Cassandra in Response Time Sensitive Environments

Gil Tene, CTO & co-Founder, Azul Systems
@giltene
About me: Gil Tene

- Co-founder, CTO @Azul Systems
- Have been working on “think different” GC approaches since 2002
- A long history building Virtual & Physical Machines, Operating Systems, Enterprise apps, etc...
- I also depress people by demonstrating how terribly wrong their latency measurements are...

* Working on real-world trash compaction issues, circa 2004
Azul Systems

We build Java Virtual Machines

Powering mission-critical Java applications for Global 2000+

Deep expertise with latency-sensitive applications

from human sensitivity to application responsiveness (fractions of a second)

to low latency trading systems (fractions of a msec)

Cassandra is one of our common deployment scenarios
Zing Overview
A JVM for Linux/x86 servers

Delivers a continuously responsive execution platform

ELIMINATES Garbage Collection as a concern for enterprise applications

Very wide operating range:

Used in everything from low latency to huge in-memory apps

1GB to 1TB Heaps. 10MB/sec to 20GB/sec allocation rates.

Combats Execution inconsistencies of all types

Not just GC: Anything that makes a JVM glitch or slow down

“Not just Fast. Always Fast.”
What is Zing good for?

- If you have a server-based Java application
- And you are running on Linux (x86)
- And you use using more than ~300MB of memory

- Then Zing will likely deliver superior behavior metrics
Where Zing shines

★ Low latency
  ◆ Eliminate behavior blips down to the sub-millisecond-units level

★ Machine-to-machine “stuff”
  ◆ Support higher *sustainable* throughput (the one that meets SLAs)

★ Human response times
  ◆ Eliminate user-annoying response time blips. Multi-second and even fraction-of-a-second blips will be completely gone.
  ◆ Support larger memory JVMs *if needed* (e.g. larger virtual user counts, or larger cache, in-memory state, or consolidating multiple instances)

★ “Large” data and in-memory analytics
  ◆ Make batch stuff “business real time”. Gain super-efficiencies.
Why Zing?
Oracle HotSpot CMS, 1GB in an 8GB heap

Hiccups by Time Interval

Hiccup Duration (msec)

Elapsed Time (sec)

Max per Interval 99% 99.90% 99.99% Max

Max=13156.352

Hiccups by Percentile Distribution

Max=13156.352

Percentile

Zing 5, 1GB in an 8GB heap

Hiccups by Time Interval

Hiccup Duration (msec)

Elapsed Time (sec)

Max per Interval 99% 99.90% 99.99% Max

Max=20.384

Hiccups by Percentile Distribution

Max=20.384

Percentile
Oracle HotSpot CMS, 1GB in an 8GB heap

Zing 5, 1GB in an 8GB heap

Drawn to scale
Sustainable Throughput: The throughput achieved while safely maintaining service levels
Percentiles Matter
Is the 99%’ile “rare”?
Cumulative probability...

What are the chances of a single web page view experiencing the 99%’ile latency of:

- A single search engine node?
- A single Key/Value store node?
- A single Database node?
- A single CDN request?
| Site                        | # of requests | page loads that would experience the 99%'ile 
\[(1 - (.99 ^ N)) * 100\%\] |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>190</td>
<td>85.2%</td>
</tr>
<tr>
<td>kohls.com</td>
<td>204</td>
<td>87.1%</td>
</tr>
<tr>
<td>jcrew.com</td>
<td>112</td>
<td>67.6%</td>
</tr>
<tr>
<td>saksfifthavenue.com</td>
<td>109</td>
<td>66.5%</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>nytimes.com</td>
<td>173</td>
<td>82.4%</td>
</tr>
<tr>
<td>cnn.com</td>
<td>279</td>
<td>93.9%</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>twitter.com</td>
<td>87</td>
<td>58.3%</td>
</tr>
<tr>
<td>pinterest.com</td>
<td>84</td>
<td>57.0%</td>
</tr>
<tr>
<td>facebook.com</td>
<td>178</td>
<td>83.3%</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>google.com</td>
<td>31</td>
<td>26.7%</td>
</tr>
<tr>
<td>(yes, that simple noise-free page)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>google.com</td>
<td>76</td>
<td>53.4%</td>
</tr>
</tbody>
</table>
Which HTTP response time metric is more “representative” of user experience?

The 95%’lie or the 99.9%’lie
Gauging user experience

Example: A typical user session involves 5 page loads, averaging 40 resources per page.

- How many of our users will NOT experience something worse than the 95% ‘lie’?
  
  Answer: ~0.003%

- How many of our users will experience at least one response that is longer than the 99.9% ‘lie’?

  Answer: ~18%
Response Time vs. Service Time
Service Time vs. Response Time
Service Time, 90K/s vs 80K/s

90K: Max Service Time In Time Interval

80K: Max Service Time In Time Interval

90K: Service Time By Percentile Distribution

80K: Service Time By Percentile Distribution
Response Time, 90K/s vs 80K/s
Response Time, 90K/s vs 80K/s: Boom!
“coordinator as savior” latency myth

“But with Cassandra’s Coordinator and Quorum Consistency levels…”

Theory: If one node pauses, other nodes are not likely to pause at the same time... so a quorum will be reached without observing any one node's pause
Anatomy of a quorum read...

A pause here won't be noticed by client...

What about a pause here?

And since every node is also a coordinator...
Cassandra behavior on Zing
op rate : 40001
partition rate : 26996
row rate : 26996
latency mean : 30.6 (0.7)
latency median : 0.5 (0.5)
latency 95th percentile : 244.4 (1.1)
latency 99th percentile : 537.4 (2.0)
latency 99.9th percentile : 1052.2 (8.4)
latency max : 1314.9 (1312.8)
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>op rate</td>
<td>40001</td>
</tr>
<tr>
<td>partition rate</td>
<td>26961</td>
</tr>
<tr>
<td>row rate</td>
<td>26961</td>
</tr>
<tr>
<td>latency mean</td>
<td>0.6 (0.5)</td>
</tr>
<tr>
<td>latency median</td>
<td>0.5 (0.5)</td>
</tr>
<tr>
<td>latency 95th percentile</td>
<td>1.0 (0.9)</td>
</tr>
<tr>
<td>latency 99th percentile</td>
<td>2.7 (1.9)</td>
</tr>
<tr>
<td>latency 99.9th percentile</td>
<td>13.3 (3.8)</td>
</tr>
<tr>
<td>latency max</td>
<td>110.6 (28.2)</td>
</tr>
</tbody>
</table>
OpenJDK: 200–1400 msec stalls

- **Op rate**: 40001
- **Partition rate**: 26996
- **Row rate**: 26996
- **Latency mean**: 30.6 (0.7)
- **Latency median**: 0.5 (0.5)
- **Latency 95th percentile**: 244.4 (1.1)
- **Latency 99th percentile**: 537.4 (2.0)
- **Latency 99.9th percentile**: 1052.2 (8.4)
- **Latency max**: 1314.9 (1312.0)

Zing

- **Op rate**: 40001
- **Partition rate**: 26961
- **Row rate**: 26961
- **Latency mean**: 0.6 (0.5)
- **Latency median**: 0.5 (0.5)
- **Latency 95th percentile**: 1.0 (0.9)
- **Latency 99th percentile**: 2.7 (1.9)
- **Latency 99.9th percentile**: 13.3 (3.8)
- **Latency max**: 110.6 (28.2)
OpenJDK: 200–1400 msec stalls

Zing (drawn to scale)

<table>
<thead>
<tr>
<th>Metric</th>
<th>OpenJDK Latency</th>
<th>Zing Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time</td>
<td>200–1400 msec</td>
<td>10–200 msec</td>
</tr>
<tr>
<td>Service time</td>
<td>400–1300 msec</td>
<td>10–200 msec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>op rate</td>
<td>40001</td>
<td>40001</td>
</tr>
<tr>
<td>partition rate</td>
<td>26996</td>
<td>26961</td>
</tr>
<tr>
<td>row rate</td>
<td>26996</td>
<td>26961</td>
</tr>
<tr>
<td>latency mean</td>
<td>30.6 (0.7)</td>
<td>0.6 (0.5)</td>
</tr>
<tr>
<td>latency median</td>
<td>0.5 (0.5)</td>
<td>0.5 (0.5)</td>
</tr>
<tr>
<td>latency 95th percentile</td>
<td>244.4 (1.1)</td>
<td>1.0 (0.9)</td>
</tr>
<tr>
<td>latency 99th percentile</td>
<td>537.4 (2.0)</td>
<td>2.7 (1.9)</td>
</tr>
<tr>
<td>latency 99.9th percentile</td>
<td>1052.2 (8.4)</td>
<td>13.3 (3.8)</td>
</tr>
<tr>
<td>latency max</td>
<td>1314.9 (1312.8)</td>
<td>110.6 (28.2)</td>
</tr>
</tbody>
</table>
What if we focused on “already low latency” setups?

“I know really bad GC pauses may happen once in a while, but I’m interested in the common behavior between those...”
A set of pure read experiments...

aimed at highly repeatable results

(focused on frequent blips, not the hard to reliably repeat huge pauses)

* Same AWS r3.8xlarge instance (underutilized)
** single node cluster, pre-primed with 5M entries
*** stressed via (enhanced) cassandra-stress, pure read test
HotSpot @90K/s & 85K/s vs. Zing @90K/s & 85K/s

Wrong Place to Look:
They both “suck” at >85K/sec
HotSpot 85K/s vs. Zing 85K/s

Looks good, but still the wrong place to look
HotSpot @40K/s vs. Zing @40K/s

More interesting...
What can we do with this?
HotSpot @10K/s vs. Zing @40K/s

E.g. if “99%’ile < 5msec” was a goal:
Zing delivers similar 99%’ile and superior 99.9%’ile+
while carrying 4x the throughput
HotSpot @2K/s vs. Zing @20K/s

E.g. if “99.9%’ile < 10msec” was a goal:
Zing delivers similar 99%’ile and 99.9%’ile
while carrying 10x the throughput
HotSpot @2k thru 80k
HotSpot @2k thru 70k
Zing @20k thru 70k
Zing & HotSpot @2k thru 70k
Lots of conclusions can be drawn from the above...
E.g. Zing delivers a consistent 100x reduction in the rate of occurrence of >20msec response times
OpenJDK: 200–1400 msec stalls

Zing (drawn to scale)

<table>
<thead>
<tr>
<th>op rate</th>
<th>40001</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition rate</td>
<td>26996</td>
</tr>
<tr>
<td>row rate</td>
<td>26996</td>
</tr>
<tr>
<td>latency mean</td>
<td>30.6 (0.7)</td>
</tr>
<tr>
<td>latency median</td>
<td>0.5 (0.5)</td>
</tr>
<tr>
<td>latency 95th percentile</td>
<td>244.4 (1.1)</td>
</tr>
<tr>
<td>latency 99th percentile</td>
<td>537.4 (2.0)</td>
</tr>
<tr>
<td>latency 99.9th percentile</td>
<td>1052.2 (8.4)</td>
</tr>
<tr>
<td>latency max</td>
<td>1314.9 (1312.8)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op rate</th>
<th>40001</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition rate</td>
<td>26961</td>
</tr>
<tr>
<td>row rate</td>
<td>26961</td>
</tr>
<tr>
<td>latency mean</td>
<td>0.6 (0.5)</td>
</tr>
<tr>
<td>latency median</td>
<td>0.5 (0.5)</td>
</tr>
<tr>
<td>latency 95th percentile</td>
<td>1.0 (0.9)</td>
</tr>
<tr>
<td>latency 99th percentile</td>
<td>2.7 (1.9)</td>
</tr>
<tr>
<td>latency 99.9th percentile</td>
<td>13.3 (3.8)</td>
</tr>
<tr>
<td>latency max</td>
<td>110.6 (28.2)</td>
</tr>
</tbody>
</table>

©2015 Azul Systems, Inc.
A simple visual summary

This is Cassandra on HotSpot

This is Cassandra on Zing

Any Questions?