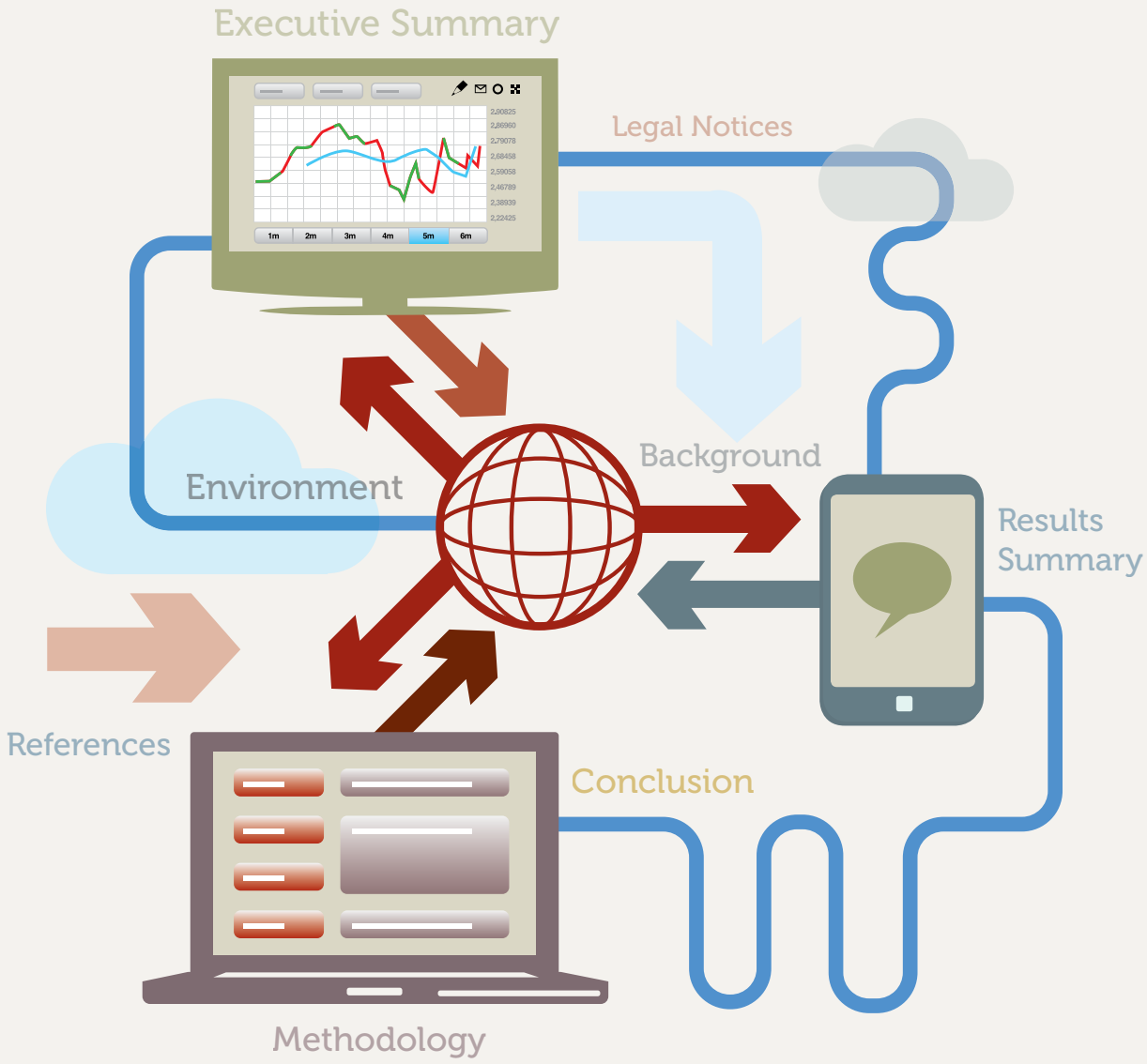
PUSH TECHNOLOGY™




# Diffusion™ Performance Study

## with Azul Systems Zing® JVM

## Legal Notices



## Contents

# 1. Executive Summary

**To compare the performance of Push Technology's Java based data distribution solution, Diffusion™, running on the Azul Systems Zing® JVM versus the standard HotSpot™ JVM, a benchmarking study was conducted in Spring/Summer 2013. A throughput benchmark with a stable working set representing a typical production load was chosen as the basis for comparison.**

Diffusion™ is a high performance data distribution technology with excellent latency and throughput characteristics, whether data distribution solutions are deployed behind or across the firewall. This study shows that by running Diffusion™ on Zing, application pauses in latency critical deployments can be effectively removed.

The combined solution delivers consistent high throughput and low latency with measurably less jitter – resulting in hugely reduced latency outliers on application messages. This is a significant advantage in environments where timeliness and responsiveness are mission critical concerns. Diffusion goes beyond traditional messaging offering fine grained control over data distribution to significantly reduce bandwidth utilisation without losing data, through features such as structural conflation of in-memory data that adapts to the bandwidth, fidelity and latency constraints of each connected device. In a nutshell, Diffusion can save up to 95% of the bandwidth compared to traditional messaging techniques. Azul's Zing makes sure the timeliness and responsiveness of the remaining 5% is delivered with stable, consistent low latency. The combination is unbeatable.

Azul's Zing® JVM is a full-featured JVM based on HotSpot™ and enhanced by Azul Systems to work in concert with the operating system to deliver consistent response times over a smooth, wide operating range. According to Azul Systems, CTO, Gil Tene, "with Zing, JVM jitter is completely unrelated to heap size or live set, and will remain the same whether your application uses 1GB or 300GB of heap. Latency sensitive applications can expect JVM jitter to be easily contained down to observed worst-case levels well below the 10 milliseconds mark. With simple Linux and hardware system tuning common in the low latency world, these worst observed case numbers often drop to 2-3 milliseconds. Some of the more aggressive system tuners that use Zing often demonstrate end-to-end application jitter levels with worst observed case levels of 1 millisecond or less over long lasting, sustained runs."

Particularly noteworthy for Zing are the 'stop-the-world' application pause times for garbage collection. Zing does its garbage collection concurrently in parallel with the running application – so it has zero 'stop-the-world' pauses for garbage collection. In comparison the HotSpot JVM demonstrated 'stop-the-world' application pause times for garbage collection of up to 115 milliseconds at the 99th percentile.

By reading this performance paper, you will understand the environment, methodology and results of the testing. The results speak for themselves.

## 2. Background

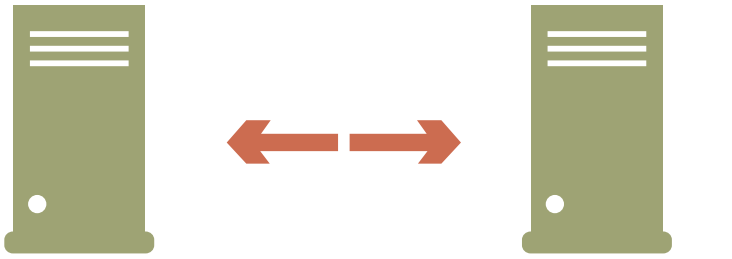
The benchmark results in this paper were produced from the Push Technology Diffusion™ open source benchmark suite.

(See 'Benchmark Suite on GitHub' in References section below.)

Diffusion servers are typically deployed at or behind the edge of private networks where services are exposed to public networks, with multiple instances running behind a load balancer. Diffusion is a real-time stream oriented distribution technology. It intelligently batches, fragments and conflates messages to optimize distribution whilst guaranteeing the currency, and timeliness of delivery.

## 3. Environment

The test environment consists of two machines. A four-socket server machine and a two-socket client machine. Both machines are configured with Hyper-Threading. The machines use Solarflare® 10GbE network interface cards and are directly connected.




| Machine Configuration | Diffusion™ Host Server |
|---|---|
| Manufacturer | Iron Systems |
| Processors (x 4) | Intel® Xeon® Processor E5-4620 (16M Cache, 2.20 GHz, 7.20 GT/s Intel® QPI) |
| Memory (x 32) | 16GB RDIMM, 1333MHz, Low Volt, Dual Rank |
| Networking | 1 x Solarflare Communications SFC9020 [Solarstorm] |
| OS | CentOS 6.3 – with Linux kernel: 2.6.32-279.el6.x86_64 |

| Machine Configuration | Benchmark Clients Host Server |
|---|---|
| Manufacturer | Iron Systems |
| Processors (x 2) | Intel® Xeon® Processor X5650<br>(12M Cache, 2.66 GHz, 6.40 GT/s Intel® QPI) |
| Memory (x 18) | 8GB RDIMM, 800MHz, Low Volt, Dual Rank |
| Networking | 1 x Solarflare Communications SFC9020 [Solarstorm] |
| OS | CentOS 6.4 – with Linux kernel: 2.6.32-358.el6.x86_64 |

The Diffusion™ server is deployed on the four-socket machine (the processor sockets are enumerated as 0, 1, 2, 3):

- For Zing® tests, the Diffusion™ server was constrained to processor sockets 2 and 3.
- For HotSpot tests, the Diffusion™ server was constrained to processor socket 1.

All clients are deployed on the two-socket machine. The clients are constrained to a single processor socket.

The client machines only use the HotSpot JVM and the configuration is the same for all test runs irrespective of server configuration. The only variables are the server JVM and the number of concurrent clients. The raw and processed results and logs are available. (See 'Captured Benchmark Data' in References section below).

**Note:** In the previous study "**Diffusion™ 4.4 Performance Benchmark**"[1]; servers and clients were constrained to a single processor on each machine to force saturated conditions.

| Java Configuration | JVM Version |
|---|---|
| Diffusion™ Benchmark Server | Azul Systems Zing®<br>java version "1.6.0_38-internal"<br>Zing Runtime Environment for Java Applications<br>(build 1.6.0_38-internal-b6)<br>Zing 64-Bit Tiered VM<br>(build 1.6.0_38-ZVM_5.6.1.0-b6-product-azlinuxM-X86_64, mixed mode) |
| Client Server | HotSpot<br>Java(TM) SE Runtime Environment (build 1.7.0_05-b06)<br>Java HotSpot(TM) 64-Bit Server VM (build 23.1-b03, mixed mode) |

The benchmarks were run with the machines directly interconnected using Solarflare Communications SFC9020 10GbE network cards. It is recommended to not exceed 30,000 concurrent connections with these cards. The Solarflare® OpenOnload® kernel bypass driver was not utilized in these test runs.

[1] *Diffusion™ 4.4 Performance Benchmark*
*http://www.pushtechnology.com/wp-content/uploads/2013/04/Diffusion_Perf_Whitepaper_Final04_13.pdf*

## 4. Methodology

**This technical benchmark was designed to determine the performance profile of a single running instance of a server at a stable production load.**

A single server instance can easily saturate multiple 10GbE network interfaces on commodity hardware. Saturating critical resources such as network IO, CPU or memory resources is not a recommended practice outside of lab or benchmarking conditions as service levels degrade beyond this point.

This benchmark configuration was run well within resource limits with no resource saturation.

Diffusion supports multiple transports. The most widely deployed transport carrying the Diffusion protocol over the web and mobile today is the IETF WebSocket[2] protocol. This is the transport documented in this whitepaper.

The same configuration of Diffusion™ was used for each benchmark run with the Zing™ and HotSpot™ JVMs. The only significant configuration differences from the out of the box default configuration of Diffusion™ were: tuning the number of client multiplexers to the number of CPU cores on the server machine; conflation was disabled; transport connectors not in use during benchmark runs were disabled.

Both the Zing and HotSpot JVMs were configured to maximize throughput and minimize pauses and delays. Configuration is sensitive to working set size, payload size and configuration of the benchmark suite. The benchmark configuration has been saved as a test suite in the open source benchmark suite. This allowed results on suitably configured hardware to be easily replicated.

This throughput benchmark model was designed around a number of configurable dimensions.

The test was run repeatedly from a cold start of a Diffusion server instance and for a fixed duration of 20 minutes. Each benchmark run used the same message payload size of 125 bytes.

Concurrent client connections were set to a fixed number for each test run. The number of concurrent connections were 1000, 5000, 10,000 and 15,000. Each set of runs was repeated with the Zing and HotSpot JVM technologies.

Data was published at a designed rate of 200 messages per second per client.

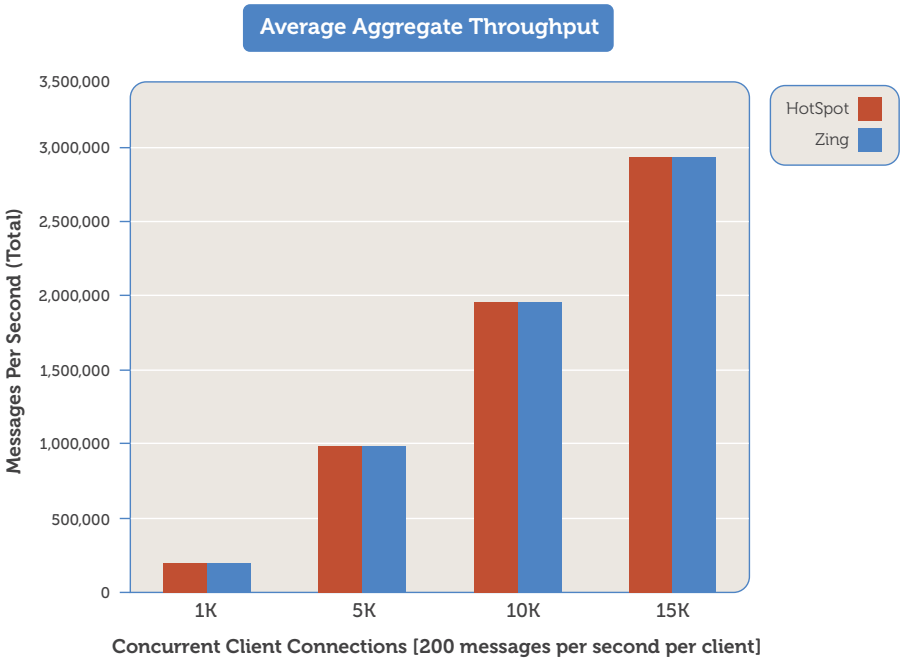
The client JVM for all test runs was the standard HotSpot JVM using CMS garbage collection.

[2] *IETF WebSocket Specification - http://www.rfc-editor.org/rfc/rfc6455.txt*

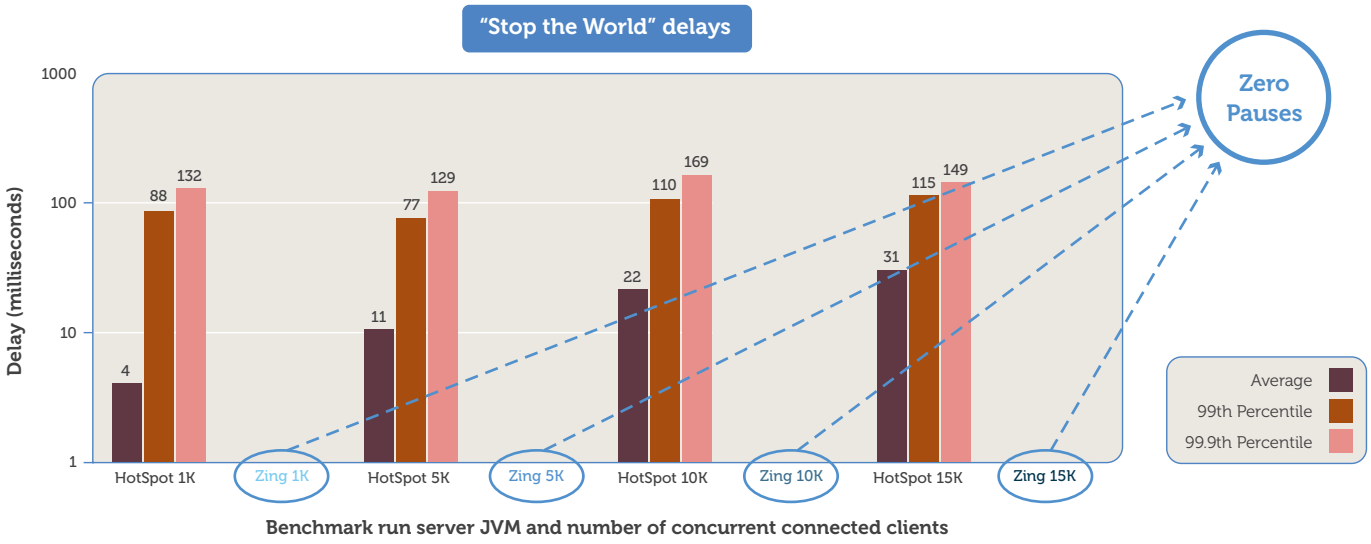
| Benchmark Configuration | |
|---|---|
| Benchmark Duration | 20 minute runs |
| Message Payload Sizes | 125 bytes |
| Mode of Operation | Without conflation |
| Ramping Interval | No ramping |
| Initial Clients | 1K, 5K, 10K and 15K clients |
| New Client Connections Per Interval | No ramping clients |
| Messages Per Second Per Client | 200 messages |

## 5. Results Summary

Given the designed rate of 200 messages per second per client for 1000, 5000, 10,000 and 15,000 client connections, we expect to achieve an aggregate throughput of 200,000, 1,000,000, 2,000,000 and 3,000,000 messages per second total. This was easily achieved by both JVMs. (Zing had a very slightly higher throughput.)



When viewed in isolation of other measurements it could be said that throughput of the two JVMs are pretty much equivalent; but they are not the same at all when viewed alongside other performance measurements taken during the test run. Analyzing garbage collection logs for 'stop the world' application pause times reveals a clear advantage when running Zing for latency critical environments.

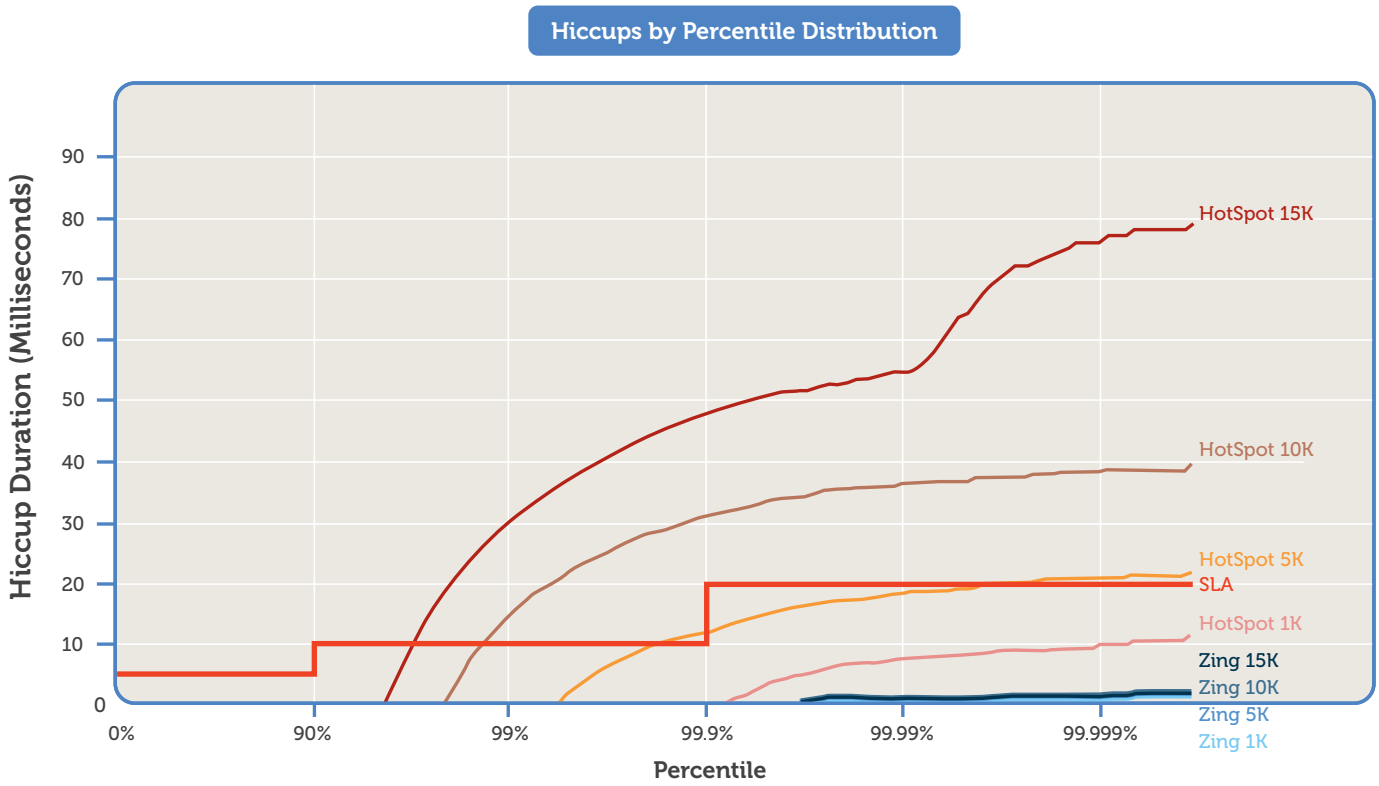



In the Zing case, there are no application pauses delaying business logic. In the HotSpot case, delays of up to 115 milliseconds were present at the 99th percentile. In application domains where latency predictability is important this is highly significant and noteworthy. In interactive environments, delays of greater than 40 milliseconds are noticeable, delays of 100 milliseconds are frustrating, and delays of more than 100 milliseconds are intolerable[3] .

End-to-end results in a real world system will include pauses in other components, services and networking infrastructure. It is important to note that the aggregate delay is what is perceptible to humans. Of course, machine-to-machine communication in latency critical environments is far more sensitive to aggregated delay.

Another view of delays, pauses and other systemic artefacts on the server indicating the health of the system under load can be generated using the open source jHiccup tool. The graph below is a composite of individual jHiccup charts generated from eight test runs.

[3] *Latency is everywhere and it costs you sales – How to crush it*
*http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it*




SLA = 5 msec for 90% | 10 msec for 99.99% | 20 msec for 100%

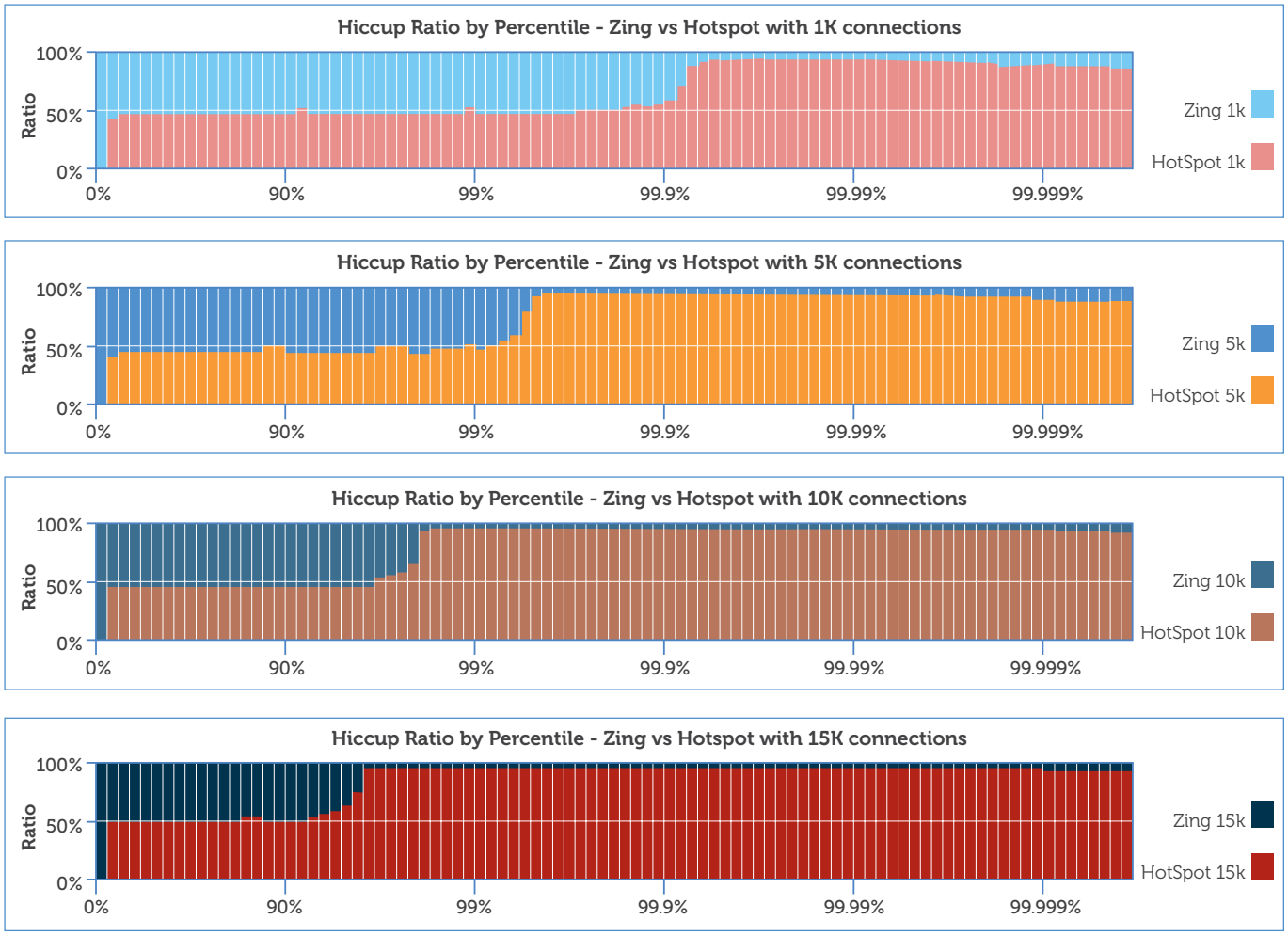
From this it is clear, that from the 95th percentile HotSpot begins to show signs of stress in excess of 1 millisecond. Azul Systems' Zing on the other hand, doesn't exhibit any significant stress until well into the 99.99th percentile and never exceeds a level that would be noticeable by a human.

Given that most Diffusion services are machine to human, interactive and near real-time there is clear value in choosing Azul Systems Zing where client experiences must be excellent at all times, such as trading, gaming, gambling and other near real-time systems with large numbers of concurrent interacting participants expecting a consistent predictable performance profile.

However, you will notice that the 'Hiccups by Percentile Distribution' chart above has an additional and highly significant addition – it has had a reasonable business service level requirement (or SLA) plotted onto the chart:

| SLA | |
|---|---|
| 5 msec | for 90% |
| 10 msec | for 99.9% |
| 20 msec | for 100% |

From the plotted SLA line it is clear that both HotSpot and Zing meet required business service levels with 1000 concurrent connections. However, with 5000 connections HotSpot breaks defined service levels. Zing on the other hand easily meets that specific SLA (even at 15,000 connections) and could in fact be tuned and operated more aggressively or the SLA could be significantly tightened.




Proportionately the above charts show a factor of up to 60x improvements in system health favouring Azul Systems Zing across all test runs.

These charts simply show the sum of hiccups for HotSpot and Zing at each percentile, colour coded by JVM. It can be interpreted in a similar way to a pie chart partitioned by percentile – in these graphs the greater the coloured area for a JVM is, then the larger its proportion is compared to the other JVM.

Through this visualisation it can be seen that up to the 95th percentile, the relative health of Zing and HotSpot are about equal.

- But beyond the 95th percentile, as the number of concurrent connections increases the ratio between the colours shifts dramatically, with HotSpot showing a much larger proportion than Zing; indicating that the HotSpot system experiences much more stress than the same working set running under Zing.
- At the 99th percentile, Zing is up to 29x improved compared to HotSpot.

The HotSpot JVM clearly exhibits increasing magnitudes of stress as the load (number of concurrent client connections) increases – at 15,000 connections, HotSpot exhibits a maximum "worst case" hiccup of 78.85 milliseconds. In comparison at the same load level, Zing exhibits stable and consistent delay free operation with a maximum "worst case" hiccup of 1.26 milliseconds.

jHiccup is a very useful tool for highlighting overall system health. It captures the aggregate affects of operating system, hardware and other stalls and pauses. It allows developers to identify the root cause of pauses pinning them to a running application or to the system the application is running upon.

## 6. Conclusion

Diffusion delivers consistent application throughput performance to large numbers of concurrent clients on HotSpot or Zing JVMs. However, when latency is a critical component it is clearly shown that a well-tuned Zing runtime outperforms a well-tuned HotSpot.

This study's test workload is typical of a production installation of Diffusion. Zing was seen to deliver zero application delays for up to 15,000 concurrently connected clients each receiving 200 messages per second per client of 125 bytes. The study did not uncover any disadvantage with respect to a choice of Zing rather than HotSpot.

The HotSpot JVM clearly exhibited increasing magnitudes of stress as the load (number of concurrent client connections) increased; whereas at the same load levels, Zing exhibited stable and consistent delay free operation across the board.

In addition to which, when a fairly reasonable SLA is imposed it can be seen that within that specific SLA, Zing delivers three to fifteen times the throughput with stable, highly consistent results.

The unique integration of messaging with in-memory data caching and in-flight data processing allows consistent, current data to be delivered to tens of thousands of client devices with optimal utilization. This study demonstrates that in concert with Zing's pedigree JVM, Diffusion enhances performance and improves overall system health under production loads and working sets.

Azul Systems' comprehensive set of in-built performance monitoring tools and accurate reporting of JVM internals such as 'time to safepoint', aid in tuning latency sensitive systems to demanding service levels not possible with HotSpot.

**If latency matters? ... "Zing it"**

## 7. References

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

**Diffusion 4.4 Performance Benchmark Whitepaper**
http://www.pushtechnology.com/wp-content/uploads/2013/01/Diffusion-Benchmarks.pdf

**Captured Benchmark Results**
http://more.pushtechnology.com/hs-fs/hub/188121/file-300842834-gz/azul-performance-study-whitepaper.tar.gz

**Benchmark Suite on GitHub**
https://github.com/pushtechnology/diffusion-benchmark-suite/tree/wp-asz-1.0

**JHiccup page**
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

**Azul Systems ZVision™**
http://www.azulsystems.com/technology/zing-vision

## About Push Technology

Push Technology solves the complexity around data distribution by removing redundant data to offer organisations intelligent delivery of real-time data to any device regardless of connectivity or location. The Company's robust and innovative flagship communication platform, Diffusion™ helps to reduce infrastructure requirements while delivering high performance and scalable services to any Internet connected device.

Push Technology works with organisations in the e-gaming, financial services, telecommunications, media and broadcast and transportation sectors to optimise data, mobile application performance, web scale and data acceleration. Delivering data that's live to the millisecond, Push Technology ensures that businesses can deliver engaging real-time customer experiences to drive revenues, increase competitiveness, develop new business models to reduce network strain and recover costs and also elevate consumer engagement across multiple channels in real time.

Customers include bet365, Betfair, Betdaq, Compliant Phones, ICAP, Lloyds Bank, Oddschecker, Racing Post, Sportingbet, Tradition and William Hill. For more information visit www.pushtechnology.com.

**For further information**
Visit **www.pushtechnology.com** or contact **sales@pushtechnology.com**

PUSH TECHNOLOGY LIMITED
3RD FLOOR HOLLAND HOUSE
1-4 BURY STREET LONDON EC3A 5AW
Telephone: +44 (0)203 588 0900

340 MADISON AVE, NY, NY 10173
Telephone: +1 201-978-5574

Twitter: @push_technology
www.pushtechnology.com