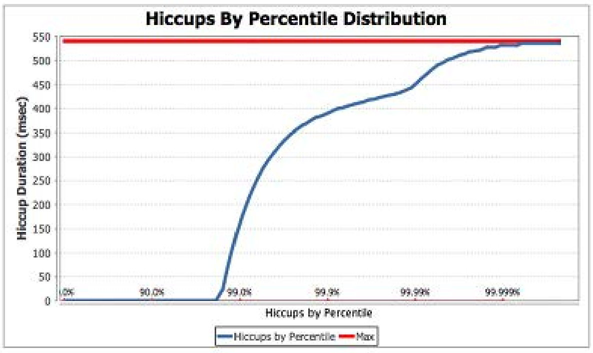

Hiccups By Percentile Distribution
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max


Hiccups By Percentile Distribution
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max


Maximum Hiccup In Time Interval
Max per interval
Max
Hiccup Duration (msec)
Elapsed Time (sec)


Maximum Hiccup In Time Interval
Max per interval
Max
Hiccup Duration (msec)
Elapsed Time (sec)


Hiccups By Percentile Distribution
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max


Hiccups By Percentile Distribution
Hiccup Duration (msec)
Hiccups by Percentile
Hiccups by Percentile
Max