THE POWER OF INFINITE POSSIBILITIES

ARIEL WEISBERG

TAMING THE 9S

ABOUT ME

- 5 YEARS WORKING ON VOLTDB
  - In-memory, durable, SQL transactional, shared nothing linear scale out
  - Storage engine, client libraries, distributed agreement, snapshots, write-ahead logging, clustering, elasticity, export, WAN replication…
- Spent the last year thinking in nanoseconds
  - Microseconds
  - Milliseconds,
  - Seconds
    - Not so much with the seconds

VOLTDB

What are the 9s?

- Percentage of recorded events < a given value
  - 99% of all tasks complete in < 30 milliseconds
- SLAs
  - Expectation of how long tasks can take
  - Every application has them
  - 99% of all tasks complete in < 30 milliseconds with a max of 1 second @ 50,000 tasks a second
- The 9s
  - Where those expectations are violated (sometimes sneakily)

What are the 9s?

- How not to measure latency – Gil Tene
  - http://goo.gl/jyuHVX
- Latency metrics
  - Min
    - Useless outside of sanity checking
  - Average
    - I hear banjos
  - Average + STD deviation
    - Run away

What are the 9s?

- Latency doesn't follow a normal distribution
- When latency spikes it spikes 10-100x or more
- Spikes are correlated across multiple unrelated tasks
  - This is important
- AVG + STD deviation hides spikes
- Small reporting intervals hide less
  - 1 second instead of one minute

VOLTDB

HdrHistogram

- There is a better way
  - github.com/HdrHistogram/HdrHistogram
  - Gil Tene of Azul Systems and Michael Barker
- Configurable space precision tradeoff
- Huge range
- Fast
- Easy to query, sum, difference – multiple consumers
- Available in Java and C++

VOLTDB

Why long tail latency matters

- Latency of all tasks is a function of the latency of subtasks
- As the # of subtasks increases latency of tasks approaches that of the long tail
- Correlated spikes in latency cause cascading failures

VOLTDB

Don't Miss 20% Off Your Favorite reddit Hoodie! (...redditgifts.com)

promoted by reddit_marketplace

10 comments share

sponsored link what's this?

9

The reddit_101 FAQ (self.AnimalsBeingJerks)

submitted 1 month ago by pencer Soda-Jerk - stickied post

1 comment share

t started out so innocent... (i.imgur.com)

submitted an hour ago by TBones0072

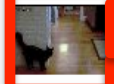
14 comments share

390

Boxing Kangaroo (i.4cdn.org)

submitted 12 hours ago by Boomsdoggie

18 comments share



ain't phased. (i.imgur.com)

comment share

40

Shh... Don't make a sound. OH SHIT! RUN! (i.imgur.com)

submitted 1 day ago by 1Voice1Life

124 comments share

ain't budging. (i.imgur.com)

ago by yall_cray

comment share

search reddit

username | password | remember me | reset password | LOGIN

SUBMIT LINK +

SUBMIT SELF +

SUBSCRIBE +

174,286 subscribers, 189 online

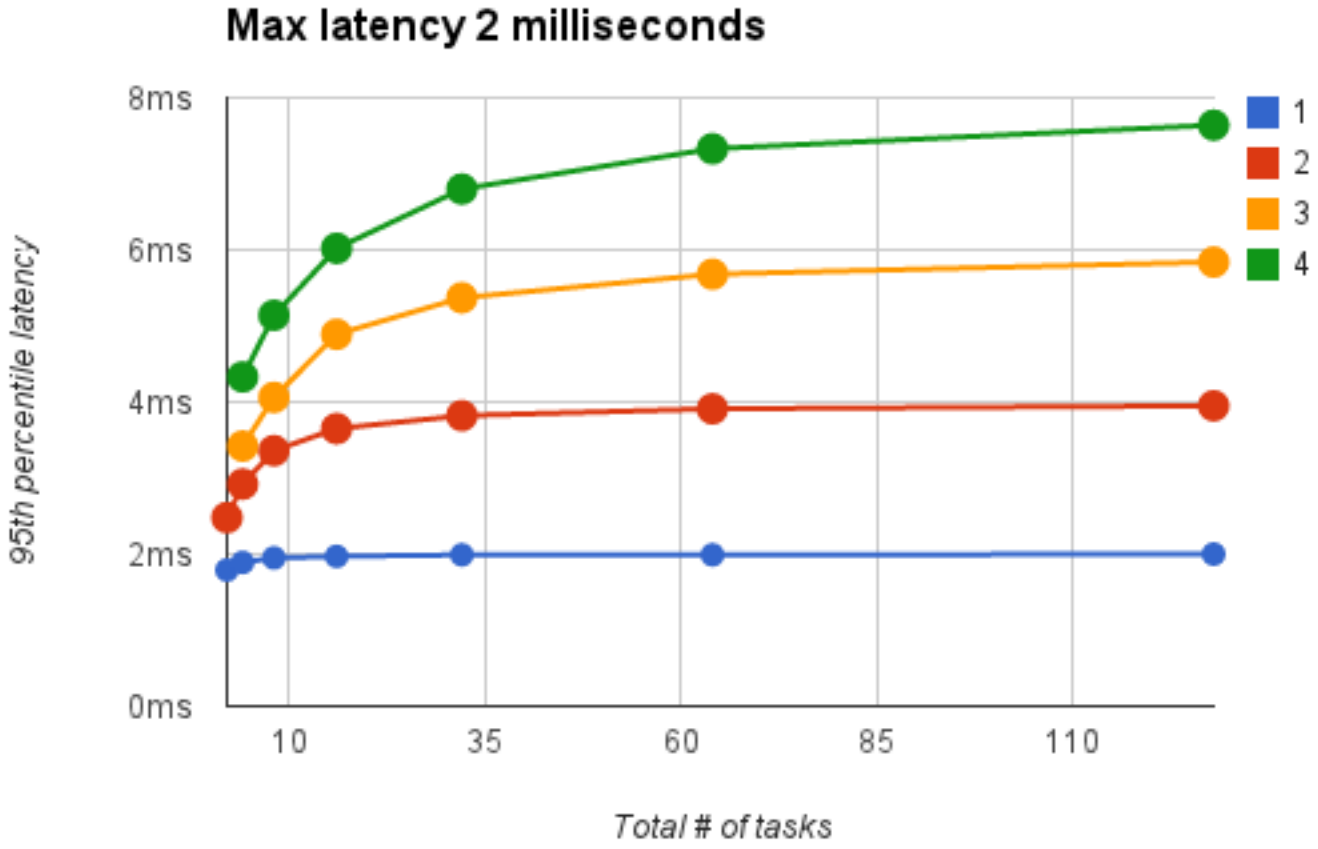
NEW TO REDDIT? CLICK HERE!

A place for sharing videos, gifs, and images of animals being total jerks.

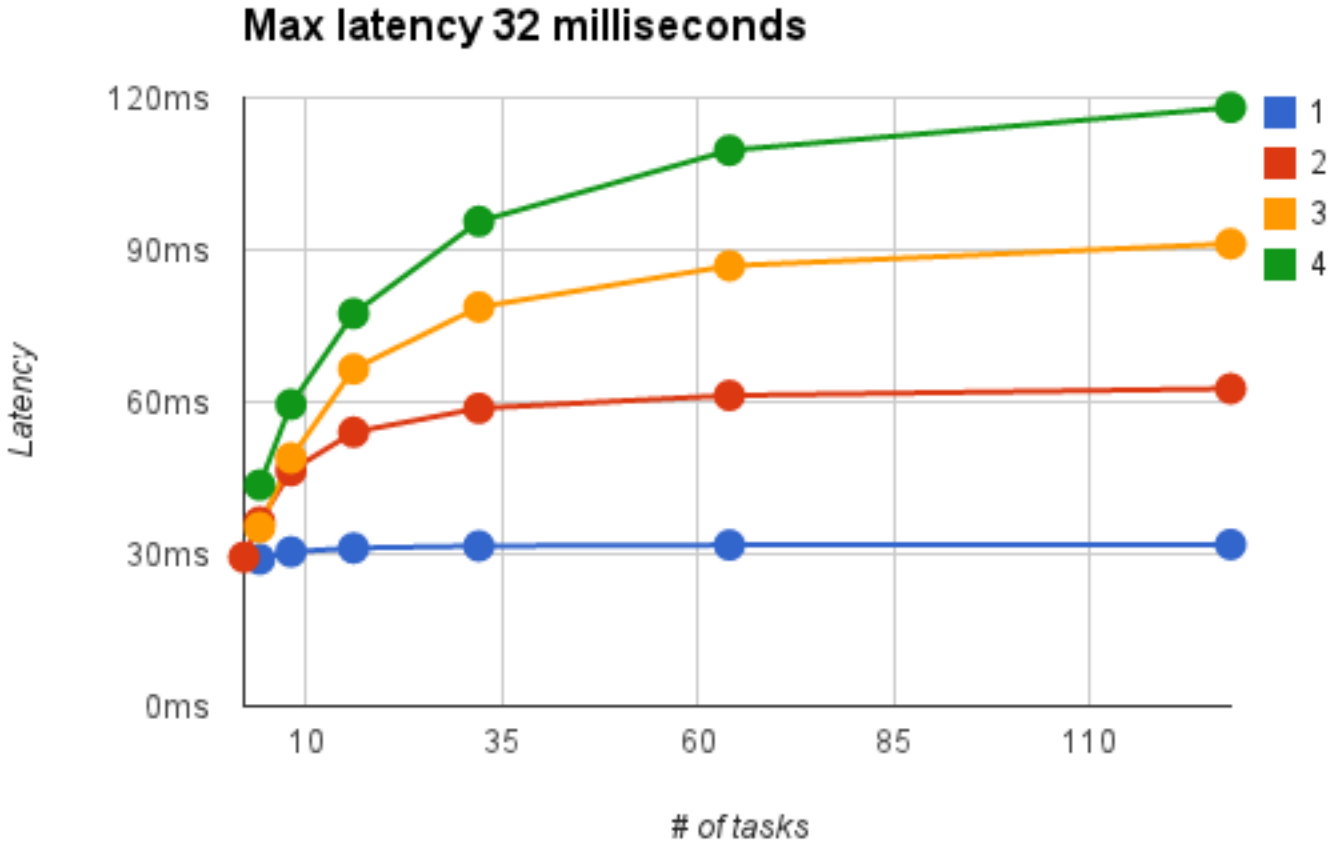
Current Filter: **None**

Latency simulation

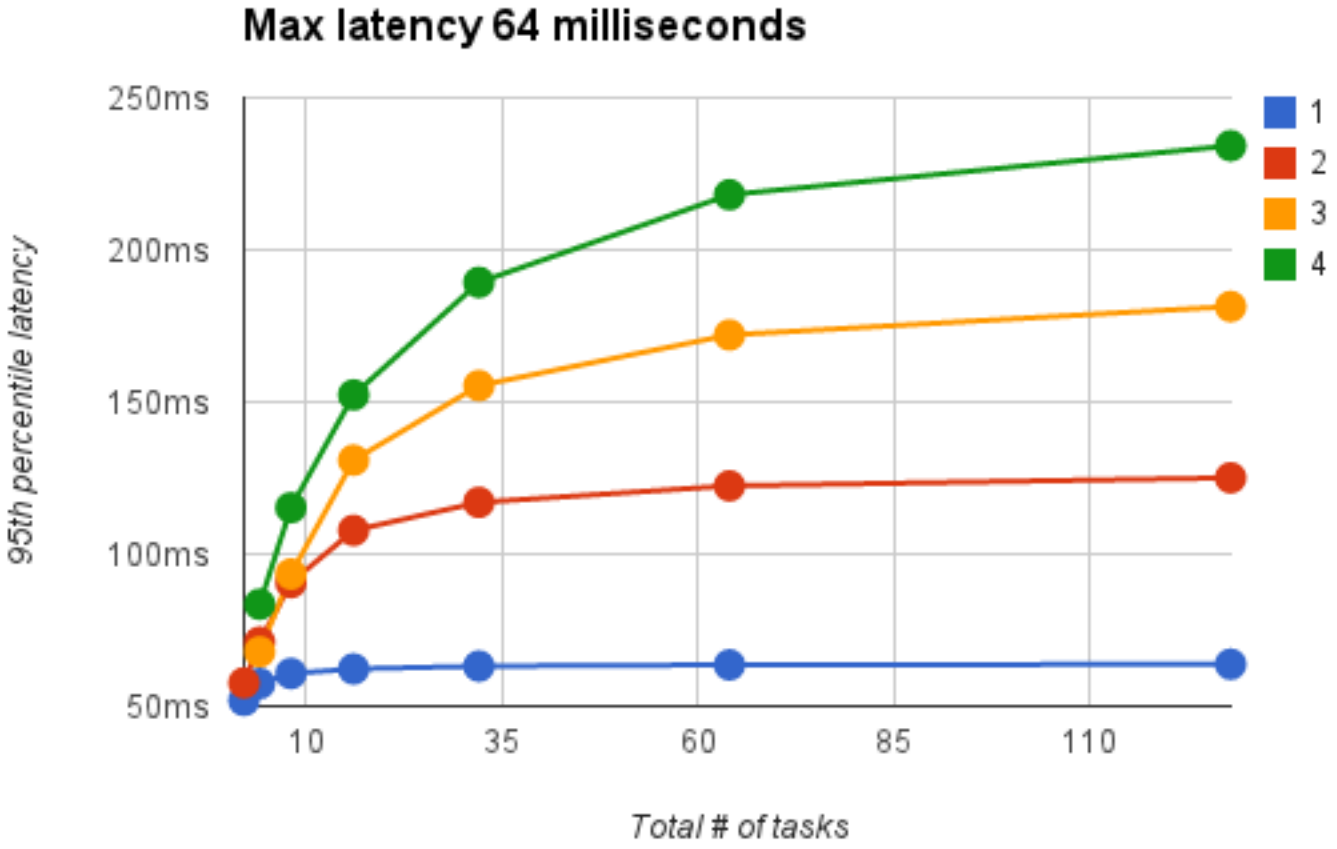
- pastebin.com/pa1B7pte
- 80 worker threads
- 10 second simulated measurement period
- Workers process tasks
- Tasks consists of variable number of subtasks
- Zipfian distribution of subtask time
- Vary between processing tasks concurrently or in batches


Max latency 2 milliseconds
8ms
6ms
4ms
2ms
0ms
95th percentile latency
10
35
60
85
110
Total # of tasks
1
2
3
4

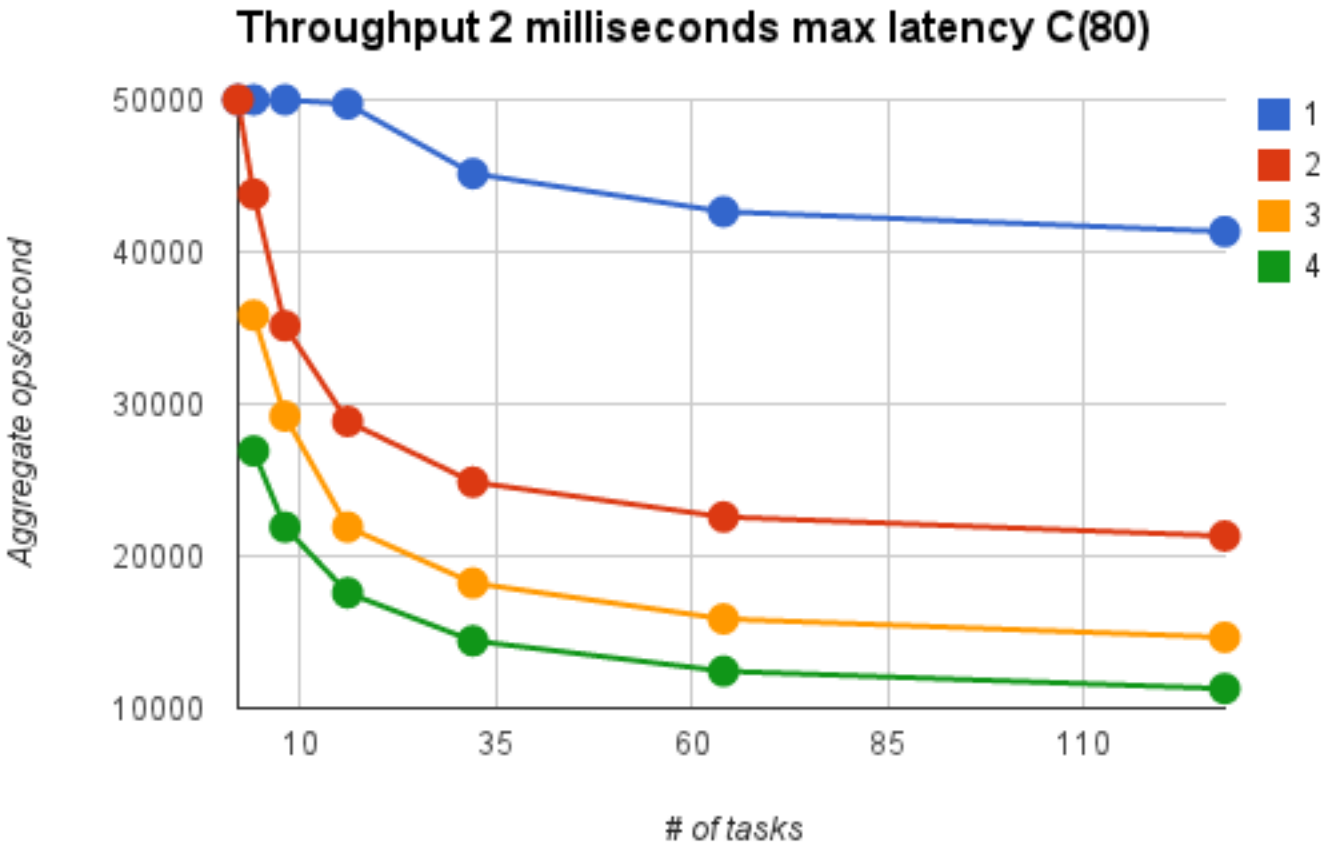
VOLTDB


Max latency 32 milliseconds
120ms
90ms
60ms
30ms
0ms
Latency
10
35
60
85
110
# of tasks
1
2
3
4

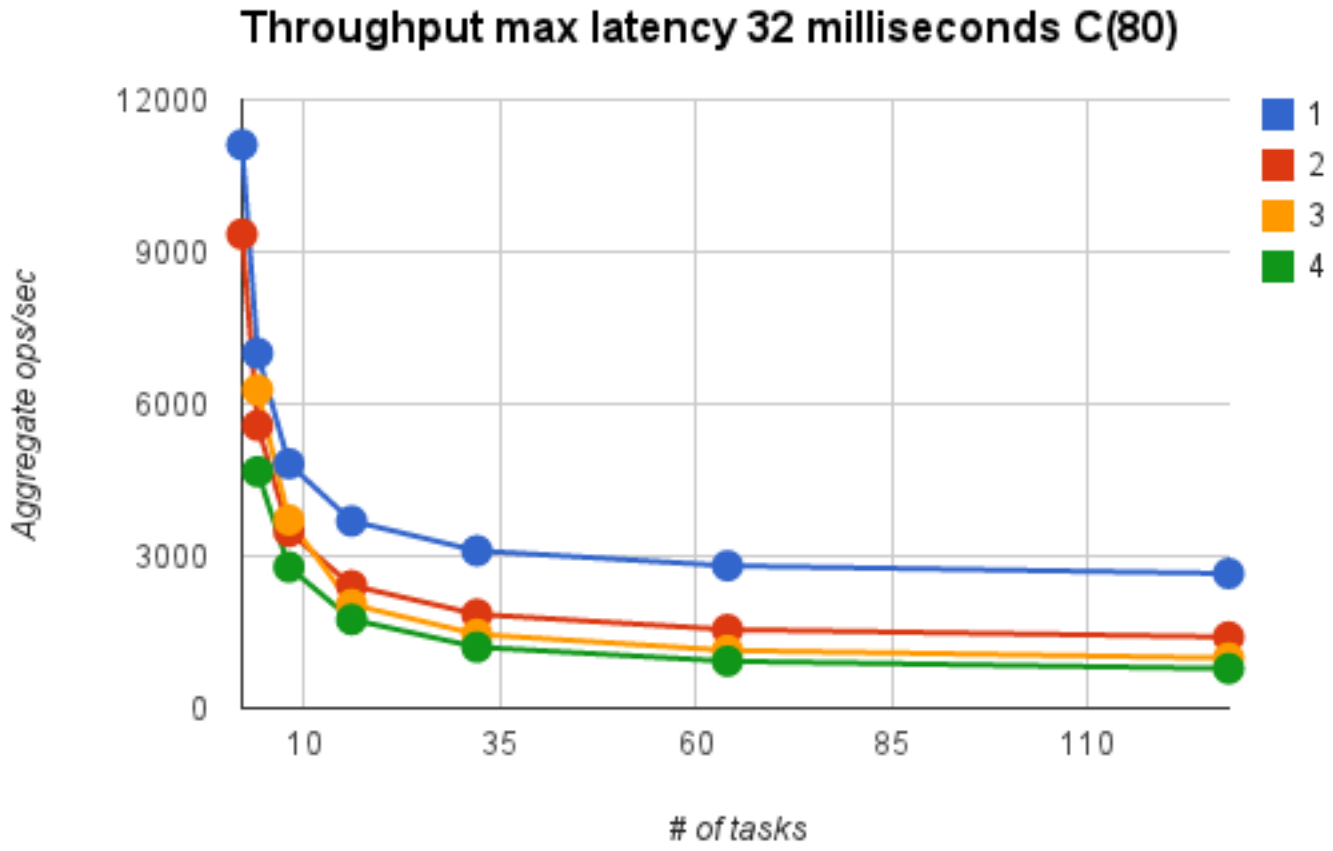
VOLTDB


Max latency 64 milliseconds
250ms
200ms
150ms
100ms
50ms
95th percentile latency
10
35
60
85
110
Total # of tasks
1
2
3
4

VOLTDB


Throughput 2 milliseconds max latency C(80)
Aggregate ops/second
50000
40000
30000
20000
10000
10
35
60
85
110
# of tasks
1
2
3
4

VOLTDB

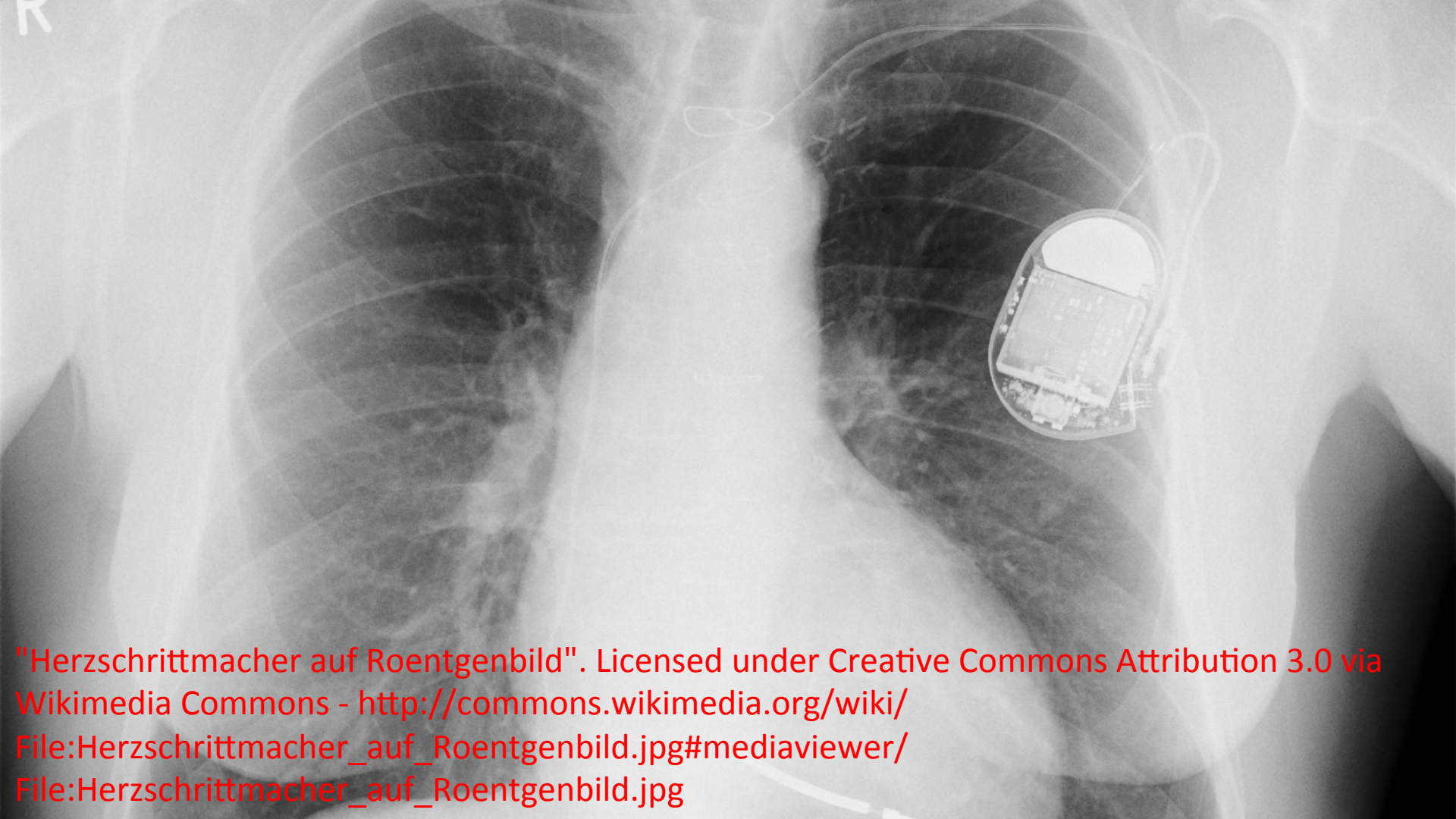

Throughput max latency 32 milliseconds C(80)
12000
9000
6000
3000
0
Aggregate ops/sec
10
35
60
85
110
# of tasks
1
2
3
4

VOLTDB

Coordinated Omission

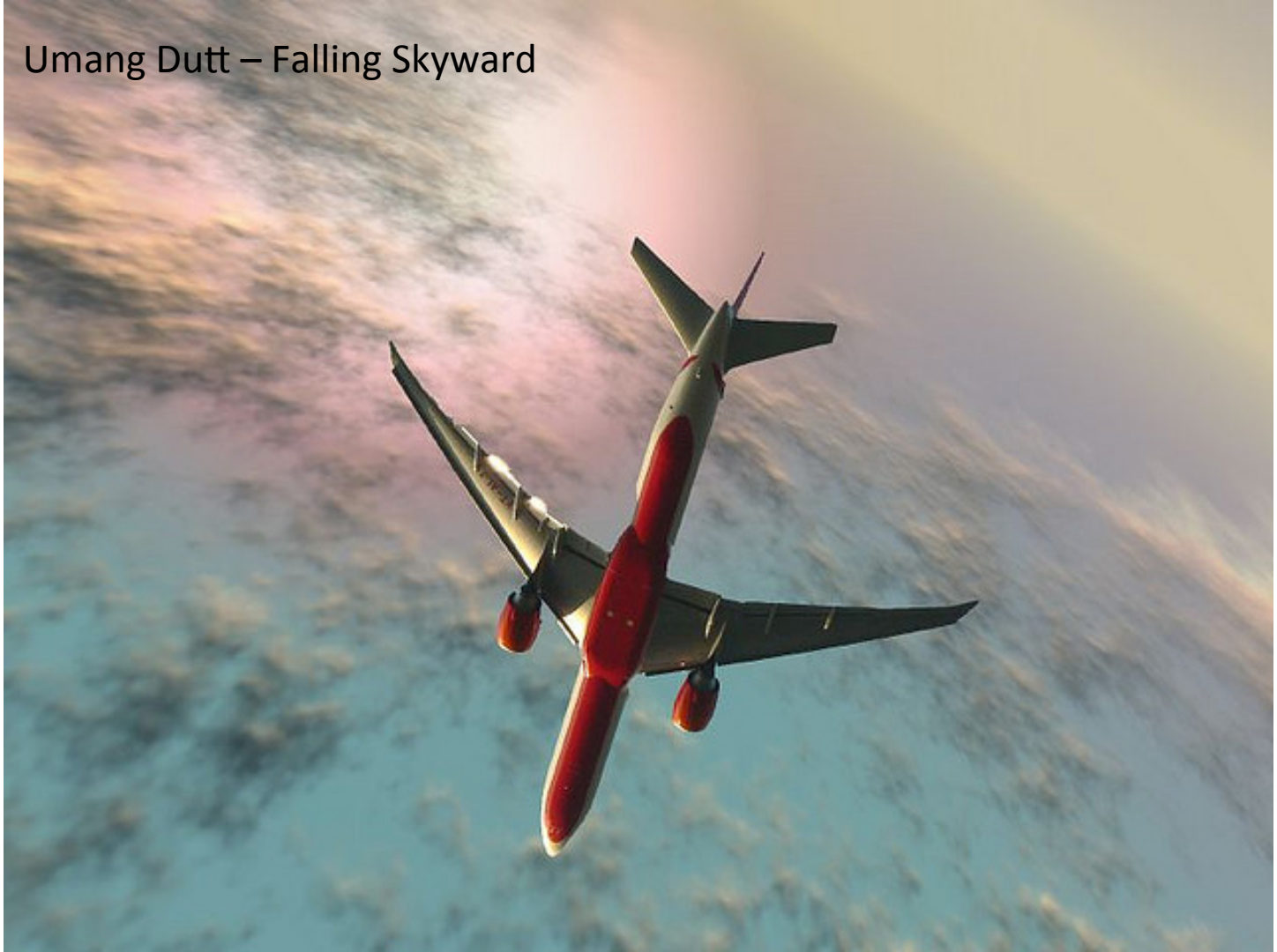
- Measurement error with bounded concurrency
- Latency only sampled for requests that are known
  - Requests not generated
  - Waiting in network buffers
- HdrHistogram to the rescue
  - Synthesize missing samples
  - Based on expected arrival rate
- latencyutils.github.io/LatencyUtils
  - Automate correcting coordinated omission

Hard real-time




Infradept, Flickr

Umang Dutt – Falling Skyward



VOLTDB

Hard real-time

- Less distinction between maximum and high percentiles
- Worst case and never failing is priority 1

Soft real-time

VOLTDB

Soft real-time

- Only need to be faster most of the time
- Worst case limits matter and are lowish
- Required performance at high percentiles differs from max

Squishy real-time

Don't Miss 20% Off Your Favorite reddit Hoodie! (...redditgifts.com)

promoted by reddit_marketplace

10 comments share

sponsored link what's this?

9

The reddit_101 FAQ (self.AnimalsBeingJerks)

submitted 1 month ago by pencer Soda-Jerk - stickied post

1 comment share

t started out so innocent... (i.imgur.com)

submitted an hour ago by TBones0072

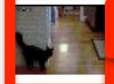
14 comments share

390

Boxing Kangaroo (i.4cdn.org)

submitted 12 hours ago by Boomsdoggie

18 comments share



ain't phased. (i.imgur.com)

comment share

40

Shh... Don't make a sound. OH SHIT! RUN! (i.imgur.com)

submitted 1 day ago by 1Voice1Life

124 comments share

ain't budging. (i.imgur.com)

ago by yall_cray

comment share

search reddit

username | password | remember me | reset password | LOGIN

SUBMIT LINK +

SUBMIT SELF +

SUBSCRIBE +

174,286 subscribers, 189 online

NEW TO REDDIT? CLICK HERE!

A place for sharing videos, gifs, and images of animals being total jerks.

Current Filter: **None**

Squishy real-time

- Tolerable max is many times higher
- Tolerable latency at higher percentiles looks higher
- In practice tolerable latency might not be so forgiving

Developed VoltDB to expectations

- In-memory
  - No reason to tolerate IO pauses
- Java
  - Must tolerate young-gen GC pauses
  - Must tolerate infrequent and small old-gen pauses
- Linux
  - See IO pauses

Developed VoltDB to expectations

- Young gen GC
  - Frequency can be designed/tuned to once a second or more
  - GC time can be tuned to order of single digit milliseconds
- Old gen GC
  - Frequency can be tuned to order of days
  - Never is a possibility (for Young Gen as well)
    - But then it's not Java
  - Time can be tuned to order of 10s of milliseconds
- Expectations
  - 99.999% at single digit milliseconds (Young gen GC)
  - Maximum at 10s of milliseconds (Old Gen GC)

VOLTDB

Continuous Integration as a starting point

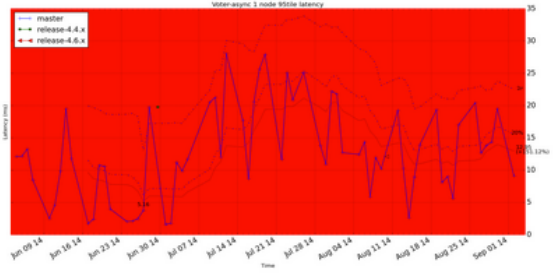
- Improvements don't stick without continuous monitoring
- Tracking 120 different KPIs
- Need KPIs to signal attention needed (120 is too many)
- KPIs need to visualize in a way that highlights changes
  - Bad news comes fractions of a % at a time
  - Catching regressions quickly eases correlation with changes

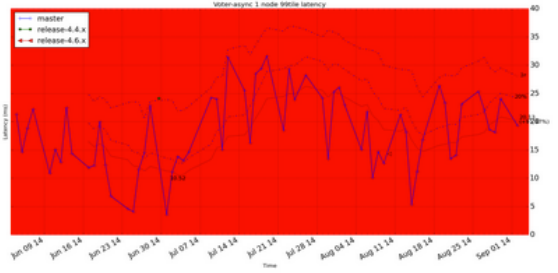
VOLTDB

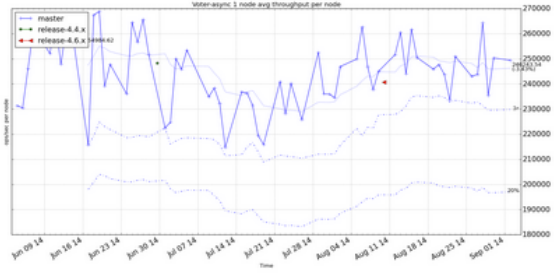
Adhoc-citadel 1 node	Adhoc-citadel 2 nodes (by 10.71%)	ClientAffinity 3 nodes (by 5.25%)	CSV-narrow-ix 1 node
CSV-narrow-ix 3 nodes	CSV-narrow-noix 1 node	CSV-narrow-noix 3 nodes	CSV-replicated-pk 1 node (by 7.09%)
CSV-replicated-pk 3 nodes	CSV-wide-ix 1 node	CSV-wide-ix 3 nodes	CSV-wide-noix 1 node
CSV-wide-noix 3 nodes	Join-cached 1 node	Join-cached 2 nodes	Join-uncached 1 node
Join-uncached 2 nodes	JoinMP-cached 1 node (by 5.19%)	JoinMP-cached 2 nodes	JoinMP-uncached 1 node
JoinMP-uncached 2 nodes	JoinSP-cached 1 node	JoinSP-cached 2 nodes	JoinSP-uncached 1 node
JoinSP-uncached 2 nodes	Joinstar-cached 1 node (by 6.66%)	Joinstar-cached 2 nodes (by 5.49%)	Joinstar-uncached 1 node
Joinstar-uncached 2 nodes	JoinstarMP-cached 1 node (by 7.65%)	JoinstarMP-cached 2 nodes	JoinstarMP-uncached 1 node
JoinstarMP-uncached 2 nodes	JoinstarSP-cached 1 node	JoinstarSP-cached 2 nodes (by 19.62%)	JoinstarSP-uncached 1 node
JoinstarSP-uncached 2 nodes	KVBenchmark-five9s 3 nodes (by 9.01%)	KVBenchmark-five9s-latency	KVBenchmark-five9s-nofail 3 nodes
KVBenchmark-five9s-nofail-latency	KVBenchmark-five9s-nofail-nocl 3 nodes	KVBenchmark-five9s-nofail-nocl-latency	MatViewBench Delete Diff 1 node (by 26.46%)
MatViewBench Delete w min MV 1 node	MatViewBench Delete w MV 1 node	MatViewBench Delete wo MV 1 node	MatViewBench Insert Diff 1 node (by 84.73%)
MatViewBench Insert w min MV 1 node (by 7.08%)	MatViewBench Insert w MV 1 node (by 8.59%)	MatViewBench Insert wo MV 1 node (by 10.20%)	MatViewBench Update Grp Diff 1 node (by 31.02%)
MatViewBench Update Grp w MV 1 node	MatViewBench Update Grp wo MV 1 node (by 5.27%)	MatViewBench Update Sum Diff 1 node (by 17.88%)	MatViewBench Update Sum w MV 1 node
MatViewBench Update Sum wo MV 1 node (by 5.21%)	OneShotKV-1PartMpGet 3 nodes (by 8.79%)	OneShotKV-1PartMpOptimisticPut 3 nodes (by 6.71%)	OneShotKV-2PartMpOptimisticPut 3 nodes (by 7.17%)
OneShotKV-MP-Reads 1 node (by 5.88%)	Perf-CL-Async-RW-0 1 node (by 6.08%)	Perf-CL-Async-RW-128 1 node (by 13.02%)	Perf-CL-Async-RW-1k 1 node (by 6.46%)
Perf-CL-Async-WO-0 1 node (by 34.01%)	Perf-CL-Async-WO-128 1 node (by 14.10%)	Perf-CL-Async-WO-1k 1 node	Perf-CL-Sync-RW-0 1 node (by 9.30%)
Perf-CL-Sync-RW-128 1 node	Perf-CL-Sync-RW-1k 1 node (by 5.93%)	Perf-CL-Sync-WO-0 1 node	Perf-CL-Sync-WO-128 1 node
Perf-CL-Sync-WO-1k 1 node	Projection-cached 1 node (by 51.30%)	Projection-cached 2 nodes (by 20.06%)	Projection-uncached 1 node
Projection-uncached 2 nodes	ProjectionMP-cached 1 node (by 6.42%)	ProjectionMP-cached 2 nodes (by 27.14%)	ProjectionMP-uncached 1 node
ProjectionMP-uncached 2 nodes (by 6.94%)	ProjectionSP-cached 1 node (by 19.42%)	ProjectionSP-cached 2 nodes (by 32.67%)	ProjectionSP-uncached 1 node (by 5.54%)
ProjectionSP-uncached 2 nodes	Scan-index 1 node	Scan-table 1 node	Snapshot-impact 1 node
TPC-C-volt3 1 node (by 11.94%)	TPC-C-volt3 3 nodes (by 5.26%)	TPC-C-volt3 6 nodes (by 5.71%)	TPC-C-volt3-commandlogenabled 1 node
TPC-C-volt3-commandlogenabled 3 nodes	TPC-C-volt3-commandlogenabled 6 nodes (by 5.41%)	VoltCache 1 node (by 5.40%)	VoltCache 3 nodes
VoltCache-commandlogenabled 1 node	VoltCache-commandlogenabled 3 nodes	VoltKV-async 1 node	VoltKV-async 3 nodes
VoltKV-async-commandlogenabled 1 node	VoltKV-async-commandlogenabled 3 nodes	VoltKV-jdbc 1 node (by inf%)	VoltKV-jdbc 3 nodes (by inf%)

ci:8080/job/performance-graphs/lastSuccessfulBuild/artifact/performance-graphs-90-days/stats-index.html#Voter-async%201%20node

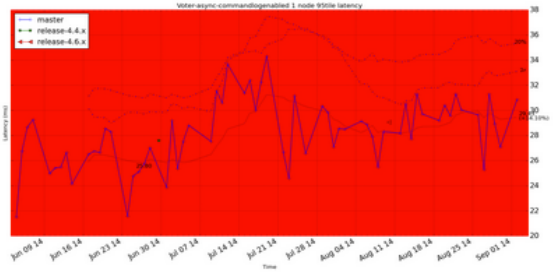
Voter-async 1 node

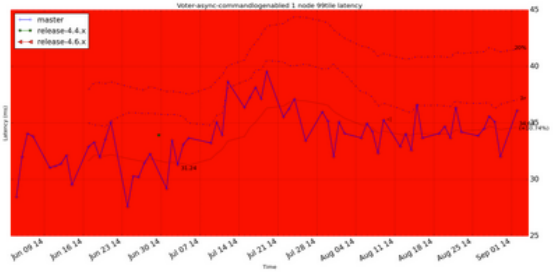




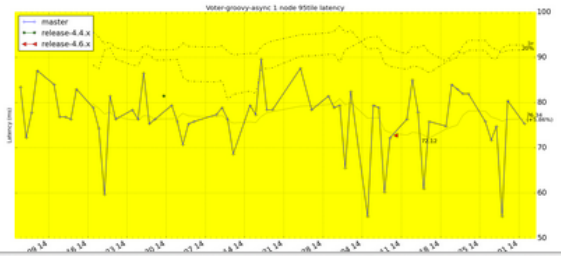


Voter-async-commandlogenabled 1 node





Voter-groovy-async 1 node

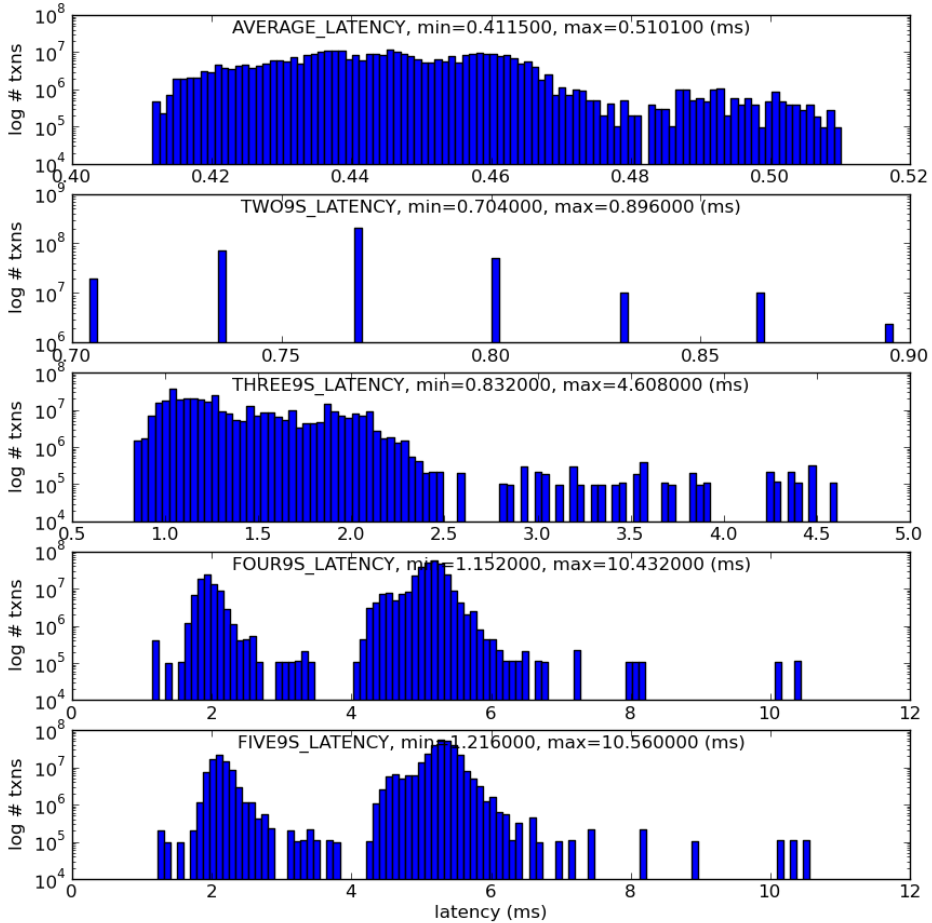


5nines-histograms.png | stats_KVBenchmark-f....png | stats_KVBenchmark-f....png | stats_KVBenchmark-f....png | Show All


KVBenchmark-five9s-nofail-nocl 3 nodes Max 5-9's latency
master
latency (ms)
600
500
400
300
200
100
0
Jun 09 14
Jun 16 14
Jun 23 14
Jun 30 14
Jul 07 14
Jul 14 14
Jul 21 14
Jul 28 14
Aug 04 14
Aug 11 14
Aug 18 14
Aug 25 14
Sep 01 14
Time


KVBenchmark-five9s-nofail-nocl 3 nodes Pct Tx with 5-9's latency < 50ms
master
Percent of completed tx
100.00
99.95
99.90
99.85
99.80
99.75
99.70
99.65
Jun 09 14
Jun 16 14
Jun 23 14
Jun 30 14
Jul 07 14
Jul 14 14
Jul 21 14
Jul 28 14
Aug 04 14
Aug 11 14
Aug 18 14
Aug 25 14
Sep 01 14
Time

latency histograms, sample count=3479, duration=3477999 (ms), # tx=377991984



plotted: 2014/08/30 10:26:18 branch=master build=jenkins-performance-nextrelease-5nines-nofail-nocl-6 kit=jenkin

Outlier whack-a-mole

- Outliers caused by things that don't occur all the time
- There is breadth to root causes
  - Bugs (no reason to suffer the outlier)
    - Infrequently holding a lock while doing a long task
  - Design (that was never going to work)
    - Using old gen GC
  - Right thing the wrong way
    - Disk IO from the wrong thread
    - Periodic batch work – wrong batch size or wrong thread

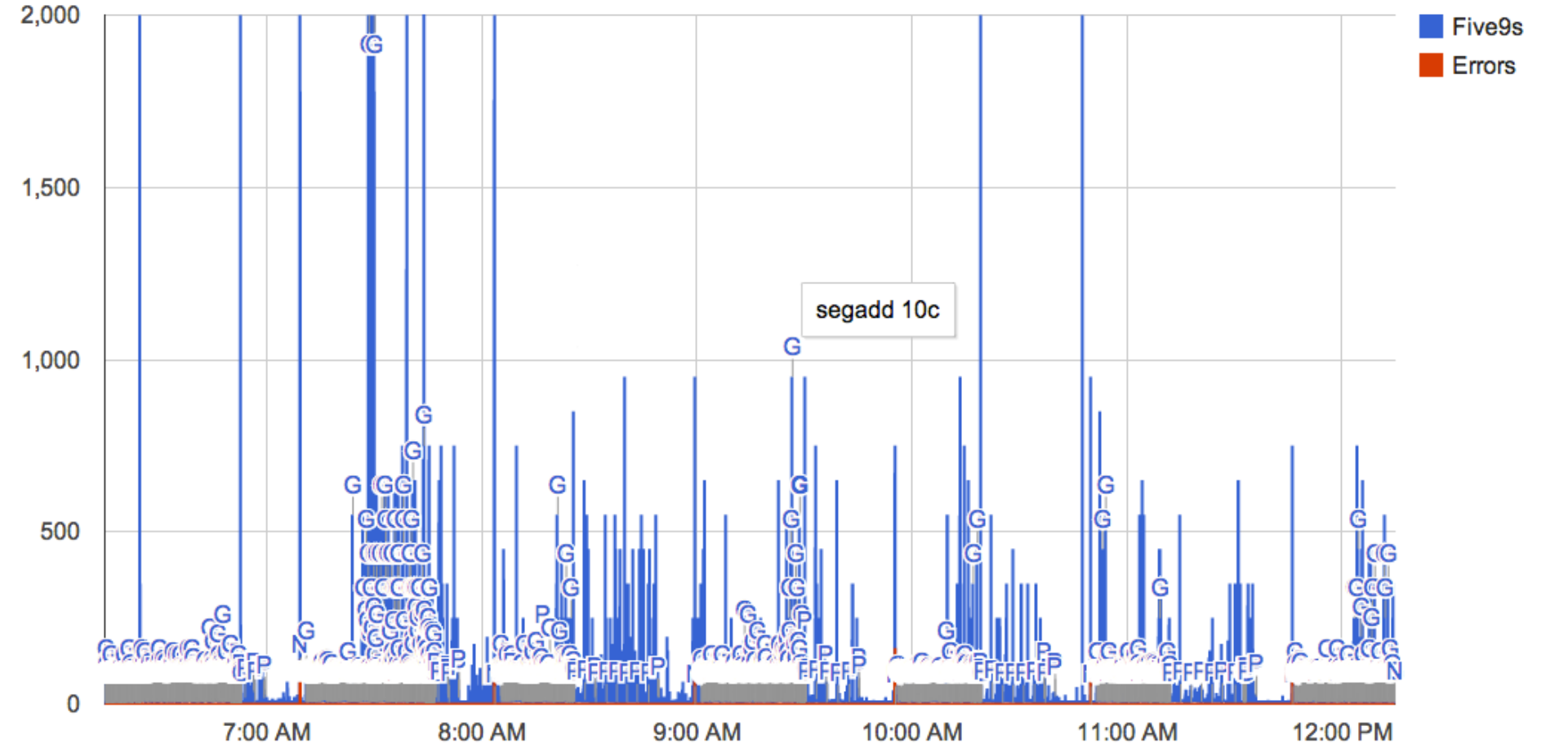
VOLTDB

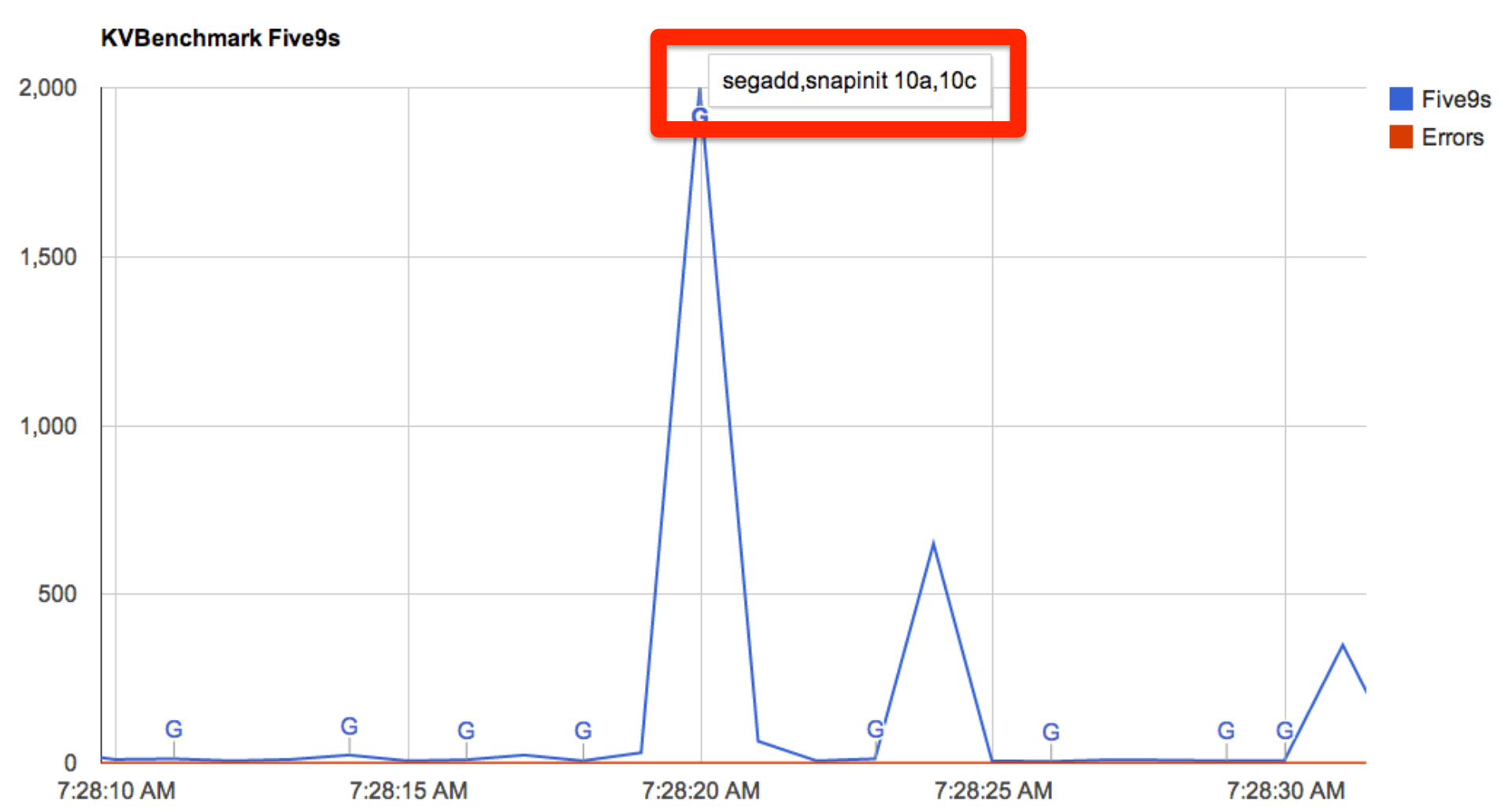
A list

- Log4j writing to file
- GC logging to file blocks VM
- Background large file copy blocks entire node
- Non-blocking socket writes block
- Linux zone_reclaim
- Linux transparent huge pages
- Time to safe point extended by threads blocked in memory mapped file page fault
- On heap object promotion to old gen causing GCs
- Slow young gen GCs due to live object copying of large live object sets
- Java RMI instantiated by JMX invoking System.gc() every hour
- System.gc() not concurrent by default
- Class unloading not concurrent by default (and perm gen default too small)
- Critical threads doing disk IO or waiting on threads doing disk IO (reads or writes)
- Threads holding locks while doing disk IO
- ??java.lang.ref.Reference$Lock?? Appearing randomly as monitor contention
- Heavy weak/soft reference usage extending GC pauses
- Load generators and client libraries pause too!
- Buggy NIC drivers, disable offloads

Visibility

- Every latency outlier has a cause
- Challenge is finding the cause
- Automating as much of the process as possible is key

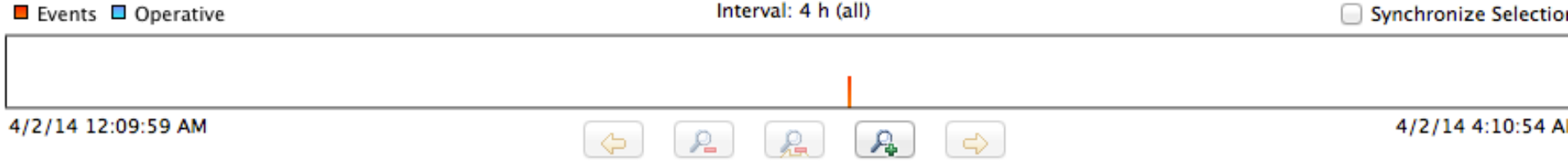

2,000
1,500
1,000
500
0
7:00 AM
8:00 AM
9:00 AM
10:00 AM
11:00 AM
12:00 PM
Five9s
Errors
segadd 10c


KVBenchmark Five9s
segadd,snapinit 10a,10c
Five9s
Errors
2,000
1,500
1,000
500
0
7:28:10 AM
7:28:15 AM
7:28:20 AM
7:28:25 AM
7:28:30 AM
G

Flight Recorder is the heat

- Continuous profiling
  - CPU sampling – accurate, not at just at safe points
  - Monitor contention
  - Time spent parked, waiting
  - Network and disk IO waits
  - GC
- Free for non-production use
- Will almost always tell you what thread pains you

GC Configuration



GC Configuration

Young Garbage Collector	ParNew
Old Garbage Collector	ConcurrentMarkSweep
Concurrent GC Threads	5
Parallel GC Threads	18
Cuncurrent Explicit GC	Yes
Disabled Explicit GC	No
Uses Dynamic GC Threads	No
GC Time Ratio	99

Heap Configuration

Initial Heap Size	4.00 GB
Minimum Heap Size	4.00 GB
Maximum Heap Size	4.00 GB
If Compressed Oops Are Used	Yes
Compressed Oops Mode	zero based Compressed Oops
Heap Address Size	32
Object Alignment	8 bytes

Young Generation Configuration

Minimum Young Generation Size	1.33 GB
Maximum Young Generation Size	1.33 GB
New Ratio	2
Initial Tenuring Threshold	7
Maximum Tenuring Threshold	6
TLABs Used	Yes
Mininum TLAB Size	2.00 kB
TLAB Refill Waste Limit	64 bytes

GC Times

Events Operative

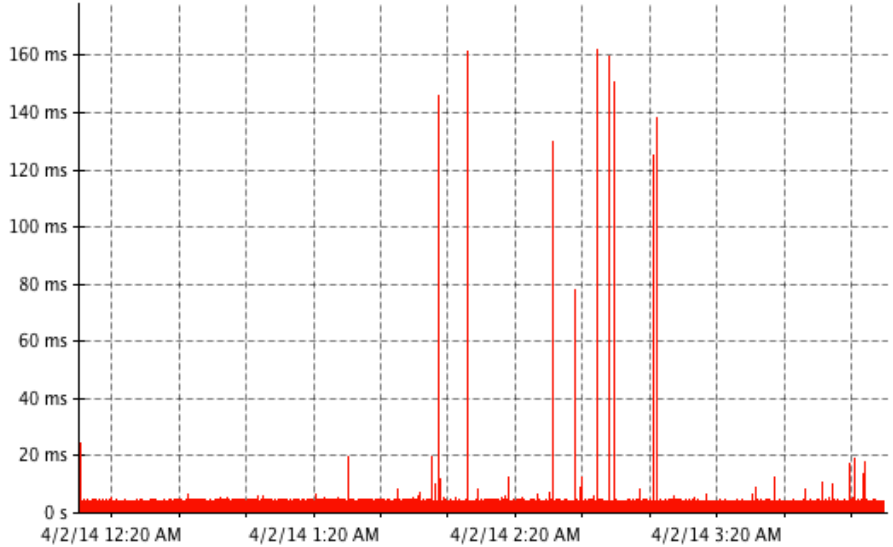
Interval: 4 h (all)

Synchronize Selection

4/2/14 12:09:59 AM

4/2/14 4:10:54 AM

GC Pauses



All Collections Pause Time

Sum of Pauses is the total time for all pauses during a GC

Average Sum of Pauses	4 ms 471 µs
Maximum Sum of Pauses	161 ms 942 µs
Total Pause Time	23 s 706 ms

GC Pauses per GC

Name	Longest Pause	Sum of Pauses
ParNew	161 ms 942 µs	161 ms 942 µs
ParNew	161 ms 510 µs	161 ms 510 µs
ParNew	159 ms 935 µs	159 ms 935 µs
ParNew	150 ms 991 µs	150 ms 991 µs
ParNew	145 ms 798 µs	145 ms 798 µs
ParNew	138 ms 452 µs	138 ms 452 µs
ParNew	129 ms 738 µs	129 ms 738 µs
ParNew	125 ms 29 µs	125 ms 29 µs
ParNew	77 ms 761 µs	77 ms 761 µs
ParNew	24 ms 245 µs	24 ms 245 µs
ParNew	22 ms 238 µs	22 ms 238 µs
ParNew	19 ms 906 µs	19 ms 906 µs

Young Collection Total Time

GC Count	5,302
Average GC Time	4 ms 471 µs
Maximum GC Time	161 ms 942 µs
Total GC Time	23 s 706 ms

Old Collection Total Time

GC Count	0
Average GC Time	N/A
Maximum GC Time	N/A
Total GC Time	N/A

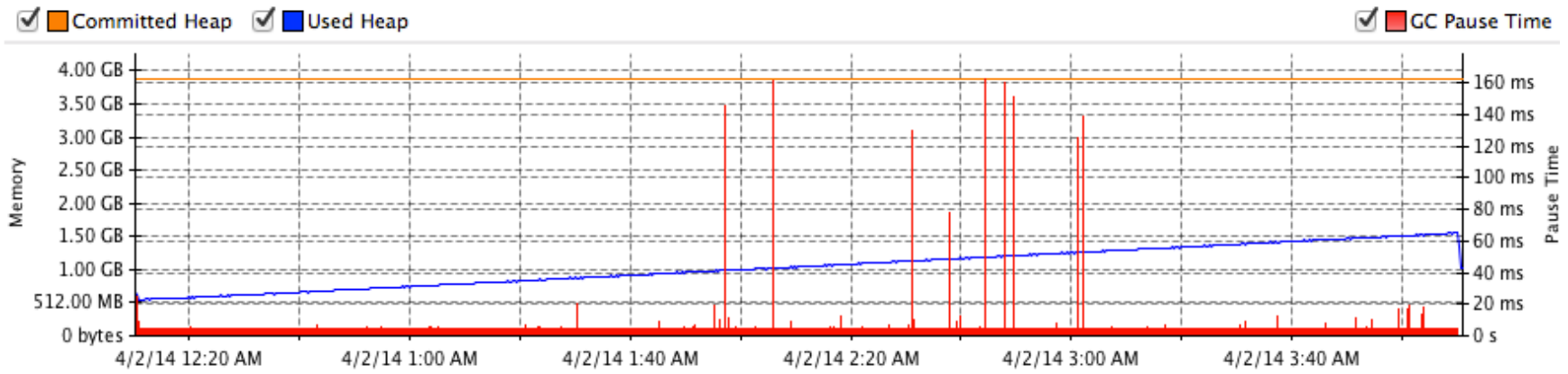
All Collections Total Time

GC Count	5,302
Average GC Time	4 ms 471 µs
Maximum GC Time	161 ms 942 µs
Total GC Time	23 s 706 ms

VOLTDB

Heap | Reference Objects | Failed Promotions | Failed Evacuations

Heap



Garbage Collections

GC ID	Longest Pause	Type	Conc Fail
3,431	161 ms 942 µs	ParNew	No
2,590	161 ms 510 µs	ParNew	No
3,511	159 ms 935 µs	ParNew	No
3,541	150 ms 991 µs	ParNew	No
2,399	145 ms 798 µs	ParNew	No
3,823	138 ms 452 µs	ParNew	No
3,142	129 ms 738 µs	ParNew	No
3,803	125 ms 29 µs	ParNew	No
3,294	77 ms 761 µs	ParNew	No
6	24 ms 245 µs	ParNew	No
0	22 ms 238 µs	ParNew	No
1,815	19 ms 906 µs	ParNew	No
2,353	19 ms 583 µs	ParNew	No
5,115	19 ms 190 µs	ParNew	No

General | GC Phases | Reference Objects | Heap | Perm Gen

GC Phases

Event Type	Name	Duration	Start Time
GC Phase Pause	GC Pause	161 ms 942 µs	4/2/14 2:44:35.923...
GC Phase Pause Level 1	SoftReference	12 µs	4/2/14 2:44:36.085...
GC Phase Pause Level 1	WeakReference	8 µs	4/2/14 2:44:36.085...
GC Phase Pause Level 1	FinalReference	10 µs	4/2/14 2:44:36.085...
GC Phase Pause Level 1	PhantomReference	6 µs	4/2/14 2:44:36.085...
GC Phase Pause Level 1	JNI Weak Reference	4 µs	4/2/14 2:44:36.085...

Overview | Garbage Collections | GC Times | GC Configuration | Allocations | Object Statistics

Top Blocking Locks | Top Blocked Threads | Top Blocking Threads

Filter Column: Class

Class	Count	Average	Longest	Duration
java.lang.Object	27	80 ms 399 µs	180 ms 330 µs	2 s 170 ms

Stack Trace	Count	Duration
sun.nio.ch.FileChannelImpl.position()	20	1 s 653 ms
sun.nio.ch.EPollSelectorImpl.wakeup()	7	517 ms 760 µs
org.voltcore.network.VoltNetwork.queueTask(Runnable)	6	356 ms 999 µs
org.voltcore.network.VoltPort.queueTask(Runnable)	6	356 ms 999 µs
org.voltcore.network.NIOWriteStream.fastEnqueue(DeferredSerialization)	6	356 ms 999 µs
org.voltcore.network.VoltNetwork.addToChangeList(VoltPort, boolean)	1	160 ms 761 µs
org.voltcore.network.VoltPort.setInterests(int, int)	1	160 ms 761 µs

Overview | Hot Threads | Contention | Latencies | Thread Dumps | Lock Instances

By Host | By Thread | By Event

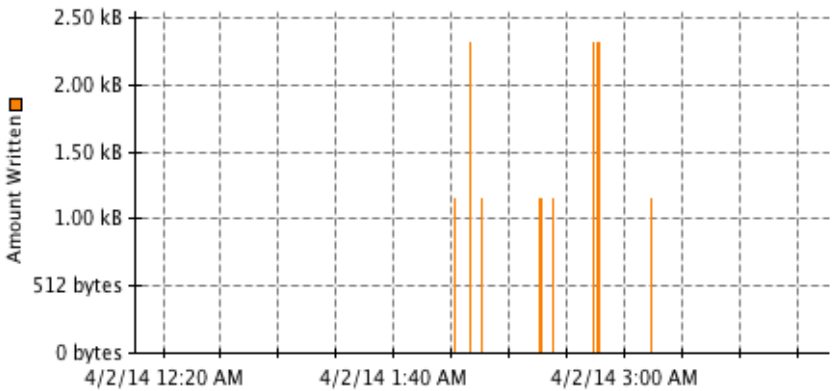
Socket Writes by Host

Filter Column: Remote Address

Remote Address	Remote Host	Number of Ports	Duration	Bytes Written	Number of Writes
10.10.180.161		1	1 s 61 ms	16.20 kB	10
10.10.180.162	volt10b	1	876 ms 967 µs	16.34 kB	8
10.10.180.164		1	867 ms 339 µs	10.68 kB	8

Socket Writes over Time

Socket Writes over Time for selected hosts



Socket Write Trace Tree

Stack Trace	Count
sun.nio.ch.SocketChannelImpl.write(ByteBuffer)	10
org.voltcore.network.PicoNIOWriteStream.drai...	10

Overview | File Read | File Write | Socket Read | Socket Write

By File | By Thread | By Event

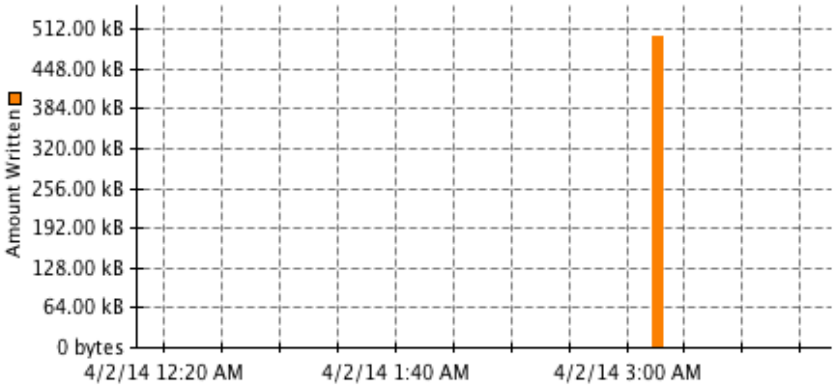
File Writes by File

Filter Column: Path

Path	Duration	Bytes Written	Number of Writes
/mnt/raid0-4hda-7k/test/five9s/snap/1396422574555-STORE-host_2.vpt	29 s 192 ms	98.42 MB	203
/mnt/raid0-4hda-7k/test/five9s/snap/1396423967451-STORE-host_2.vpt	27 s 410 ms	96.99 MB	201
/mnt/raid0-4hda-7k/test/five9s/snap/1396420740316-STORE-host_2.vpt	26 s 968 ms	94.49 MB	195
/mnt/raid0-4hda-7k/test/five9s/snap/1396424187639-STORE-host_2.vpt	26 s 206 ms	96.46 MB	200
/mnt/raid0-4hda-7k/test/five9s/snap/1396422354284-STORE-host_2.vpt	25 s 795 ms	95.50 MB	198
/mnt/raid0-4hda-7k/test/five9s/snap/1396419348088-STORE-host_2.vpt	25 s 205 ms	98.44 MB	203
/mnt/raid0-4hda-7k/test/five9s/snap/1396420960546-STORE-host_2.vpt	25 s 125 ms	96.43 MB	199
/mnt/raid0-4hda-7k/test/five9s/snap/1396417730922-STORE-host_2.vpt	24 s 846 ms	96.95 MB	201
/mnt/raid0-4hda-7k/test/five9s/snap/1396425579888-STORE-host_2.vpt	24 s 387 ms	95.04 MB	196
/mnt/raid0-4hda-7k/test/five9s/snap/1396416112027-STORE-host_2.vpt	23 s 921 ms	97.98 MB	203
/mnt/raid0-4hda-7k/test/five9s/snap/1396417510192-STORE-host_2.vpt	23 s 689 ms	95.99 MB	198

File Writes over Time

File Writes over Time for selected threads



File Write Trace Tree

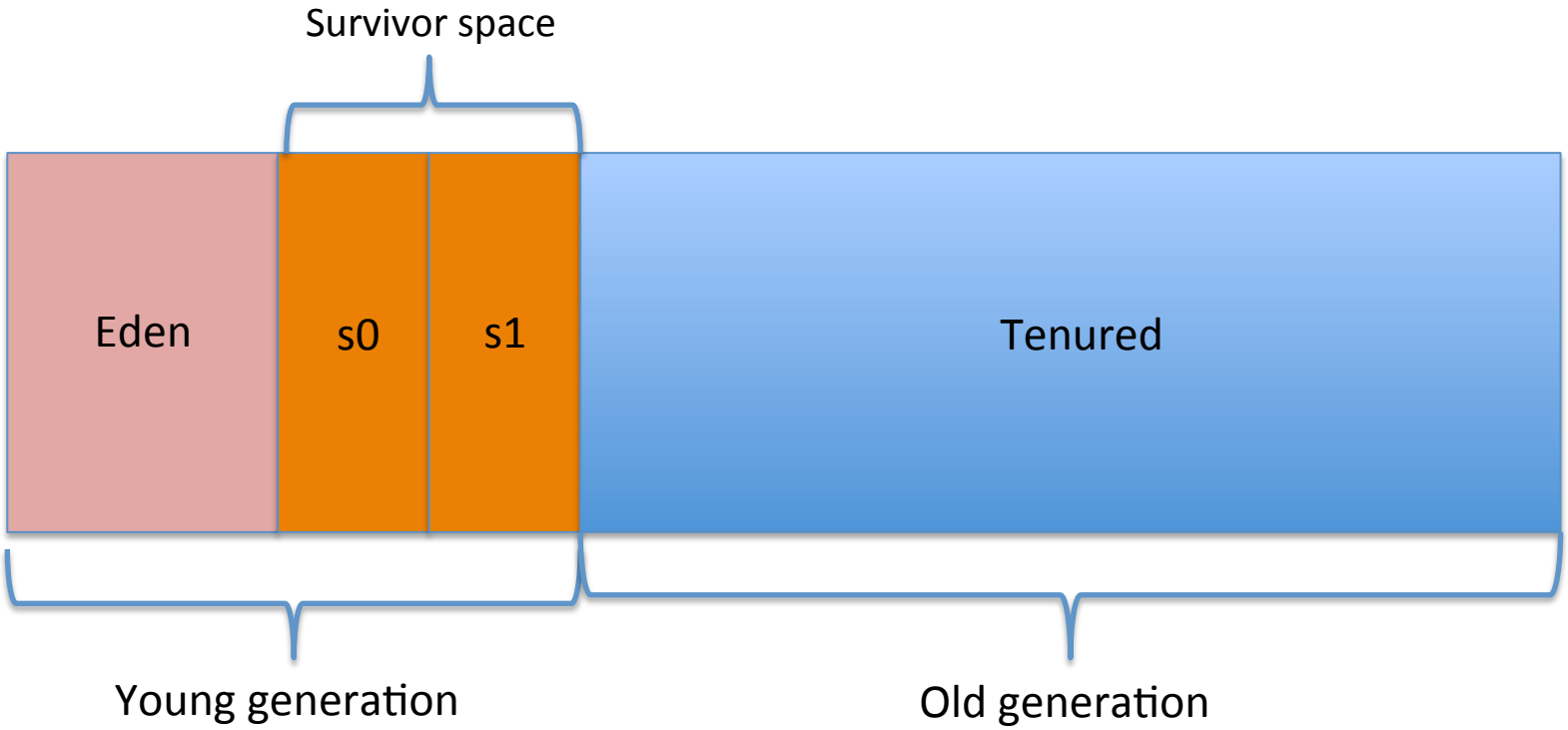
Stack Trace	Count
sun.nio.ch.FileChannelImpl.write(ByteBuffer)	203
org.voltdb.DefaultSnapshotDataTarget$3.call()	203
java.util.concurrent.FutureTask.run()	203
java.util.concurrent.ThreadPoolExecutor...	203

Overview | File Read | File Write | Socket Read | Socket Write

Segue into GC

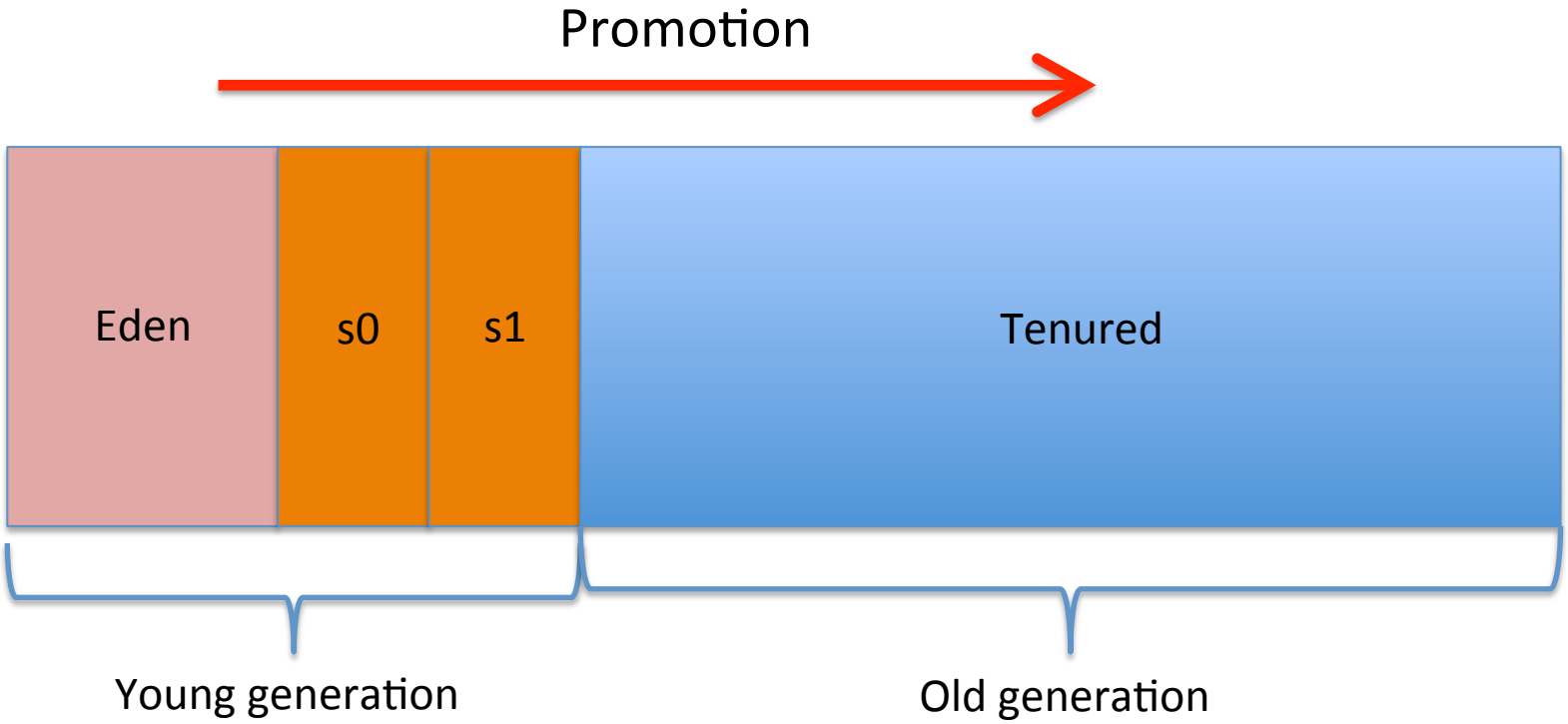
- Weak Generational Hypothesis
- Most objects soon become unreachable
- Few references from old to young objects
- Exploit by using different strategies for young and old

Segue into GC



VOLTDB

Segue into GC



VOLTDB

Segue into GC

- Previously mentioned GC expectations
  - Old-gen 10s of milliseconds every number of days
  - Young gen single digit milliseconds
- You make GC fast by
  - Not asking it do stuff
  - Not asking it do stuff it's bad at

Old-gen GC

- Doesn’t meet expectations of in-memory database
  - Pauses are too long (10s of milliseconds)
  - Fragmentation leads to periodic pauses that are orders of magnitude worse
- Azul Zing ostensibly offers an alternative
  - Deployment friction
  - Sales friction
- VoltDB has to work awesome on Oracle JDK

VOLTDB

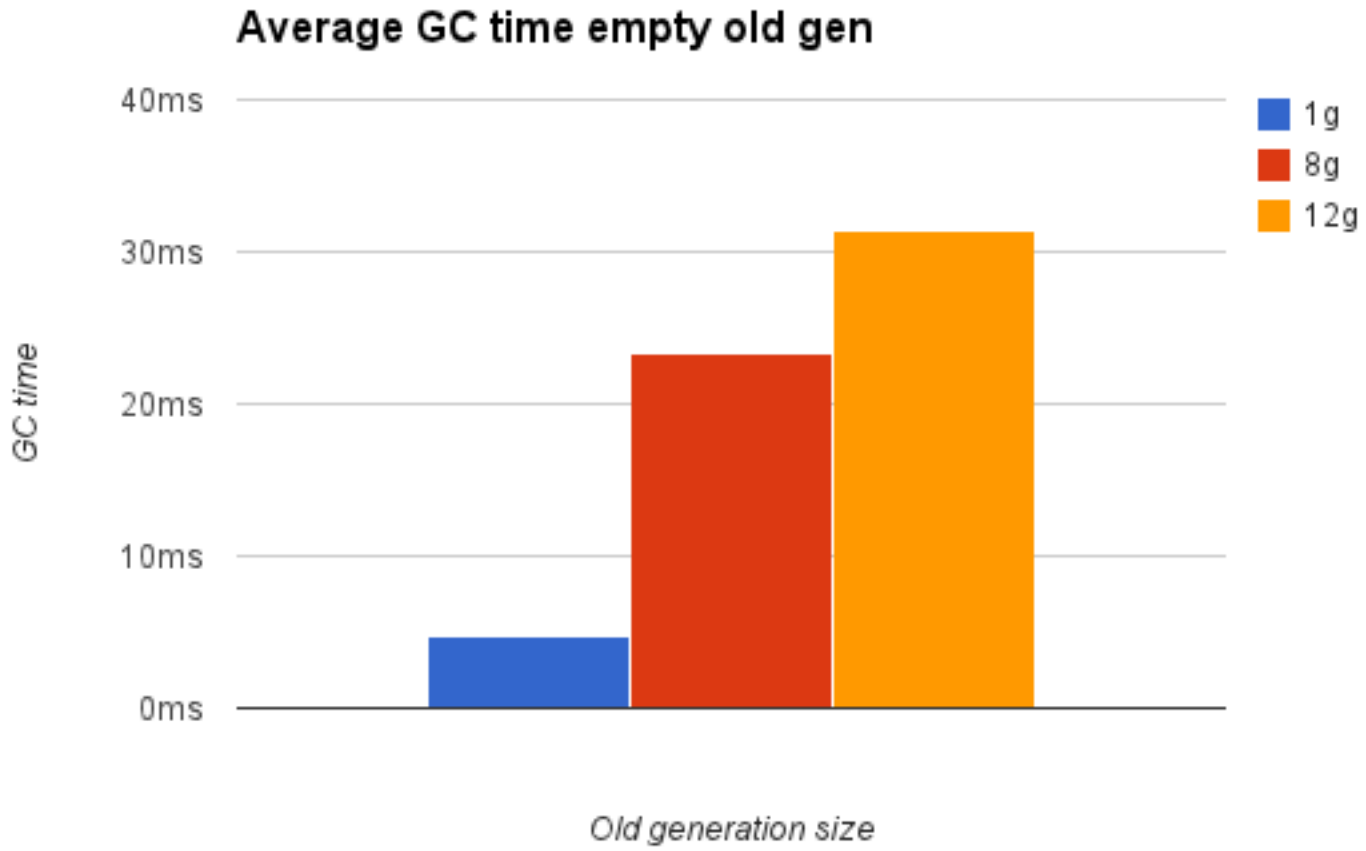
Old-gen GC

- Only use old-gen for small && infrequently changing data
  - For most code this preserves the goodness of Java and GC
- Run a small old-gen
- Anything that might get promoted goes off heap
  - Memory-mapped file
  - Direct byte buffer
  - Unsafe

VOLTDB

YoungGenPunisher

- pastebin.com/Sw27Wf7D
- Single threaded
- Allocate random size arrays 0-1024 bytes
- Stored map of linked lists by size
- Queue used to remove allocations in FIFO order
- Vary # of allocations retained
- The next graph uses CMS, the rest are parallel

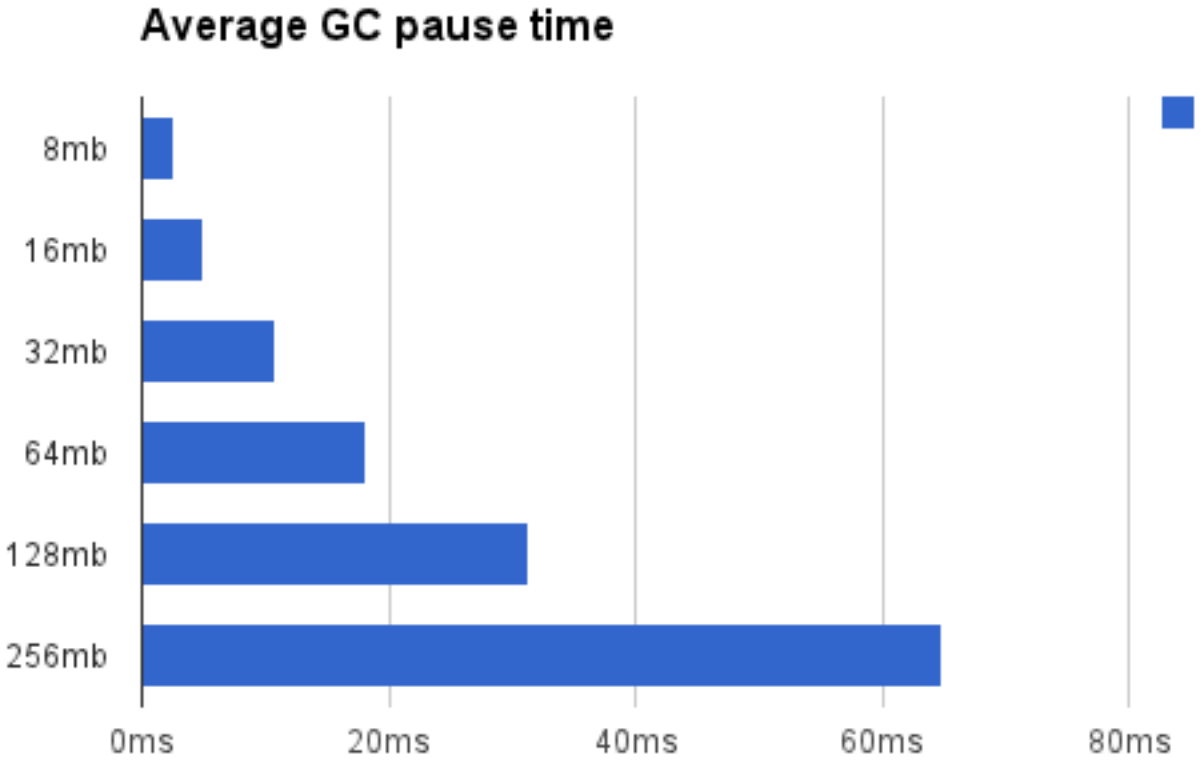

Average GC time empty old gen
40ms
30ms
20ms
10ms
0ms
GC time
1g
8g
12g
Old generation size

VOLTDB

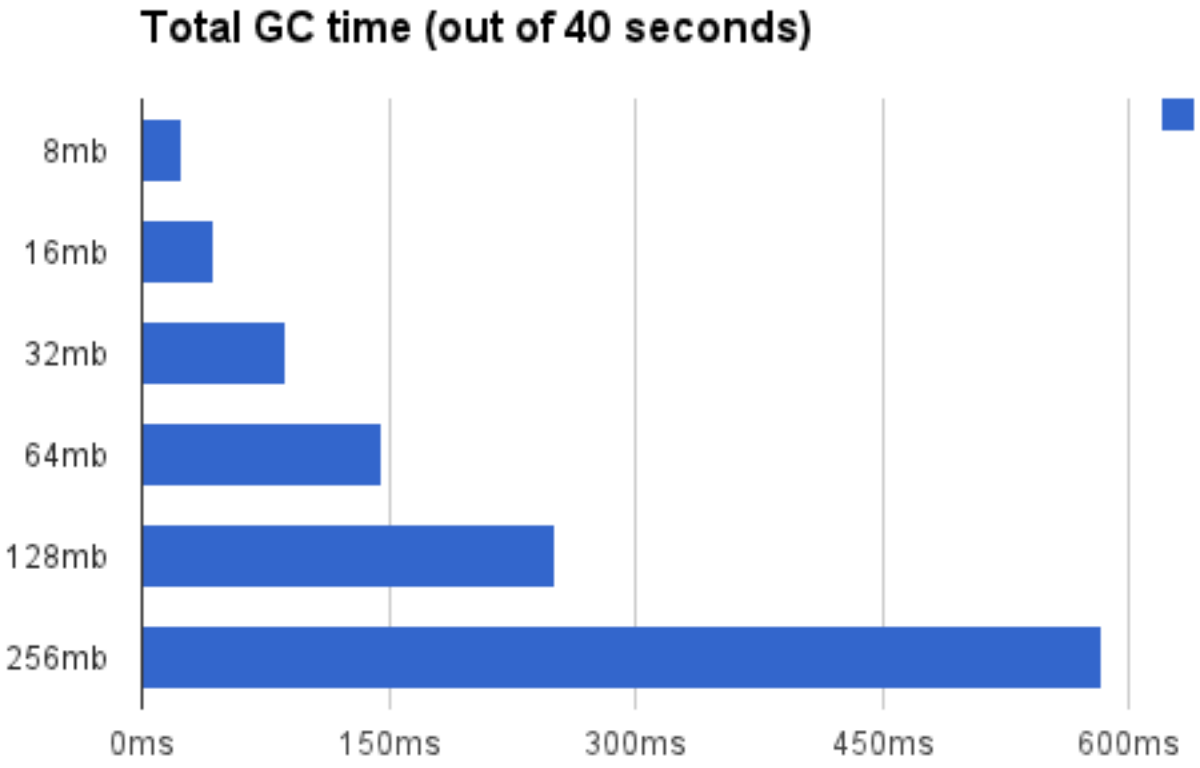
Young-gen GC

- http://goo.gl/clSi6l - Alexey Ragozin
- Copying collector
- Pay for
  - Copying live objects
  - Card scanning
  - Old gen scanning
  - Stack scan

VOLTDB


Average GC pause time
8mb
16mb
32mb
64mb
128mb
256mb
0ms
20ms
40ms
60ms
80ms

VOLTDB

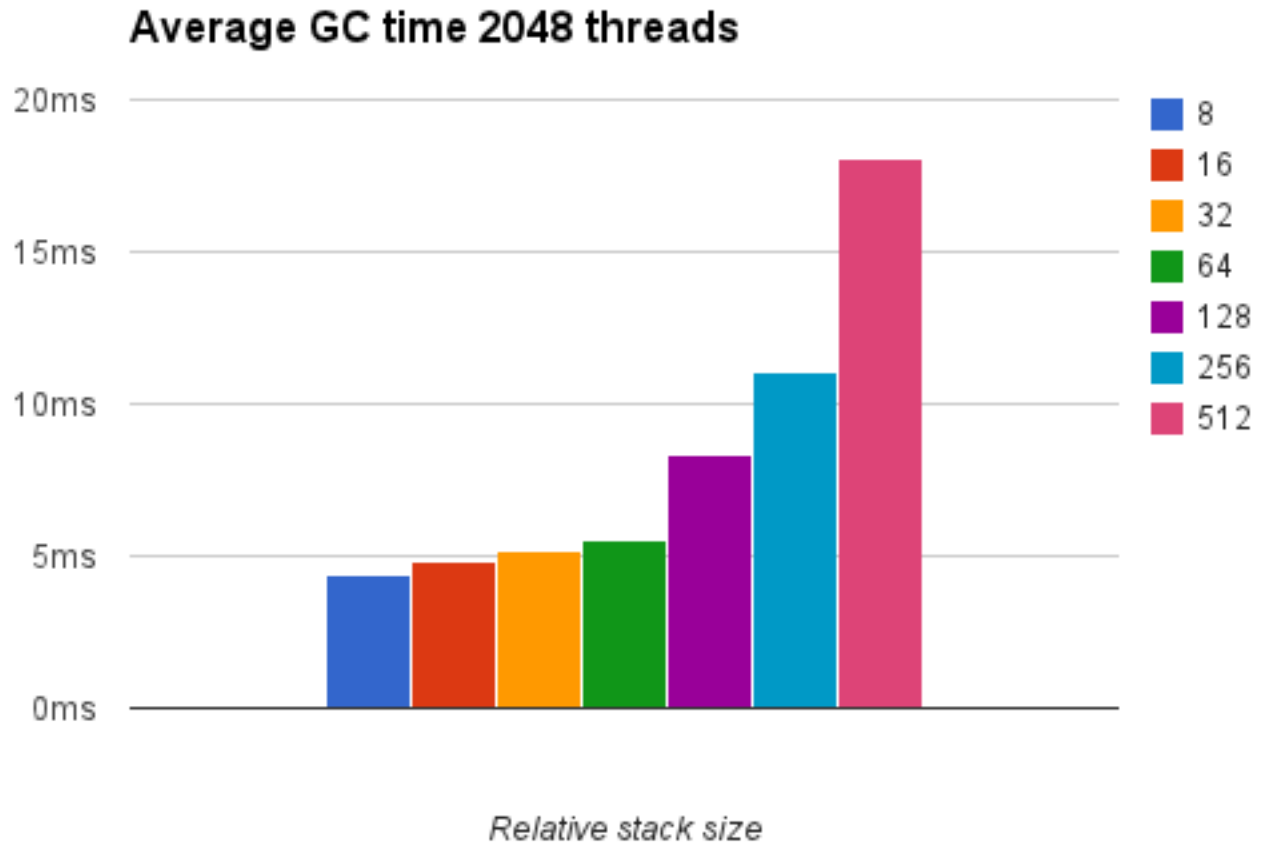

Total GC time (out of 40 seconds)
8mb
16mb
32mb
64mb
128mb
256mb
0ms
150ms
300ms
450ms
600ms

VOLTDB

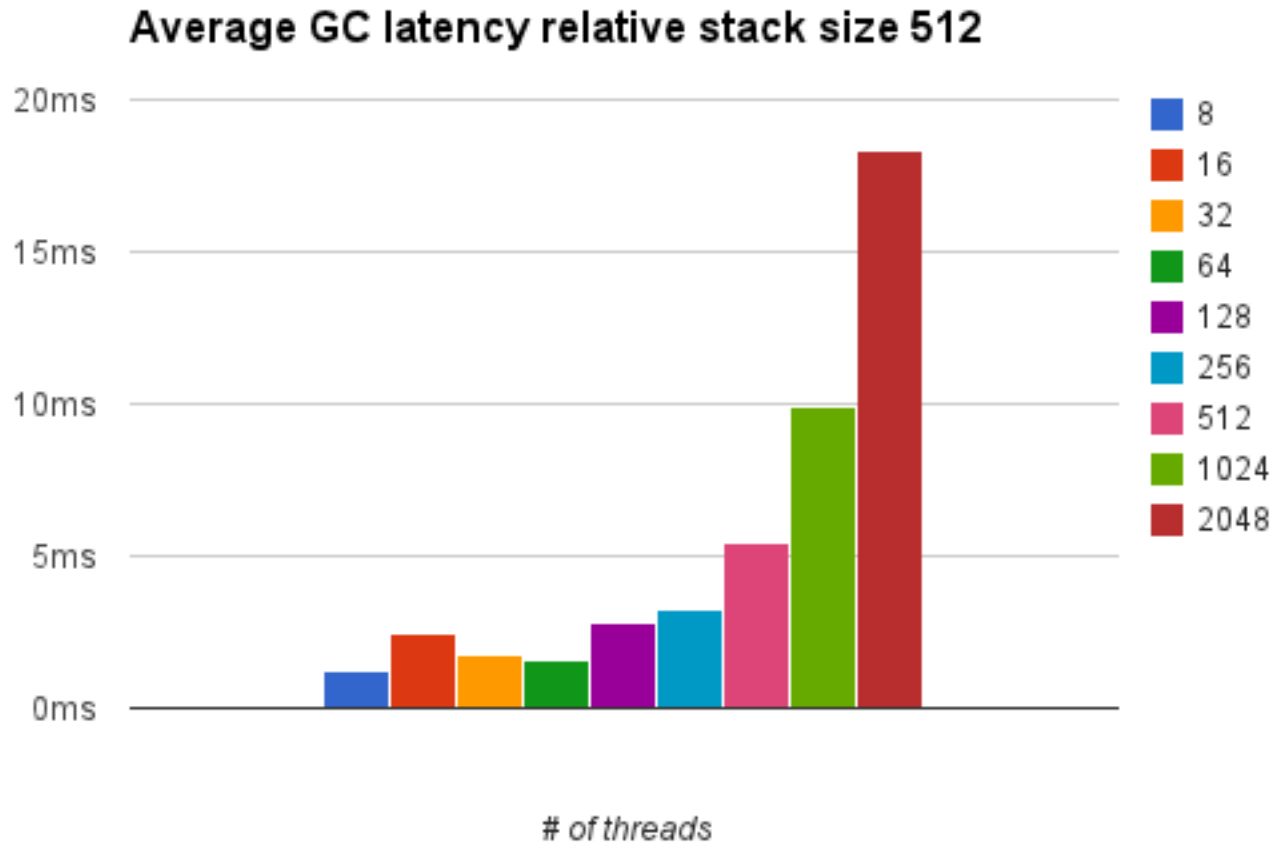
StackScanPunisher

- pastebin.com/JbqfVqvv
- Create variable number of sleeping threads
- Threads sleep at the top of a variable size stack
- Stack is grown through recursive method calls
- Main thread allocates memory to trigger GC
- relative stack size != recursion depth

VOLTDB


Average GC time 2048 threads
20ms
15ms
10ms
5ms
0ms
8
16
32
64
128
256
512
Relative stack size

VOLTDB


Average GC latency relative stack size 512
20ms
15ms
10ms
5ms
0ms
8
16
32
64
128
256
512
1024
2048
# of threads

VOLTDB

GC guided design

- Modest # of threads
- For young-gen, smallest working set possible
  - Reduce copying overhead
- Small old generation
  - Reduce card scanning overhead
- Working set size hardest to control
- Requires short running tasks, small # of tasks

VOLTDB

Going off heap

- VoltDB storage was always in C++
- Persistent Binary Deque
  - Larger than memory
  - Persistent
- Chronicle is similar
- Peter Lawrey – OpenHFT - http://goo.gl/dck2xm
  - Several off heap data structures
- Peter Lawrey – OpenJDK and HashMap … safely teaching an old dog new tricks - http://goo.gl/zq8y0p

Unsafe safe memory

- Off heap ByteBuffers rely on GC to de-allocate native memory
- We’re not cleaning the old gen remember?
- ((DirectBuffer)buf).cleaner().clean()
- -XX:MaxDirectMemorySize

VOLTDB

Unsafe safe memory

- There be no Valgrind here
- Track double free
- Use after free
- GC based leak detection using finalization
- Tag trails for debugging
- Conditional compilation to remove finalizer and fields

Keeping thread counts low

- Fundamentally VoltDB is thread per core
  - Shards are a core not a node
- Network is non-blocking so also low # of threads
- Additional threads
  - Shared timers
  - Threads as actors
  - Disk IO
- Event based design already meets this goal

I have the Event Loop tattoo

- Multi-plexing many tasks/roles onto a single thread
  - No context switch overhead/latency
  - Cache affinity, tasks execute where they most recently executed
  - Locking can be removed with careful design
- But
  - Skew necessitates work stealing
  - Now you're back to locking
  - And writing your own scheduler

VOLTDB

ListenableFuture, the event loop glue

- Available as CompletableFuture in Java 8
- Have a task
  - DB query
  - File IO
- Don’t want to block the current thread but need notification
  - Future.get() blocks
- ListenableFuture.addListener(Runnable, ExecutorService)

VOLTDB

So how about Linux?

- So far visibility has been poor
- Work around blocking system calls by using victim threads
  - In thread per core model blocking an event thread is bad
- Buffered IO isn't
  - Writes block in various conditions
- XFS and ext4 produce very different latency graphs
- Turn off transparent huge pages, zone_reclaim_mode = 0

VOLTDB

Enough with the environmentals

- Not all pauses are caused by the JVM or OS
- Visibility problem is the same
- Flight recorder
  - Not always available
  - Limited understanding of your application

System.nanoTime()

- Aleksey Davidovich – Nanotrusting nanotime - http://goo.gl/yoShNA
- 20-25 nanoseconds latency
- Similar granularity
- Linearly scalable on Linux
- Not always so scalable elsewhere
- If tasks are 10s or 100s of *us* this is cheap

VOLTDB

Canaries

- Poor person's flight recorder
- Or maybe the precise person?
- We assert on correctness, why not performance
- Outliers are typically things that are violating expectations
- So code up your expectations
  - How long to open/close that file, list directory etc.
  - Allocate memory
  - Read/write file
  - Acquire lock, block on condition

Canaries

```
//Canary has static final field for enabling/disabling
//JIT can optimize away Canary
long start = Canary.start()
// Do  work
Canary.finish(start, message, expectedTime, TimeUnit)
```

Latency numbers every programmer should know

```
L1 cache reference ..........................  0.5 ns
Branch mispredict ............................    5 ns
L2 cache reference ...........................    7 ns
Mutex lock/unlock ...........................   25 ns
Main memory reference ......................  100 ns
Compress 1K bytes with Zippy ............. 3,000 ns = 3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns = 20 µs
SSD random read ........................ 150,000 ns = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns = 250 µs
Round trip within same datacenter ......  500,000 ns = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns = 1 ms
Disk seek ........................... 10,000,000 ns = 10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns = 150 ms
```

https://gist.github.com/jboner/2841832

What about variance

- SSD operations can have several millisecond outliers
- Disks have even worse outliers, hundreds of milliseconds
- Disk schedulers have even worse outliers, seconds
- Packet loss TCP, latency of hundreds of milliseconds
- Requesting memory can have serious outliers
  - Kernel
  - JVM

VOLTDB

Watch dogs

- Thread pets the dog
- Dog registers thread, expects petting at specific frequency
- Thread is tardy, dog bites thread, Thread drops stack trace
  - Stack trace is at a safe point
- Works best with steady synthetic workloads
- Can also be compiled out by the JIT using static final

VOLTDB

Rate Limited Logger

- Canaries and Watch Dogs produce TMI
  - Outliers may persist or cluster
- Rate limit log messages based on format string
- Enforcement is fast and linearly scalable
- Requires a hash lookup of format string
- Cache line of last log time enters shared state

Debugging production

- VoltDB is deployed by others
- No control of hardware, software, network
- Difficult to acquire monitoring artifacts
- Send us the logs has highest odds of success
- Helpful for one off or non-reproducible events to trigger canaries

Mesh Monitor

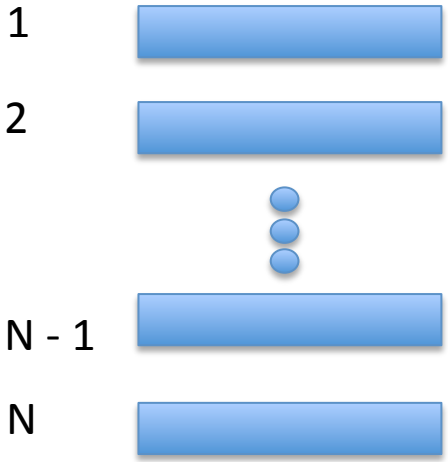
- Inspired by jHiccup – Gil Tene – azulsystems.com/jHiccup
- Solves the distributed version of the jHiccup problem
- All processes connected via TCP mesh, heartbeating
- Measures w/Histogram
  - Accuracy of thread wakeup
  - Interval between receiving heartbeat
  - Delta between time heartbeat sent and heartbeat received
- Can differentiate node wide pauses from network pauses

How applications cause outliers

- Ignoring environmental causes
  - JVM
  - Linux
  - Contended IO devices
- Non-uniform task size
  - Actual task size is variable
  - Victim task(s) suffer from unrelated or deferred work

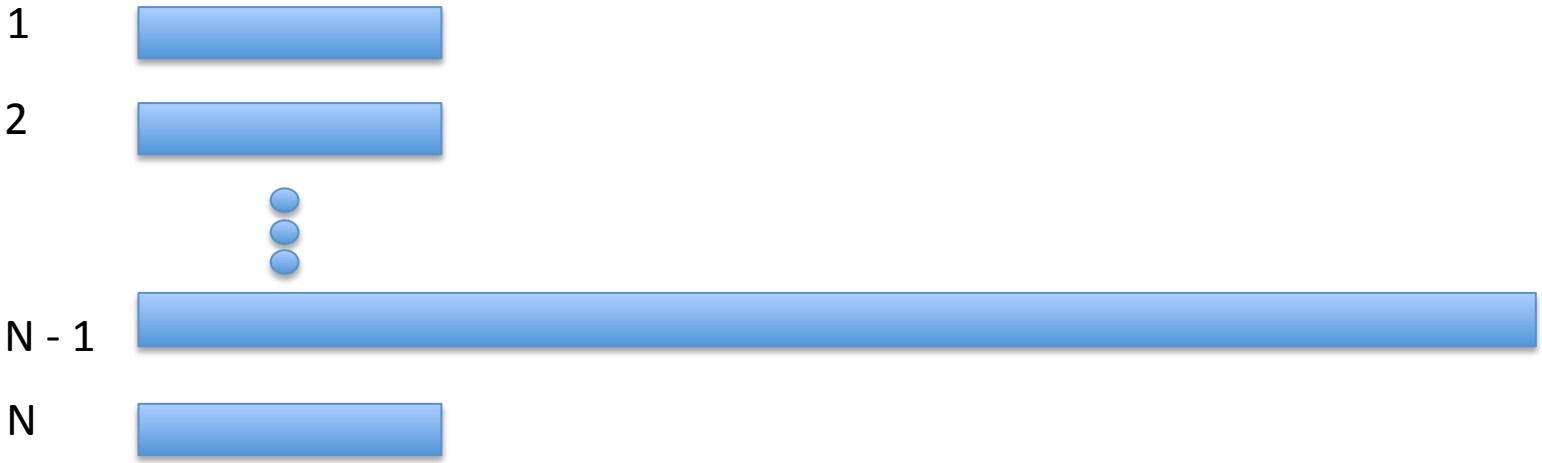
VOLTDB

Ideal task size



VOLTDB

Bad task sizes



VOLTDB

Large tasks

- Reduce task size
- Only need to be big enough to amortize startup cost
- Process concurrently where possible
- Limit # of outlier tasks that can be processed concurrently
- Enforce ratios or rate limits
  - Guava's Rate Limiter
  - VoltDB's Minimum Ratio Maintainer
- Being able to weigh tasks is helpful

VOLTDB

Bad task sizes

VOLTDB

Outlier tasks

- Victim tasks
  - Hash table resize
  - Cache expiration
  - Garbage collection
  - Flush to filesystem
- Same strategies as large tasks +

Outlier tasks

- Hash table resize
  - Use incremental resizing hash table, preallocate, use a tree
- Cache expiration
  - Expire/replace asynchronously
- Garbage collection
  - Do incremental collection with each task
- Flush to filesystem
  - Use a dedicated thread, preemptively prepare filesystem state

Getting to 5 9s

- Examining all 1 second periods
- Started with 5 9s at several hundred milliseconds
- Ended with 5 9s under 80 milliseconds
  - Even while rebuilding a failed node
- 5 9s for one hour is under 25 milliseconds
- 6 9s for one hour under 250 milliseconds
- No silver bullet

VOLTDB

Questions?

VOLTDB