Understanding Application Hiccups

and what you can do about them

An introduction to the Open Source jHiccup tool

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

- co-founder, CTO @Azul Systems
- Have been working on "think different" GC approaches since 2002
- Created Pauseless & C4 core GC algorithms (Tene, Wolf)
- A Long history building Virtual & Physical Machines, Operating Systems, Enterprise apps, etc...

* working on real-world trash compaction issues, circa 2004
About Azul

- We make scalable Virtual Machines
- Have built “whatever it takes to get job done” since 2002
- 3 generations of custom SMP Multi-core HW (Vega)
- Now Pure software for commodity x86 (Zing)
- “Industry firsts” in Garbage collection, elastic memory, Java virtualization, memory scale
A classic look at response time behavior

Key Assumption: Response time is a function of load

Average? Max? Median? 90%? 99.9%

* source: IBM CICS server documentation, “understanding response times”
Common fallacies

- Computers run application code continuously
  - CPUs stop processing application code for all sorts of reasons
  - e.g: Interrupts. Scheduling of other work, swapping, etc.
  - Modern system architectures add more: Power management, Virtualization (cross-image context switching, physical VM motion), Garbage Collection, etc.

- Response time can be measured as work units/time.

- Response time exhibits a normal distribution
  - Leading to attempts to represent with average + std. deviation
Response time over time

When we measure behavior over time, we often see: "Hiccups"

* source: ZOHO QEngine White Paper: performance testing report analysis
Application Hiccups

Where do they come from?
- Usually a factor of outside of the individual transaction work
- E.g: Queueing, accumulated work, platform inconsistency

Do they matter?
- That depends. What are your end-user’s expectations?
- Hiccups often dominate response time behavior

How can/should we measure them?
- Average? Max? 99.9%? Mean with Std. deviation?

Hiccup magnitude is often not a function of load
What happened here?

*Source: Gil running an idle program and suspending it five times in the middle*
Pitfall: Calculating %iles “naïvely”

Common Example:
- build/buy simple load tester to measure throughput
- issue requests one by one at a certain rate
- measure and log response time for each request
- results log used to produce histograms, percentiles, etc.

So what’s wrong with that?
- works well only when all responses fit within in rate interval
- technique includes “automatic backoff” and coordination
- But requirements interested in random, uncoordinated requests

Bad how bad can this get, really?
Example of naïve %’ile

System easily handles 100 requests/sec
Responds to each in 1 msec

How would you characterize this system?

Naïve results:
10,000 @ 1 msec
1 @ 100 second

Naïve characterization: 99.99% below 1 sec !!!
Proper measurement

System easily handles 100 requests/sec
Responds to each in 1msec

System Stalled for 100 Sec

10,000 results Varying linearly from 100 sec to 10 msec

10,000 results @ 1 msec each

Proper characterization: 50% below 1 second
jHiccup

- A tool for capturing and displaying platform hiccups
  - Records any observed non-continuity of the underlying platform
  - Plots results in simple, consistent format

- Simple, non-intrusive
  - As simple as adding the word “jHiccup” to your java launch line
  - % jHiccup java myflags myApp
  - Adds a background thread that samples time @ 1000/sec

- Open Source
  - Released to the public domain, creative commons CC0
what jHiccup measures

jHiccup measures the platform, not the application

- Experiences whatever delays application threads would see
- Highlights inconsistency in platform execution of “nothing”
- No attempt to identify cause (e.g. scheduling, GC, swapping, etc.)

An application cannot behave better than it’s platform

- If the platform stalls, the application stalled as well.
- jHiccup provides a lower bound for “best application behavior”

Useful control measurements

- Comparing jHiccup results for an idle application running on the same platform, at the same time, narrows down behavior to process-specific artifacts (-c option)
The anatomy of Hiccup Charts
Telco App Example

Max Time per interval

Optional SLA plotting

Hiccup duration at percentile levels
Hiccups by Time Interval

Hiccup Duration (msec) vs. Elapsed Time (sec)

- Max per Interval
- 99%
- 99.90%
- 99.99%
- Max

Hiccups by Percentile Distribution

Hiccup Duration (msec) vs. Percentile

Max=967.68
How jHiccup works

- Background thread, measures time to do "nothing"
  - Sleeps for 1 milliseconds, allocates 1 object, records time gap
  - Collects detailed "floating point" histogram of observed times
  - Constant, small memory footprint, virtually no cpu load
  - No effect on application threads

- Outputs two files
  - A time-based log with 1 line per interval (default 5 sec interval).
  - A percentile distribution log of accumulated histogram data

- Spreadsheet imports files and plots data
  - A standard, simple chart format
Some fun with jHiccup
Examples
A 1GB Java Cache under load

Hiccups by Time Interval

- Max per Interval
- 99%
- 99.90%
- 99.99%
- Max

Hiccups by Percentile Distribution

Max=3448.832
Fun with jHiccup

Charles Nutter  @headius

jHiccup, @AzulSystems' free tool to show you why your JVM sucks compared to Zing: bit.ly/wsH5A8 (thx @bascule)

Retweeted by Gil Tene
Obviously, we can use it for comparison purposes
Zing 5, 1GB in an 8GB heap

Oracle HotSpot CMS, 1GB in an 8GB heap
Zing 5, 1GB in an 8GB heap

Hiccups by Time Interval

- Max per Interval
- 99%
- 99.90%
- 99.99%
- Max

Hiccups by Percentile Distribution

- Max=20.384
- 99%

Oracle HotSpot CMS, 1GB in an 8GB heap

Hiccups by Time Interval

- Max per Interval
- 99%
- 99.90%
- 99.99%
- Max

Hiccups by Percentile Distribution

- Max=13156.352
Oracle HotSpot CMS, 4GB in a 18GB heap

Hiccups by Time Interval

Hiccup Duration (msec)

Elapsed Time (sec)

Hiccups by Percentile Distribution

Max=40173.568

Oracle HotSpot CMS, 1GB in an 8GB heap

Hiccups by Time Interval

Hiccup Duration (msec)

Elapsed Time (sec)

Hiccups by Percentile Distribution

Max=13156.352
Q & A

http://www.azulsystems.com

http://www.azulsystems.com/dev_resources/jhiccup