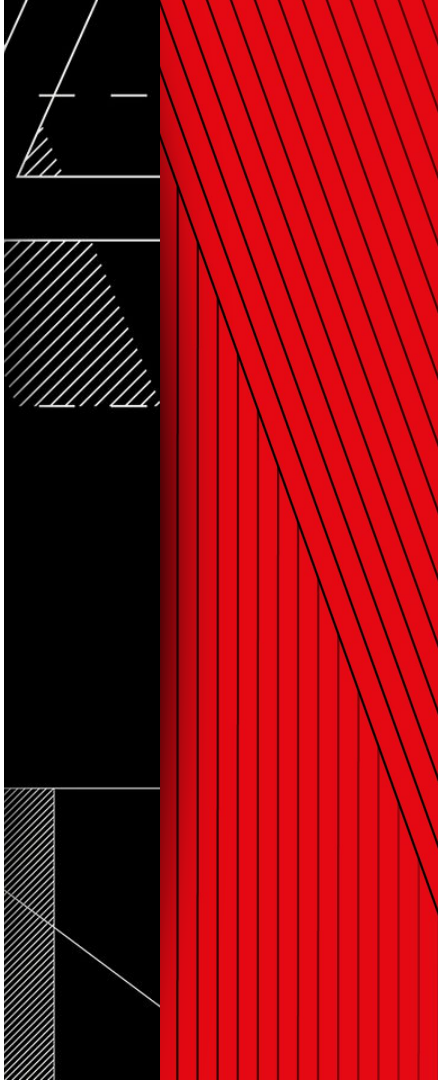
How Netflix Debugs and Fixes Apache Cassandra

... when it breaks

Joey Lynch



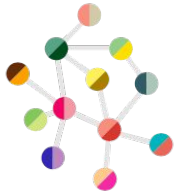
Speaker

Joey Lynch

Senior Software Engineer
Cloud Data Engineering at Netflix

Distributed system addict and data wrangler



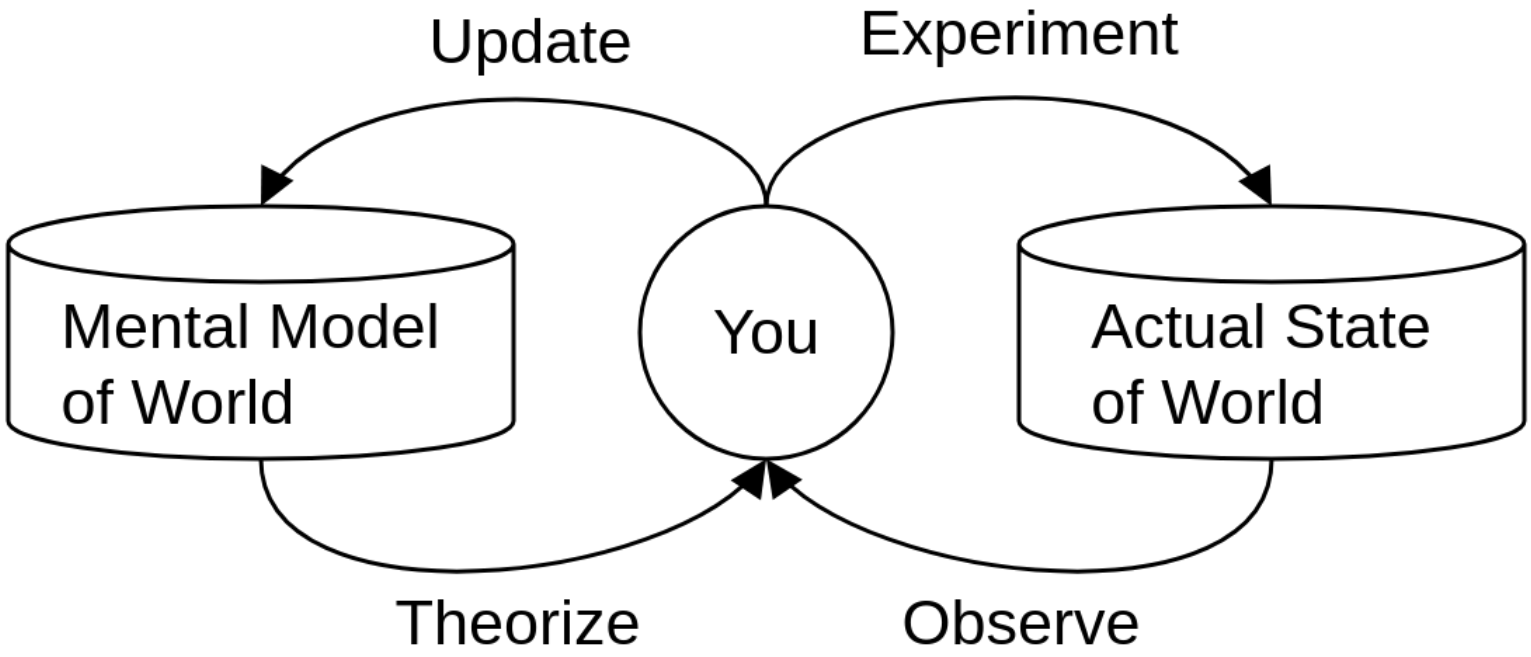



Debugging Methodology

Debugging is the practice of observing a system, building a mental model of the system, and then rapidly testing your newly formed mental model.

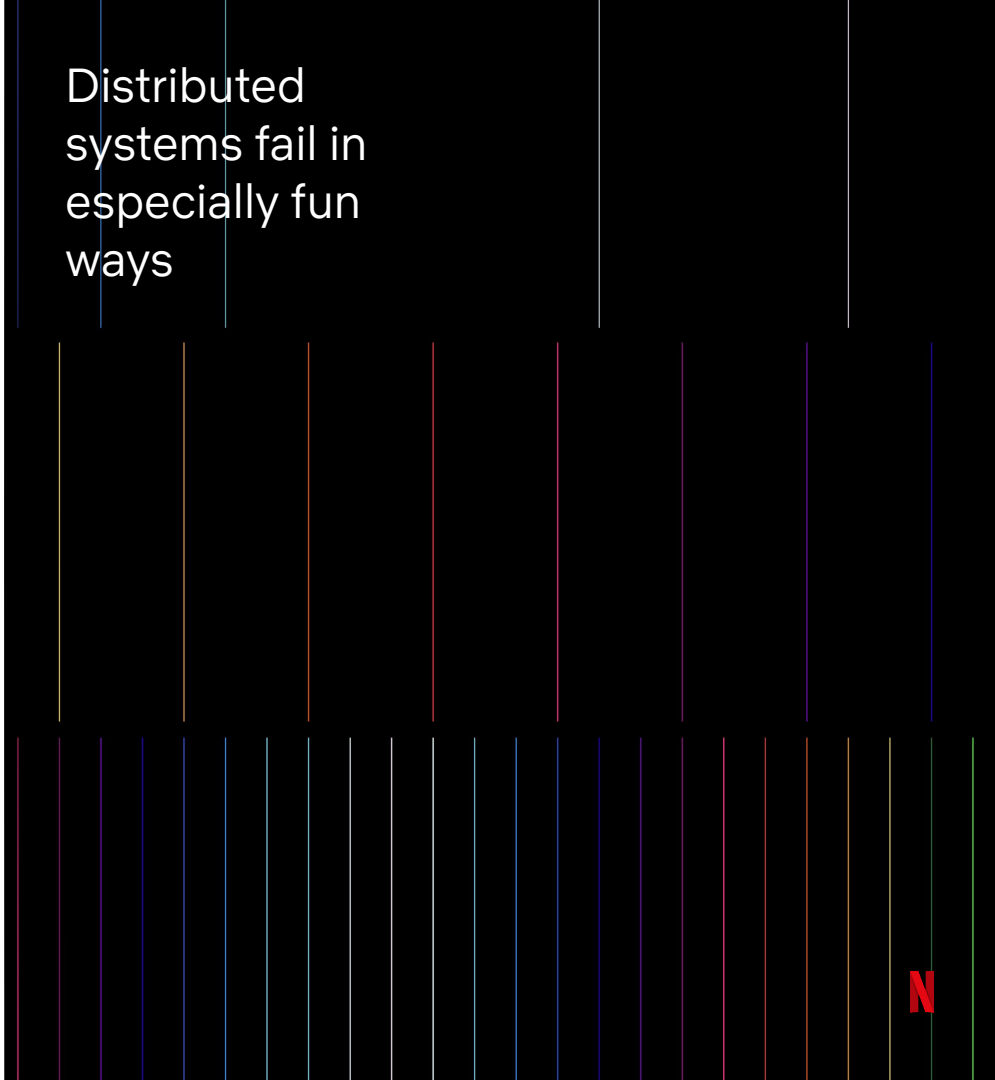
Typically while that system is on fire 🔥

Building Mental Models



Distributed Woes

Distributed systems fail in especially fun ways



“We see a large number of timeouts from Cassandra”

-Many a developer

Initial Theories

No actual problem

Degraded replica

Retry storm

Load shift

Action Plan

Initial Theories

No actual problem

Degraded replica

Retry storm

Load shift

Action Plan

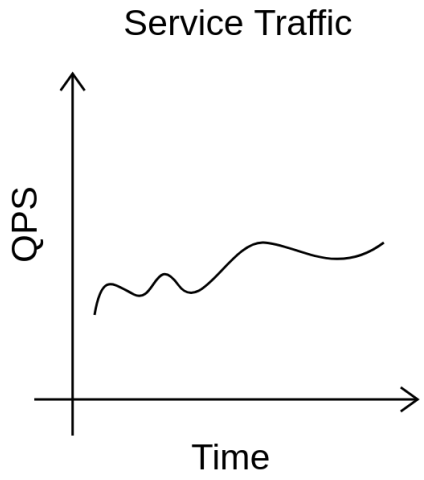
Observe metrics

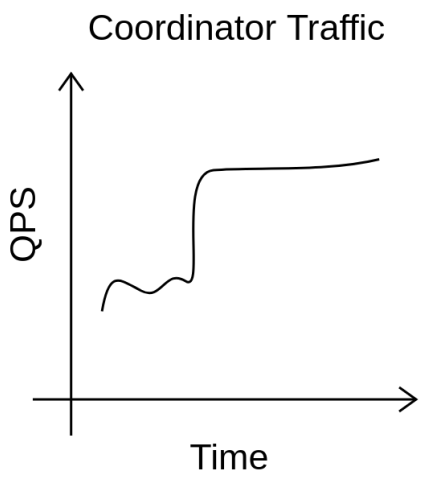
+

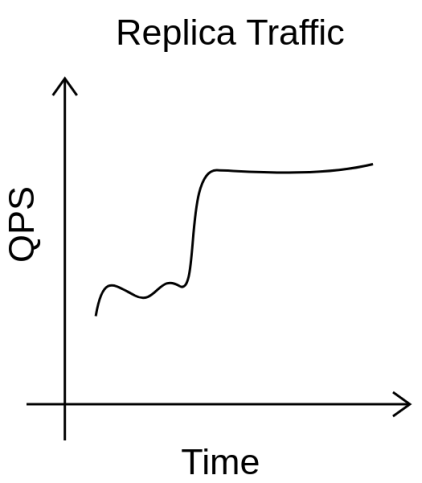
Read client logs, look for clues of timeouts

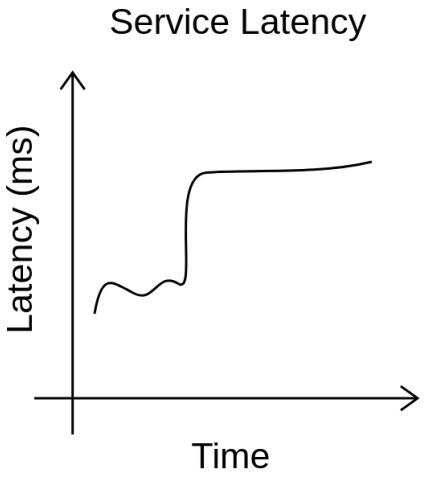
+

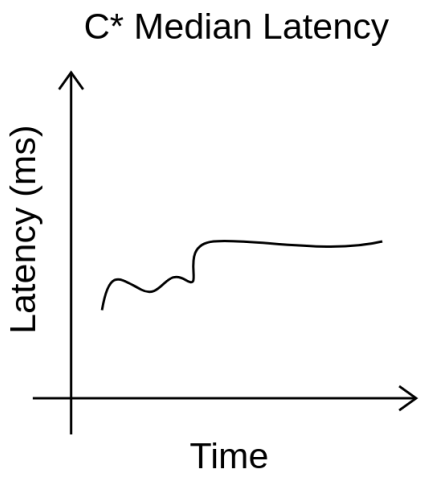
Read server logs, look for errors

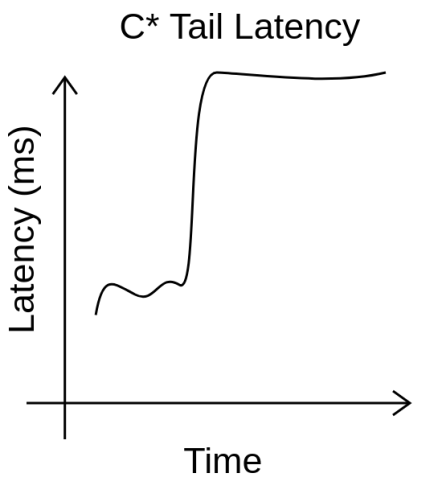

Service Traffic
QPS
Time

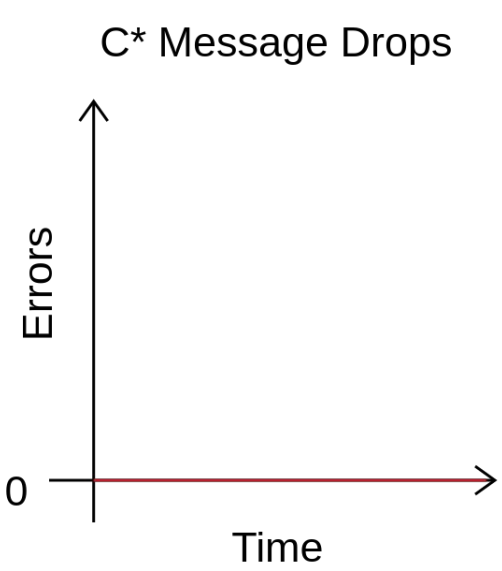

Coordinator Traffic
QPS
Time

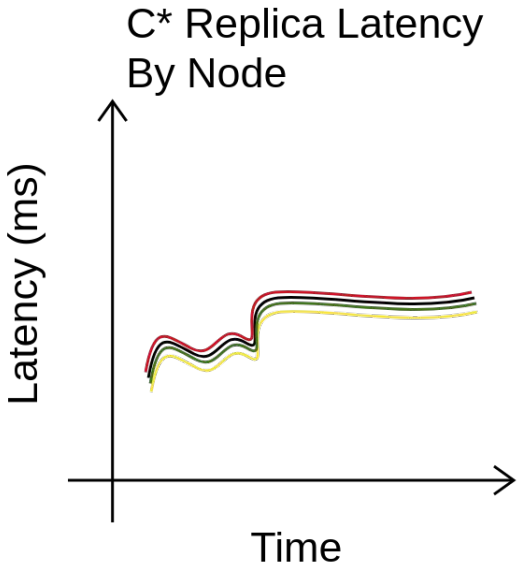

Replica Traffic
QPS
Time

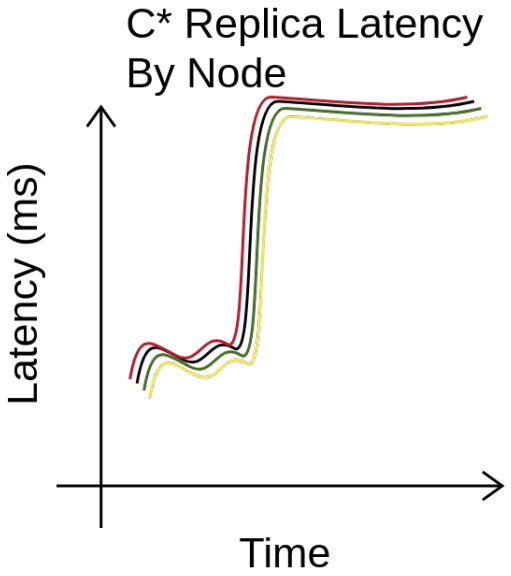

Service Latency
Latency (ms)
Time


C* Median Latency
Latency (ms)
Time


C* Tail Latency
Latency (ms)
Time


C* Message Drops
Errors
0
Time


C* Replica Latency
By Node
Latency (ms)
Time


C* Replica Latency
By Node
Latency (ms)
Time

```
$ grep Exception client.log | wc -l
1213  # not good
```

```
$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | tail
com.datastax.driver.core.exceptions.OperationTimedOutException
com.datastax.driver.core.exceptions.OperationTimedOutException

$ grep OperationTimedOutException client.log -C 5
OperationTimedOutException: [/IP:9042] Timed out waiting for server
response ...

$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | sort
| uniq -c
   1211  com.datastax.driver...OperationTimedOutException
      2  com.netflix...Exception
```

How many Errors are there?

```
$ grep Exception client.log | wc -l
1213  # not good

$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | tail
com.datastax.driver.core.exceptions.OperationTimedOutException
com.datastax.driver.core.exceptions.OperationTimedOutException
```

```
$ grep OperationTimedOutException client.log -C 5
OperationTimedOutException: [/IP:9042] Timed out waiting for server
response ...
```

Which Errors precisely are happening?

```
$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | sort
| uniq -c
   1211  com.datastax.driver...OperationTimedOutException
      2  com.netflix...Exception
```

```
$ grep Exception client.log | wc -l
1213  # not good

$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | tail
com.datastax.driver.core.exceptions.OperationTimedOutException
com.datastax.driver.core.exceptions.OperationTimedOutException

$ grep OperationTimedOutException client.log -C 5
OperationTimedOutException: [/IP:9042] Timed out waiting for server
response
```

```
$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | sort
| uniq -c
   1211  com.datastax.driver.core.exceptions.OperationTimedOutException
      2  com.netflix...Exception
```

More context

What is the error distribution?

```
$ grep Exception client.log | grep "Caused by" | cut -f 2 -d ':' | sort
| uniq -c
    1211  com.datastax.driver...OperationTimedOutException
       2  com.netflix...Exception
```

```
$ grep <time> server.log | egrep “WARN|ERROR” |
less
# … quickly skim … nothing interesting
```

Initial Theories	Reason
~~No actual problem~~	Latency and Exceptions
~~Degraded replica~~	Consistent latencies
Retry storm	Likely
Load shift	Plausible

What do we know?

There is a problem

Spike in C* traffic

Client side timeouts

No server timeouts or irregularities

Action Plan

Retry storm appears most likely, need to determine client timeout value.

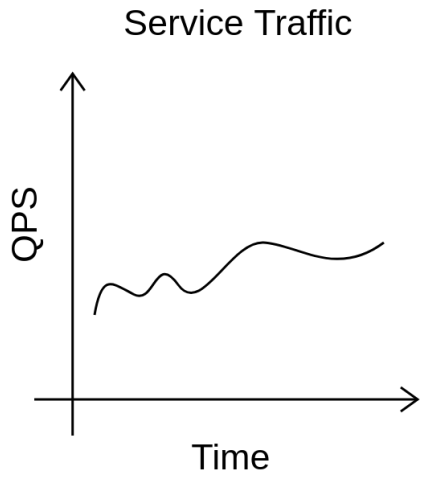
Property	Value
[redacted] aeneas.connectionReadTimeoutMillis	40

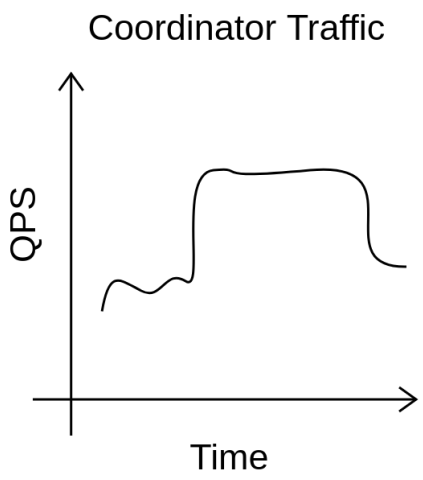
1. Timeout catalogs are convenient
2. 40ms is very low for a timeout
3. Datastax driver retries by default … a lot of times

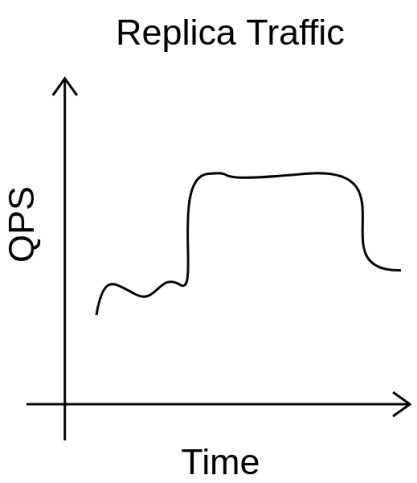
```
/**
 * {@inheritDoc}
 *
 * <p>This implementation rethrows read and write failures, and retries other errors on the next
 * node.
 */
@Override
public RetryDecision onErrorResponse(
    @NonNull Request request, @NonNull CoordinatorException error, int retryCount) {

  RetryDecision decision =
      (error instanceof ReadFailureException || error instanceof WriteFailureException)
          ? RetryDecision.RETHROW
          : RetryDecision.RETRY_NEXT;

  if (decision == RetryDecision.RETRY_NEXT && LOG.isTraceEnabled()) {
    LOG.trace(RETRYING_ON_ERROR, logPrefix, retryCount, error);
  }

  return decision;
}
```

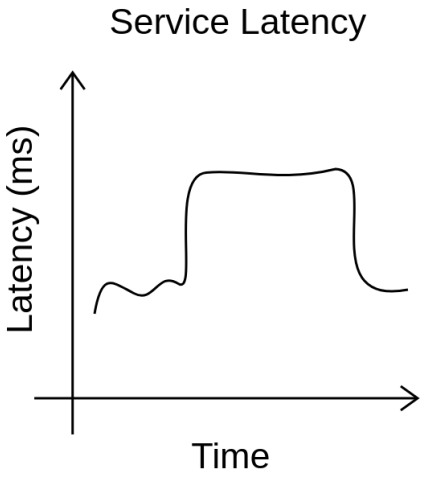
Why Even.

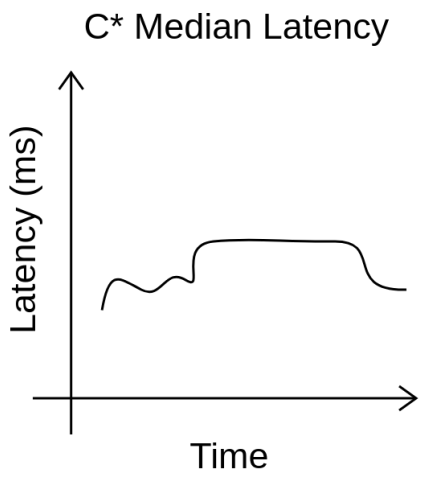
Proposed Experiment	Reasoning
Turn off retries	Retries are almost always wrong (except UnavailableExceptions)
Increase timeout to C* SLO of 100ms	Concurrency limited speculations are acceptable, but not commonly implemented.

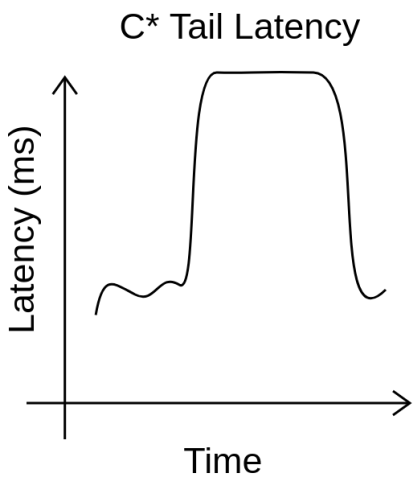

Service Traffic
QPS
Time


Coordinator Traffic
QPS
Time


Replica Traffic
QPS
Time


Service Latency
Latency (ms)
Time


C* Median Latency
Latency (ms)
Time


C* Tail Latency
Latency (ms)
Time

Automate

Stop retrying

— Turn off the retries!! 😀

- Fail fast instead! Inflict chaos! 😀

Automate

Stop retrying

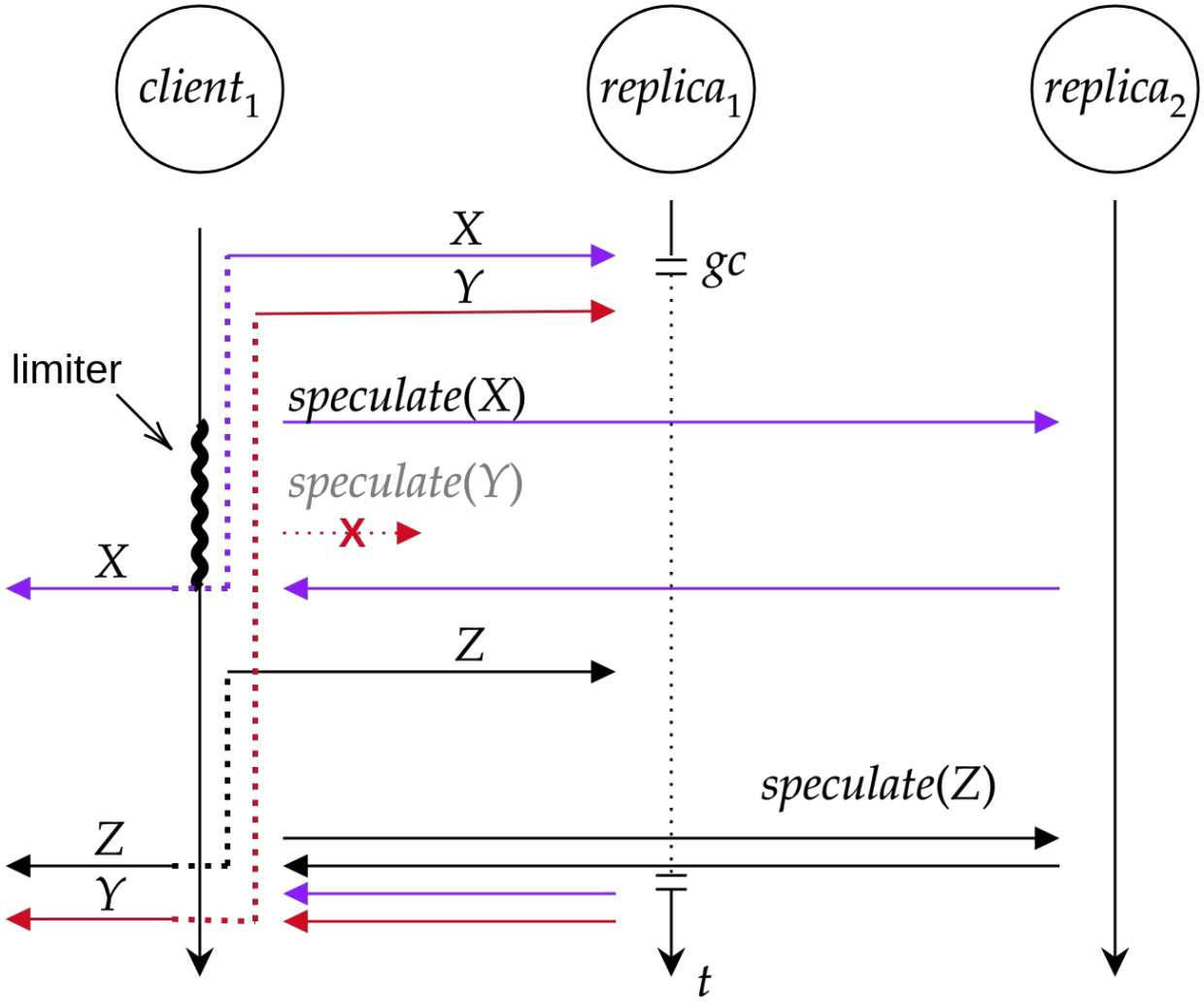
- Turn off the retries!! 😀
  - Fail fast instead! Inflict chaos! 😀
- Still want to retry? 😢
  - Concurrency limited p95 speculation instead?

Automate

Stop retrying

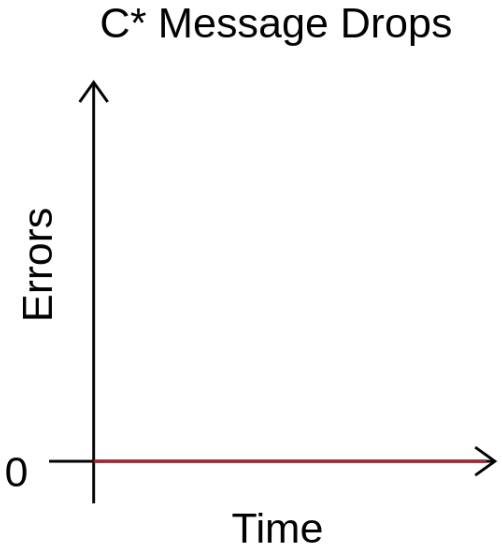
Or raise the timeout

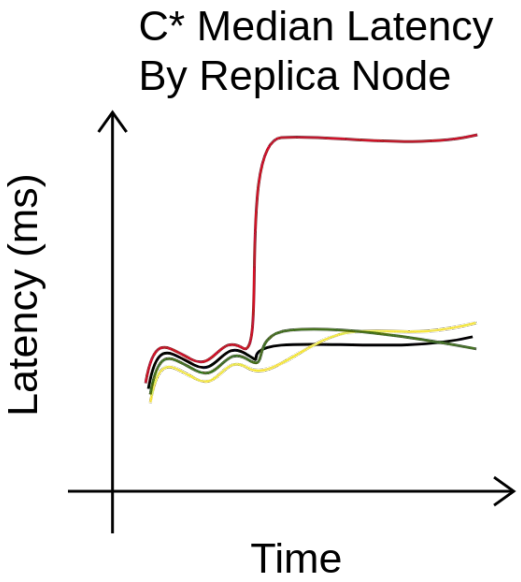
- Turn off the retries!! 😀
  - Fail fast instead! Inflict chaos! 😀
- Still want to retry? 😢
  - Concurrency limited p95 speculation instead?
- Too hard? 😢
  - Can you use exponential backoff?
  - CASSANDRA-15013+ will fix this server side ...

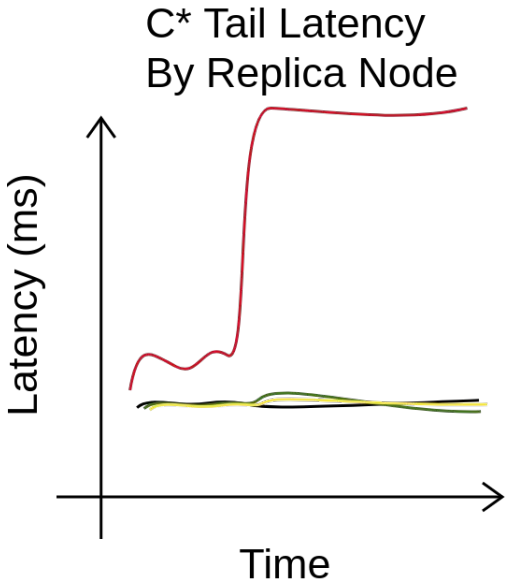


Concurrency limits prevent cascading failures

What if it wasn’t that easy?

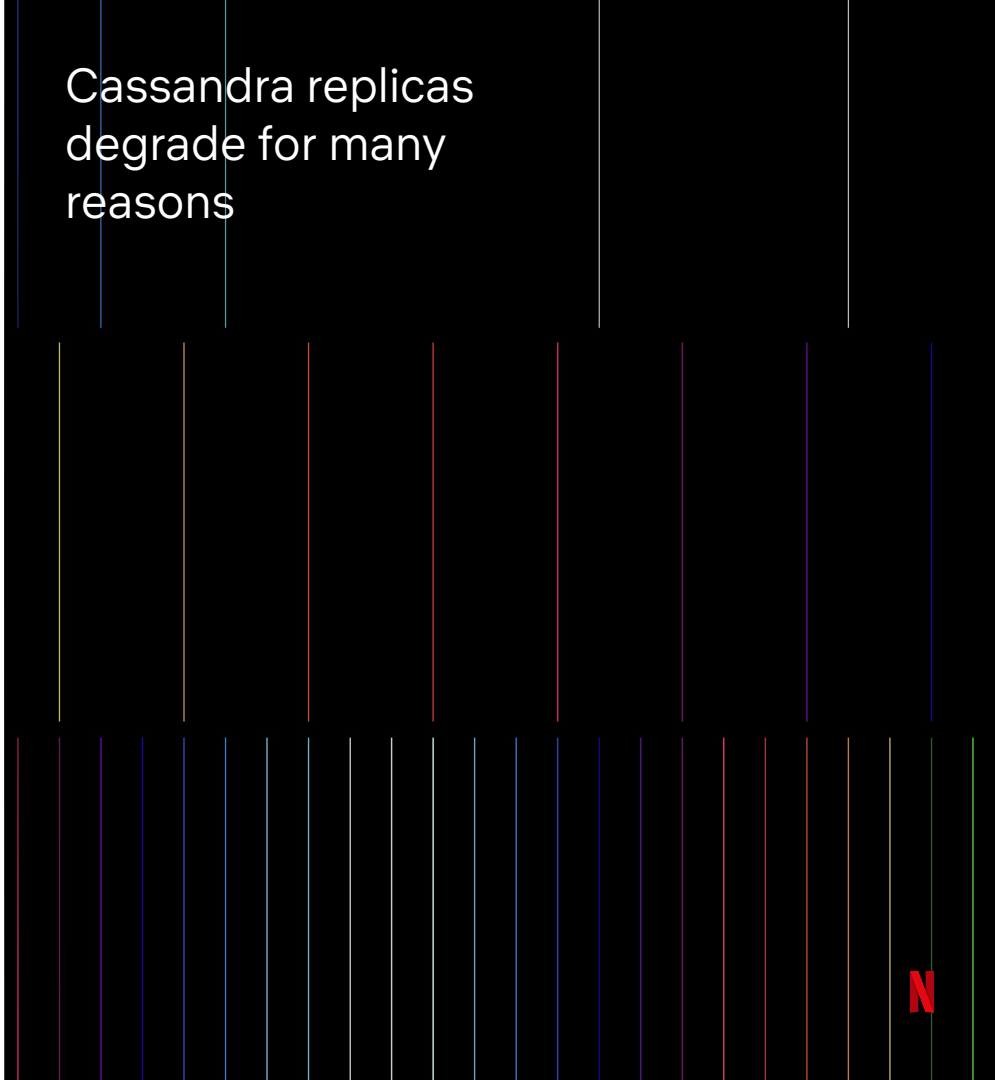

C* Message Drops
Errors
0
Time


C* Median Latency
By Replica Node
Latency (ms)
Time


C* Tail Latency
By Replica Node
Latency (ms)
Time

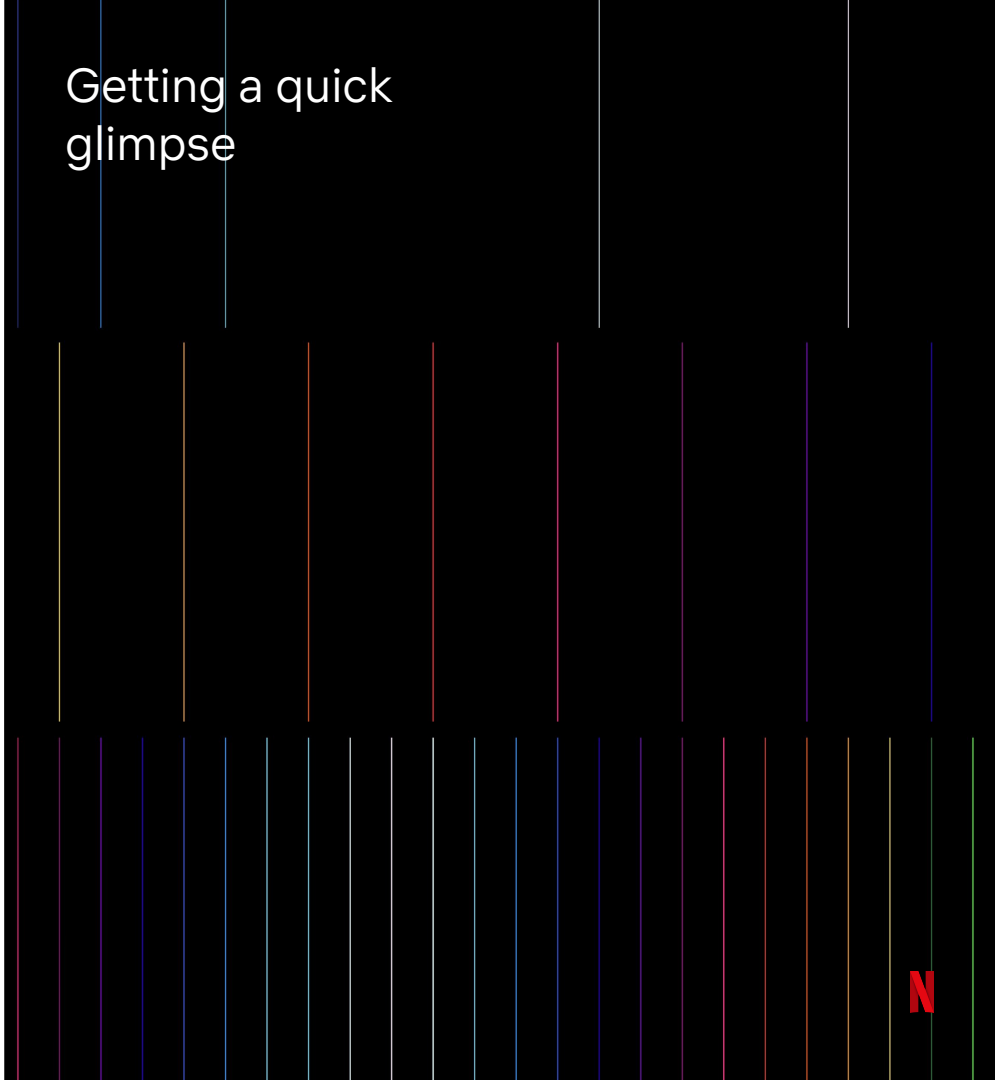
Ways Replicas Break

Cassandra replicas degrade for many reasons

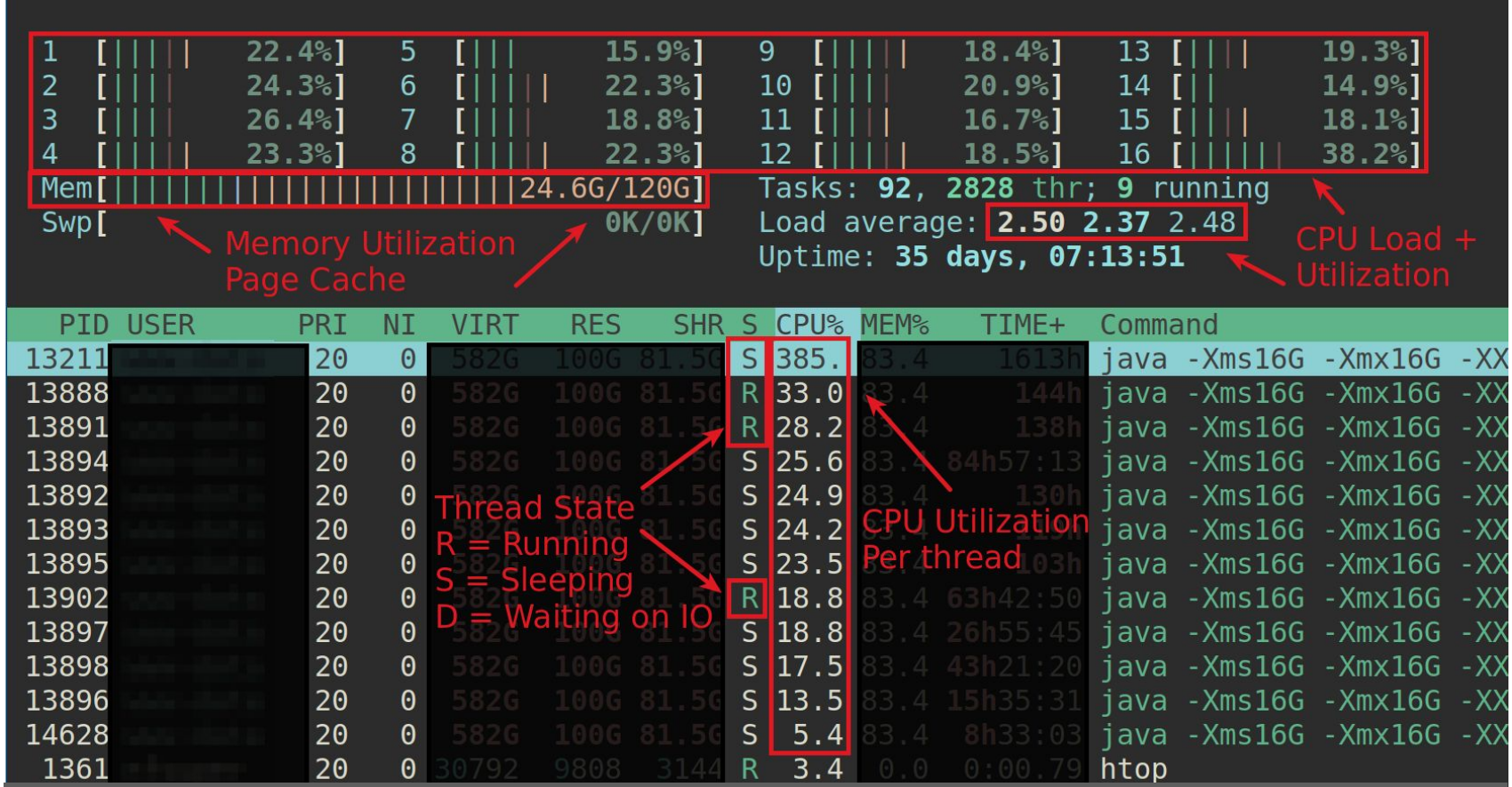


Lay of Land Tools

Getting a quick glimpse



```
# CPU resources
$ htop; top; schedstat; flamegraphs

# Filesystem
$ df; dmesg; mount; vmtouch; cachestat

# Drives
$ iostat; biosnoop; biolatency; dstat

# Network
$ netstat; iftop; tcpdump; wireshark

# JVM
$ jstat; gclog; gcviewer
```



What is the machine doing?

Disk Usage

```
$ df -h / /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda       9.8G  6.4G  3.4G  66% /
/dev/md0        3.5T  622G  2.9T  18% /mnt
```

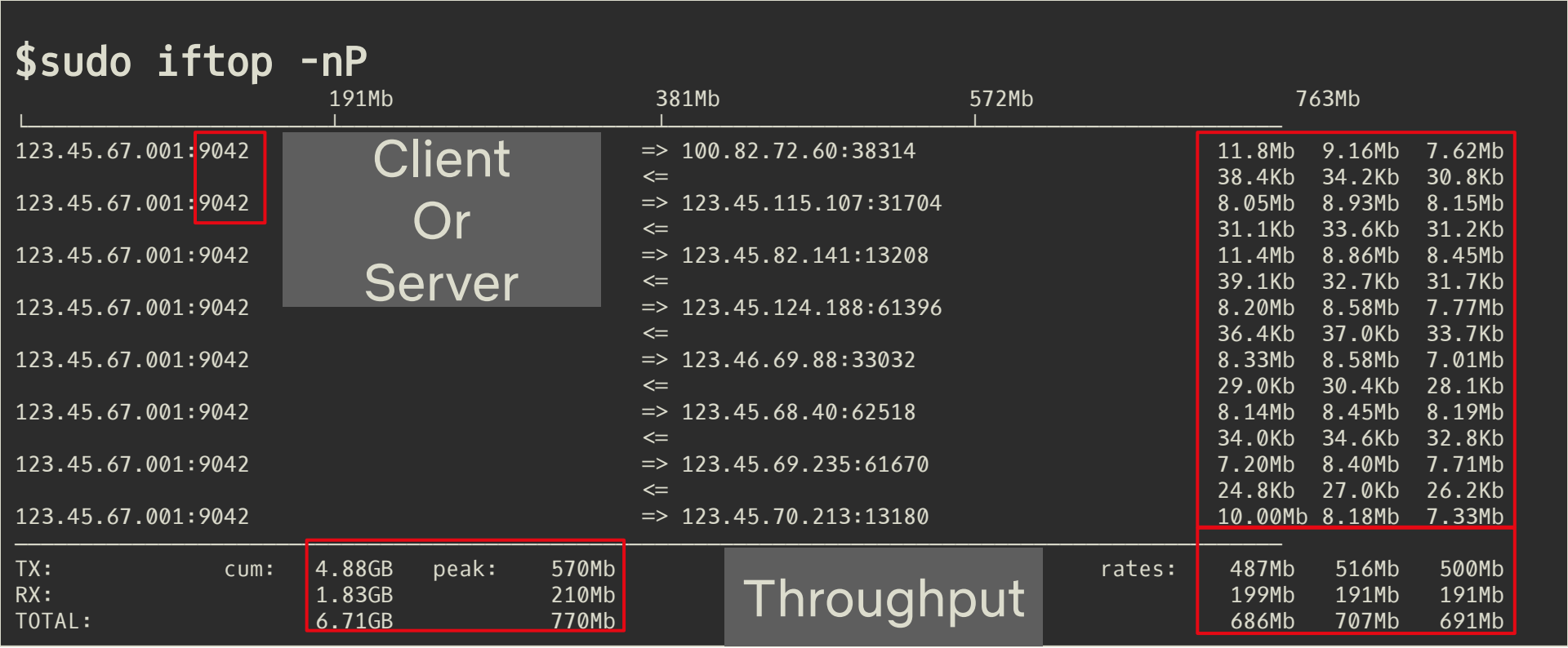
Are we out of disk space?

```
$ sudo iostat -xdm 2 /dev/nvme*
Linux 4.15.0-1045-aws (cass-xyz--useast1d-i-0db4886b5fd3fd01e)     09/05/2019     _x86_64_     (16 CPU)

Device:  rrqm/s  wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
nvme0n1    8.39    1.31  542.60    7.32     9.77     5.15    55.56     0.08    0.14    0.13    0.94   0.11   5.94
nvme1n1    8.40    1.18  542.63    7.09     9.77     5.15    55.56     0.08    0.14    0.13    0.96   0.11   5.97

Device:  rrqm/s  wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
nvme0n1    0.00    6.50  388.50   79.00     7.66     0.63    36.30     0.03    0.07    0.09    0.00   0.07   3.20
nvme1n1    0.00    4.50  389.00   69.50     7.86     0.55    37.57     0.04    0.08    0.08    0.12   0.07   3.00

Device:  rrqm/s  wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
nvme0n1    0.00    0.00  330.50    0.00     6.76     0.00    41.90     0.04    0.13    0.13    0.00   0.11   3.60
nvme1n1    0.00    0.00  323.00    0.00     6.46     0.00    40.94     0.04    0.13    0.13    0.00   0.12   3.80
```

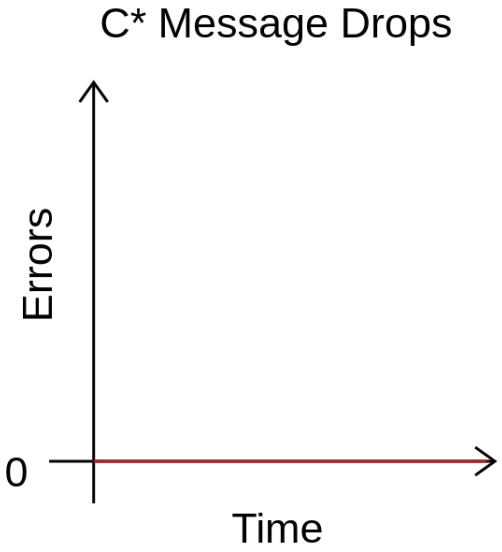
Throughput

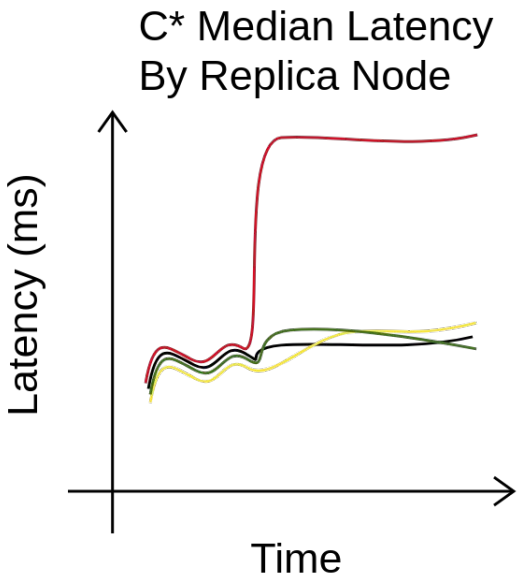
Latency

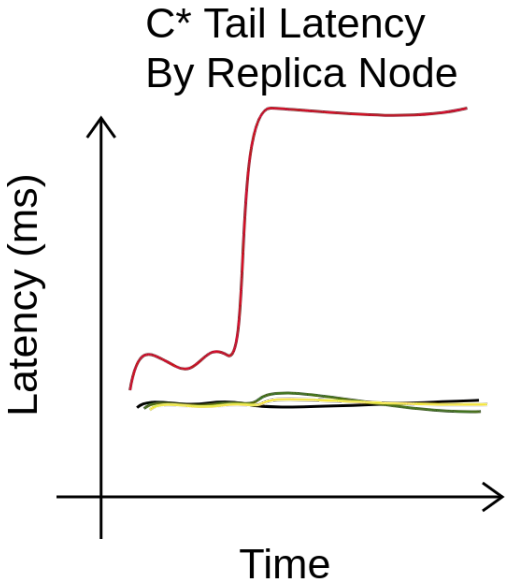
What are our disks doing?

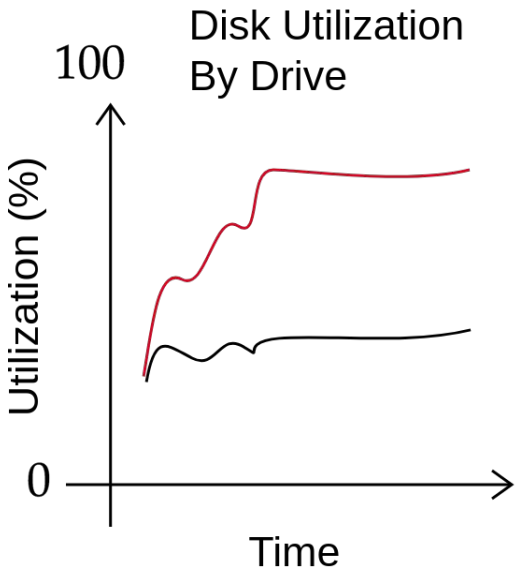


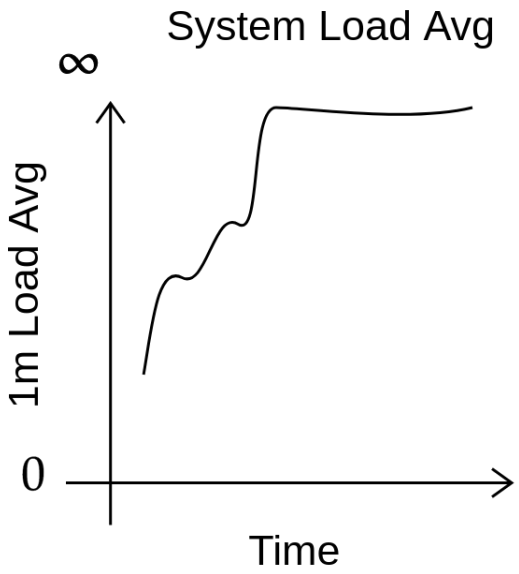
What is the network doing?

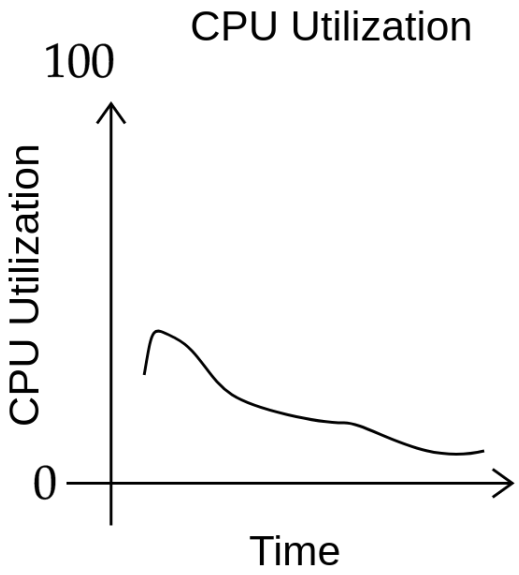

C* Message Drops
Errors
0
Time


C* Median Latency
By Replica Node
Latency (ms)
Time


C* Tail Latency
By Replica Node
Latency (ms)
Time


Disk Utilization
By Drive
100
Utilization (%)
0
Time


System Load Avg
∞
1m Load Avg
0
Time


CPU Utilization
100
CPU Utilization
0
Time

Initial Theories

Failed drive

Slow drive

Action Plan

Get lay of land

Investigate anomalies

```
$ sudo dmesg | grep -i XFS
[   19.004581] SGI XFS with ACLs, security attributes, realtime,
no debug enabled
[   19.007941] XFS (md0): Mounting V5 Filesystem
[   19.146420] XFS (md0): Starting recovery (logdev: internal)
[   19.229059] XFS (md0): Ending recovery (logdev: internal)

$ cat /proc/mdstat
Personalities : [raid0] [linear] [multipath] [raid1] [raid6]
[raid5] [raid4] [raid10]
md0 : active raid0 nvme1n1[1] nvme0n1[0]
      3710675328 blocks super 1.1 64k chunks

unused devices: <none>
```

This looks fine

```
$ sudo iostat -xdm 2 /dev/nvme*

Linux 4.15.0-1045-aws (cass-xyz--useast1d-i-0db4886b5fd3fd01e)       09/05/2019

Device:     r/s    w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await   svctm  %util
nvme0n1  542.60   7.32     9.77     5.15    55.56     0.08    0.14    0.13    0.94    0.11   5.94
nvme1n1  542.63   7.09     9.77     5.15    55.56     0.08    0.14    0.13    0.96    0.11   5.97

Device:     r/s    w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await   svctm  %util
nvme0n1  388.50  79.00     7.66     0.63    36.30     0.03 8110.37 8110.37    0.00 8084.27  82.20
nvme1n1  389.00  69.50     7.86     0.55    37.57     0.04    0.08    0.08    0.12    0.07   3.00

Device:     r/s    w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await   svctm  %util
nvme0n1  330.50   0.00     6.76     0.00    41.90     0.04 8082.27 8082.27    0.00 8081.27  80.60
nvme1n1  323.00   0.00     6.46     0.00    40.94     0.04    0.13    0.13    0.00    0.12   3.80
```

8 second IO latency … yeah not ok

Initial Theories	Reason
~~Failed drive~~	Healthy raid array, `dmesg`
Slow drive	Highly likely, `iostat` basically confirmed it

Proposed Experiment

Measure disk IO latency with `biosnoop`

Remove node from coordination and replication, **detach** from cluster

Reasoning

Before we remove load, measure the drives

Want to mitigate impact to user if possible. Don’t want to get same node back.

```
$ sudo /usr/share/bcc/tools/biosnoop | grep nvme0n1 > ios
$ cat ios | tr -s ' ' | cut -f 8 -d ' ' | histogram.py
# NumSamples = 10000; Min = 0.08; Max = 695.35
# Mean = 8.908690; Variance = 3680.9398; SD = 60.6707; Median 0.120000
# each ∎ represents a count of 96
     0.0800 -      0.7596 [  7260]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
     0.7596 -      2.1189 [   101]: ∎
     2.1189 -      4.8375 [   120]: ∎
     4.8375 -     10.2746 [   454]: ∎∎∎∎
    10.2746 -     21.1488 [  1922]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
    21.1488 -     42.8972 [    58]:
    42.8972 -     86.3941 [     0]:
    86.3941 -    173.3878 [     0]:
   173.3878 -    347.3752 [     0]:
   347.3752 -    695.3500 [    85]:
```

This does *not* look fine

```
$ sudo /usr/share/bcc/tools/biosnoop | grep nvme0n1 > ios
$ cat ios | tr -s ' ' | cut -f 8 -d ' ' | histogram.py
# NumSamples = 877; Min = 0.03; Max = 0.57
# Mean = 0.121209; Variance = 0.000891; SD = 0.029856; Median 0.120000
# each ∎ represents a count of 10
    0.0300 -     0.0840 [     7]:
    0.0840 -     0.1380 [   764]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎
    0.1380 -     0.1920 [   102]: ∎∎∎∎∎∎∎∎∎∎
    0.1920 -     0.2460 [     1]:
    0.2460 -     0.3000 [     0]:
    0.3000 -     0.3540 [     0]:
    0.3540 -     0.4080 [     0]:
    0.4080 -     0.4620 [     0]:
    0.4620 -     0.5160 [     1]:
    0.5160 -     0.5700 [     2]:
```

This is what i3s should look like

What would a failed drive look like?

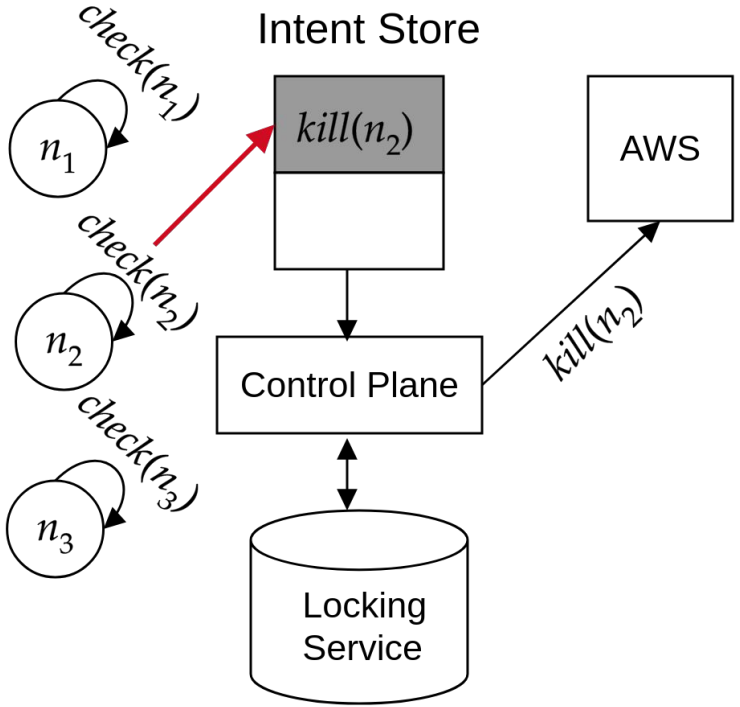
Failed drives would look like ...

```
$ ls /mnt
ls: cannot access '/mnt': Input/output error

$ sudo dmesg | grep -i XFS
[Jun 23 17:47:05 2018] print_req_error: I/O error, dev nvme0n1, sector 683734920
[Sat Jun 23 17:47:05 2018] print_req_error: I/O error, dev nvme0n1, sector 781933064
[Sat Jun 23 17:47:05 2018] XFS (md0): metadata I/O error: block 0x90
("xfs_buf_iodone_callback_error") error 5 numblks 16
[Sat Jun 23 17:47:05 2018] XFS (md0): xfs_do_force_shutdown(0x2) called from line
1232 of file ...
[Sat Jun 23 17:47:05 2018] XFS (md0): Log I/O Error Detected.  Shutting down
filesystem
[Sat Jun 23 17:47:05 2018] XFS (md0): Please umount the filesystem and rectify the
problem(s)
```

They are easy to detect

Automate

Detect bad hardware



$$
\begin{aligned}
check(n_i) \; = \; (\\
& fs\_failed(n_i) \qquad\qquad \vee \\
& |ioerror(n_i)| \; > \; 5 \qquad \vee \\
& \neg write(/\,\mathsf{mnt},\; n_i)) \quad \vee \\
& latency_{disk}(n_i) \; > 100ms \\
)
\end{aligned}
$$

$$
check(n_i) \; \rightarrow \; kill(n_i)
$$

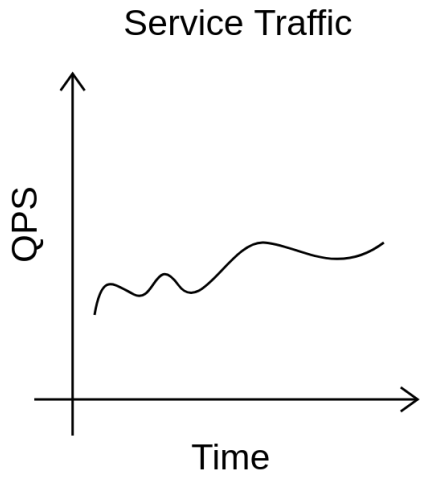
Only tricky one is disk latency

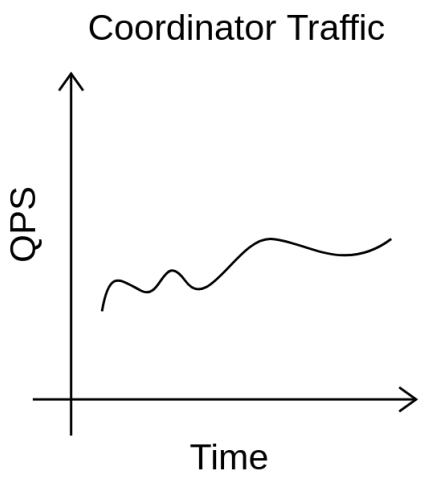
```
$ stat /mnt | grep Device
Device: 900h/2304d      Inode: 128         Links: 4
$ python3 -c 'print("{}:{}".format(2304 // 256, 2304 % 256))'
9:0

$ cat /sys/dev/block/9\:0/slaves/nvme0n1/stat
1663853604 25728830 61355902381 210415916 22426602  3994385 32168733071 20920828
0 182305980 230937712

$ cat /sys/dev/block/9\:0/slaves/nvme0n1/stat
#    reads  rmerges       rsect     rtime   writes  wmerges       wsect    wtime
1663853888 25728830 61355913957 210415948 22426603  3994385 32168733199 20920828
#      ios active_time   io_time",
         0   182306012 230937744
# See https://www.kernel.org/doc/Documentation/iostats.txt
```

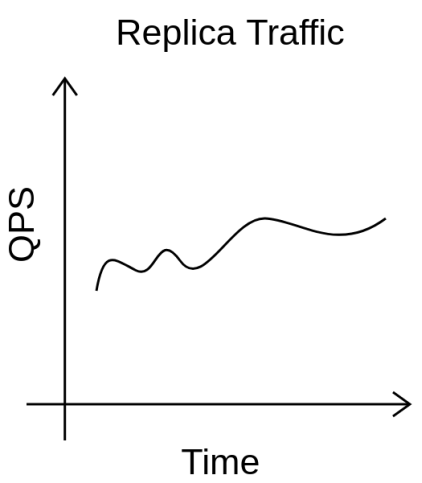
/proc/diskstats to the rescue!

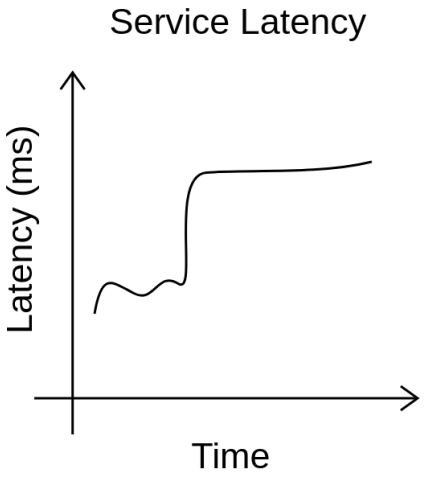
“We see high latency from Cassandra”

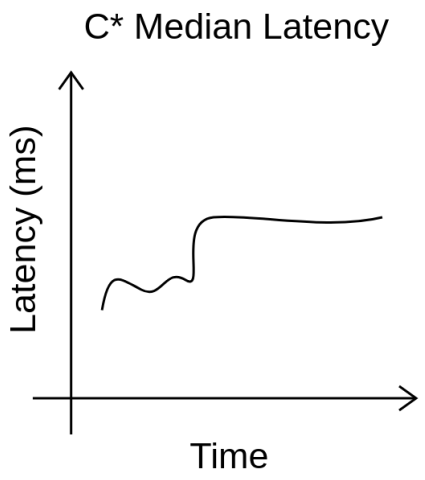
-Many a developer

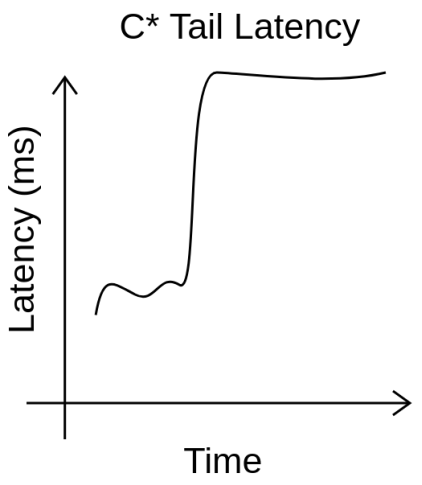

Service Traffic
QPS
Time

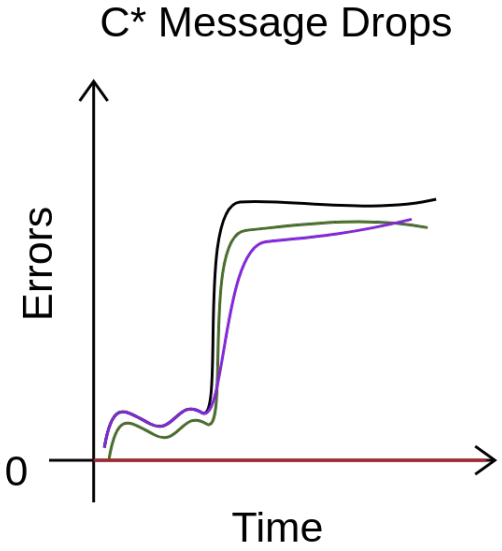

Coordinator Traffic
QPS
Time

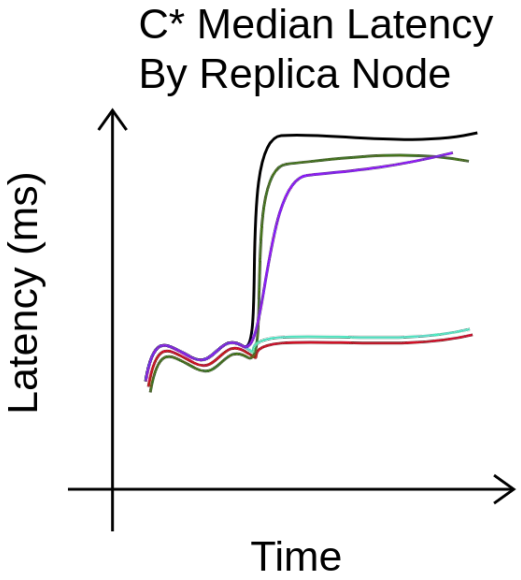

Replica Traffic
QPS
Time

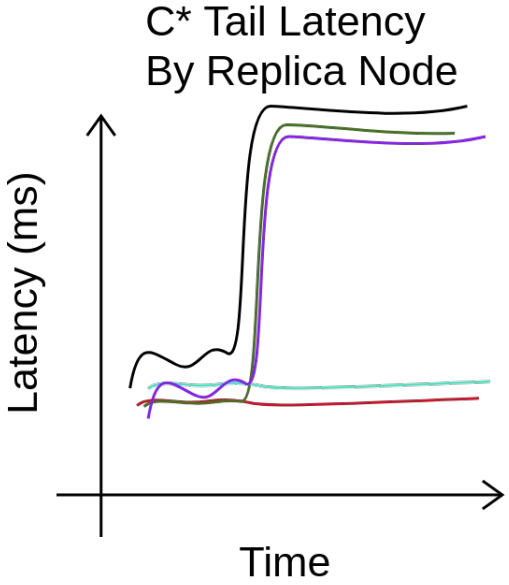

Service Latency
Latency (ms)
Time

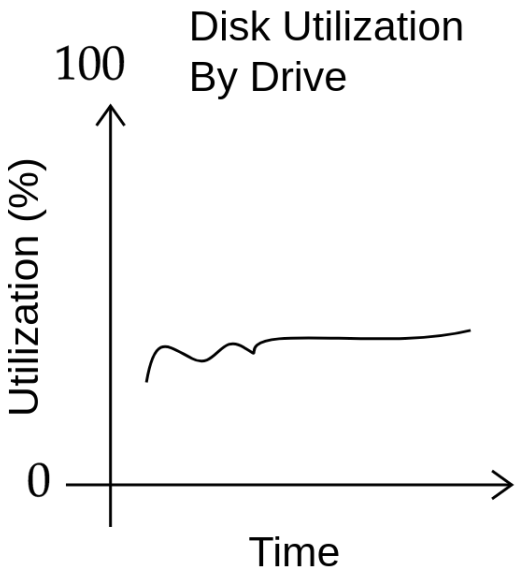

C* Median Latency
Latency (ms)
Time

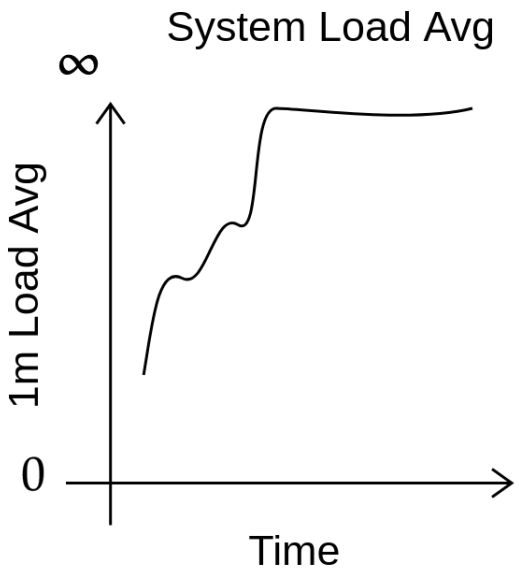

C* Tail Latency
Latency (ms)
Time

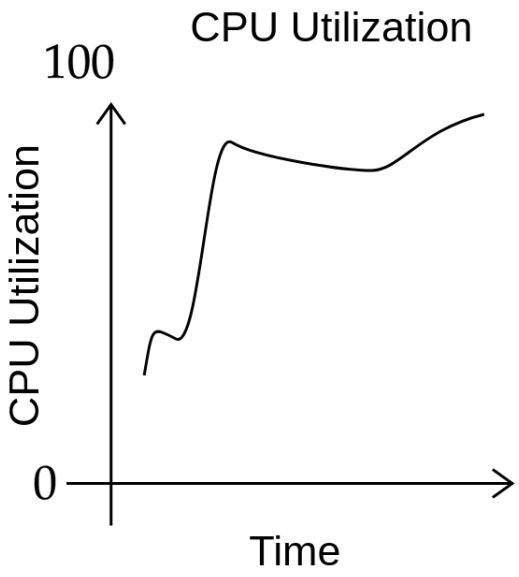

C* Message Drops
Errors
0
Time


C* Median Latency
By Replica Node
Latency (ms)
Time


C* Tail Latency
By Replica Node
Latency (ms)
Time


Disk Utilization
By Drive
100
Utilization (%)
0
Time


System Load Avg
∞
1m Load Avg
0
Time


CPU Utilization
100
CPU Utilization
0
Time

Initial Theories	Action Plan
Hot partition	Nodetool toppartitions
Repair	Check repair state
GC spiral of death	Check GC
Cassandra bug	Let’s hope it’s not this

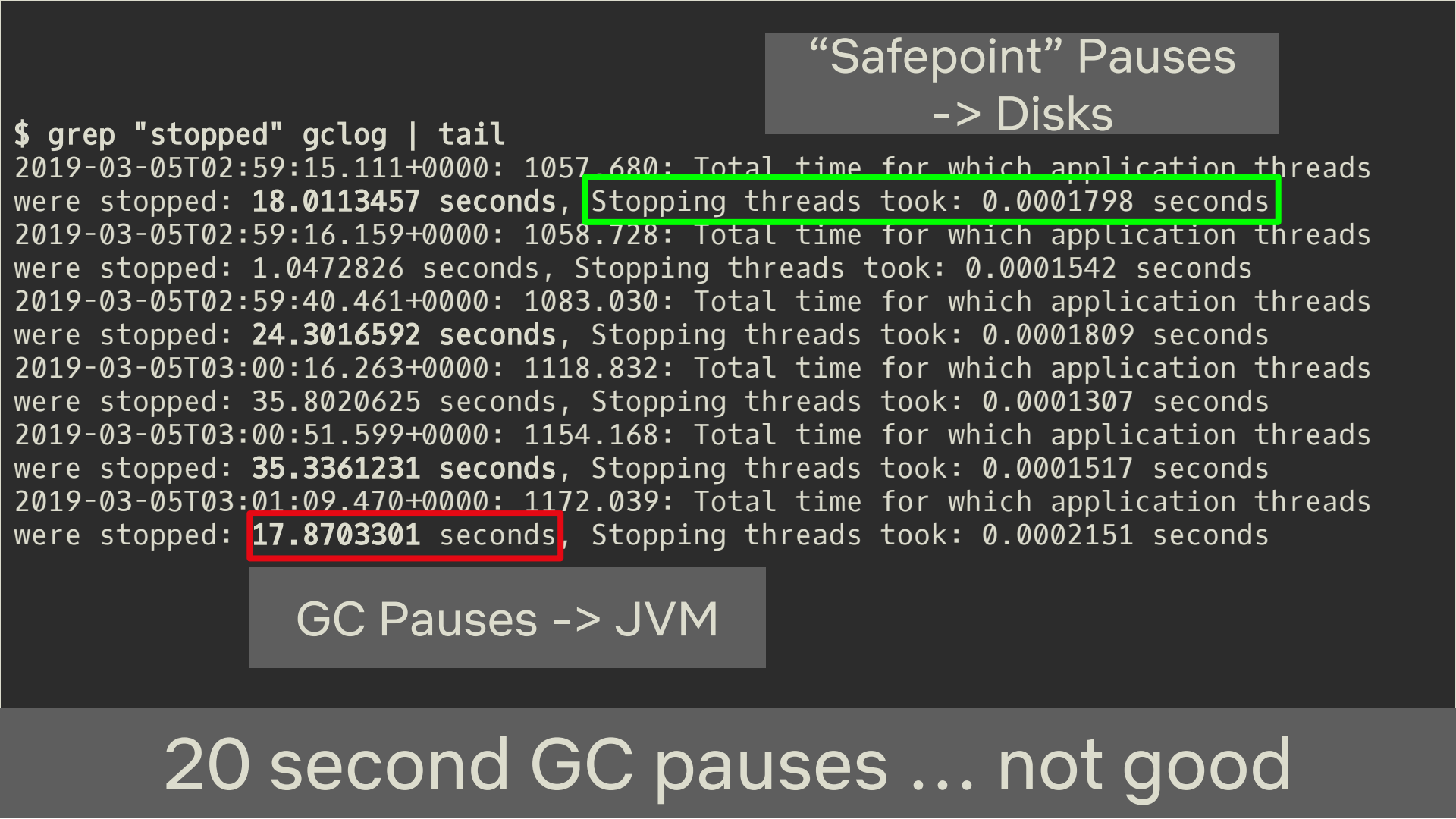
```
$ nodetool toppartitions -k 5 ks table 1000
READS Sampler:
  Cardinality: ~2 (256 capacity)
  Top 10 partitions:
        Partition               Count       +/-
        5045910909799600004         2         0
        1604866323413045 25         2         0
        4193640063441575 74         2         0
        1239440289879924 08         2         0
        2344241199854510 12         2         0

WRITES Sampler:
  Cardinality: ~2 (256 capacity)
  Top 10 partitions:
        Partition               Count       +/-
        5028886666964121 93         6         0
        0998829714520332 50         4         2
        1345398602110182 95         4         2
        2349486722118845 11         3         1
        2349462606712096 81         3         2
```

This looks reasonable

Seq #	Token	Node ID	Zone	Start Time	End Time	Duration	Status
	Filter by Token	Filter by Node ID	Filter by Zone				Filter by Status
123	-1537228670841756416	[redacted]	us-west-2c	Sep 4, 2019, 10:28:33 PM	Sep 4, 2019, 10:28:41 PM	8 seconds	FINISHED
124	-1345075088335240055	[redacted]	eu-west-1b	Sep 4, 2019, 10:30:15 PM	Sep 5, 2019, 4:53:15 AM	6 hours	FINISHED
125	-1345075086899412567	[redacted]	us-east-1c	Sep 5, 2019, 4:54:39 AM	Sep 5, 2019, 4:54:48 AM	9 seconds	FINISHED
126	-1345075086740615274	[redacted]	us-west-2b	Sep 5, 2019, 4:56:22 AM	Sep 5, 2019, 4:56:32 AM	10 seconds	FINISHED
127	-1152921504234098872	[redacted]	eu-west-1a	Sep 5, 2019, 4:57:34 AM		4 hours	STARTED (97%)
128	-1152921502798271384	[redacted]	us-east-1e				NOT_STARTED
129	-1152921502639474091	[redacted]	us-west-2a				NOT_STARTED

Not the same replica set...

Initial Theories	Action Plan
~~Hot partition~~	Equal query rates
~~Repair~~	Unrelated neighbors
GC spiral of death	Check GC log
Cassandra bug	Let’s hope it’s not this


“Safepoint” Pauses
-> Disks
$ grep "stopped" gclog | tail
2019-03-05T02:59:15.111+0000: 1057.680: Total time for which application threads
were stopped: 18.0113457 seconds, Stopping threads took: 0.0001798 seconds
2019-03-05T02:59:16.159+0000: 1058.728: Total time for which application threads
were stopped: 1.0472826 seconds, Stopping threads took: 0.0001542 seconds
2019-03-05T02:59:40.461+0000: 1083.030: Total time for which application threads
were stopped: 24.3016592 seconds, Stopping threads took: 0.0001809 seconds
2019-03-05T03:00:16.263+0000: 1118.832: Total time for which application threads
were stopped: 35.8020625 seconds, Stopping threads took: 0.0001307 seconds
2019-03-05T03:00:51.599+0000: 1154.168: Total time for which application threads
were stopped: 35.3361231 seconds, Stopping threads took: 0.0001517 seconds
2019-03-05T03:01:09.470+0000: 1172.039: Total time for which application threads
were stopped: 17.8703301 seconds, Stopping threads took: 0.0002151 seconds
GC Pauses -> JVM
20 second GC pauses … not good

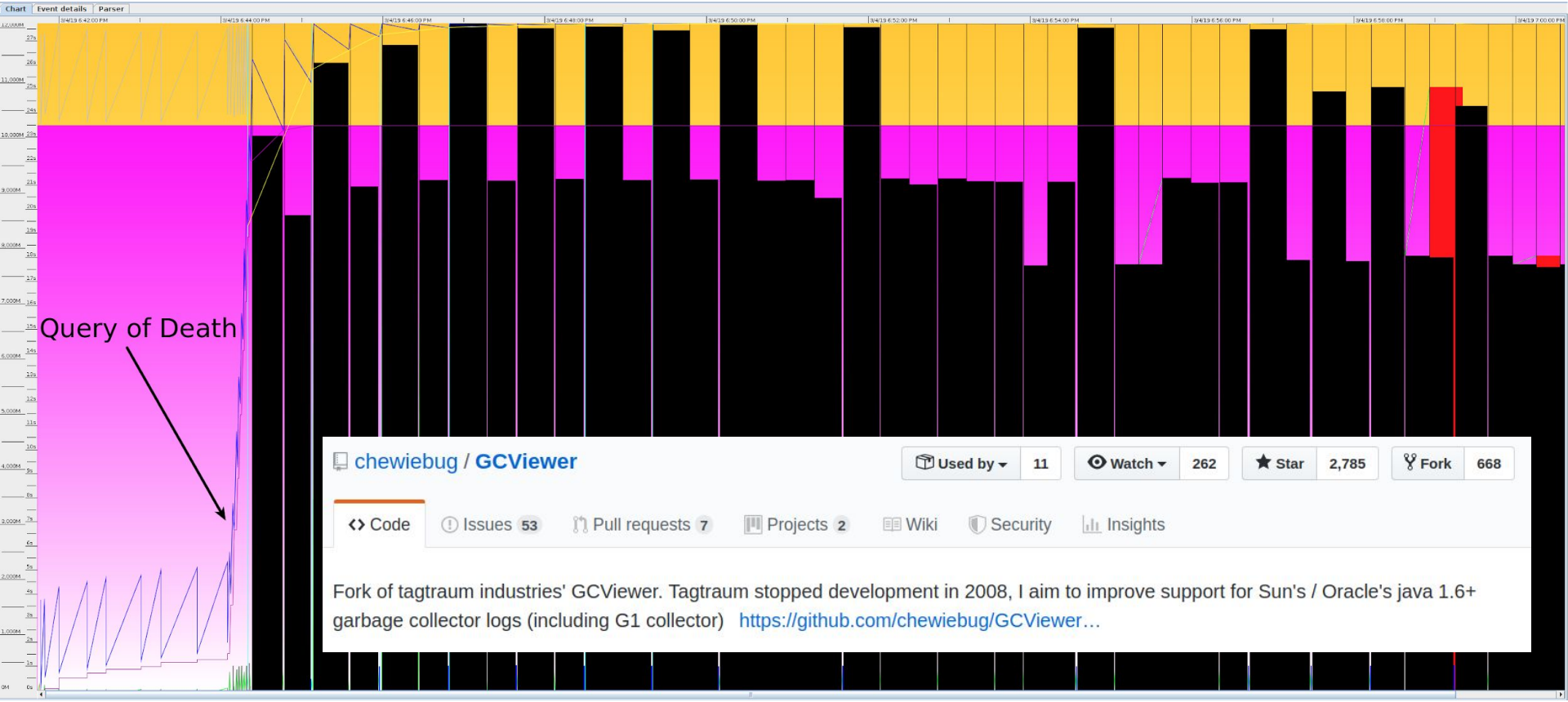


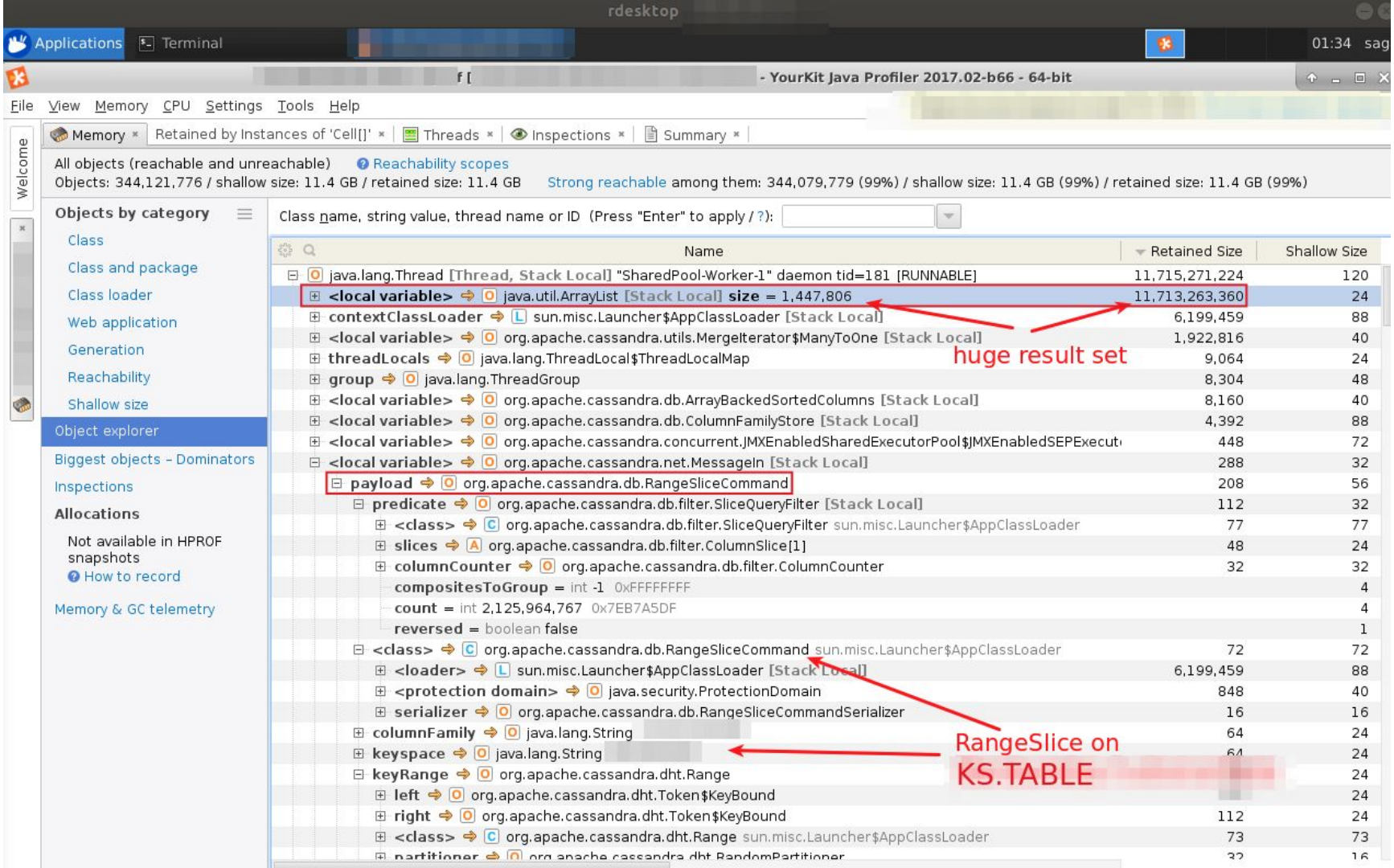
Chart
Event details
Parser
Query of Death
chewiebug / GCViewer
Used by 11
Watch 262
Star 2,785
Fork 668
Code
Issues 53
Pull requests 7
Projects 2
Wiki
Security
Insights
Fork of tagtraum industries' GCViewer. Tagtraum stopped development in 2008, I aim to improve support for Sun's / Oracle's java 1.6+ garbage collector logs (including G1 collector) https://github.com/chewiebug/GCViewer…

This is bad

Proposed Experiment	Reasoning
Observe JVM state with `jstat`	Confirm the GC theory
Take heap dump and restart node	Want to be able to debug while mitigating

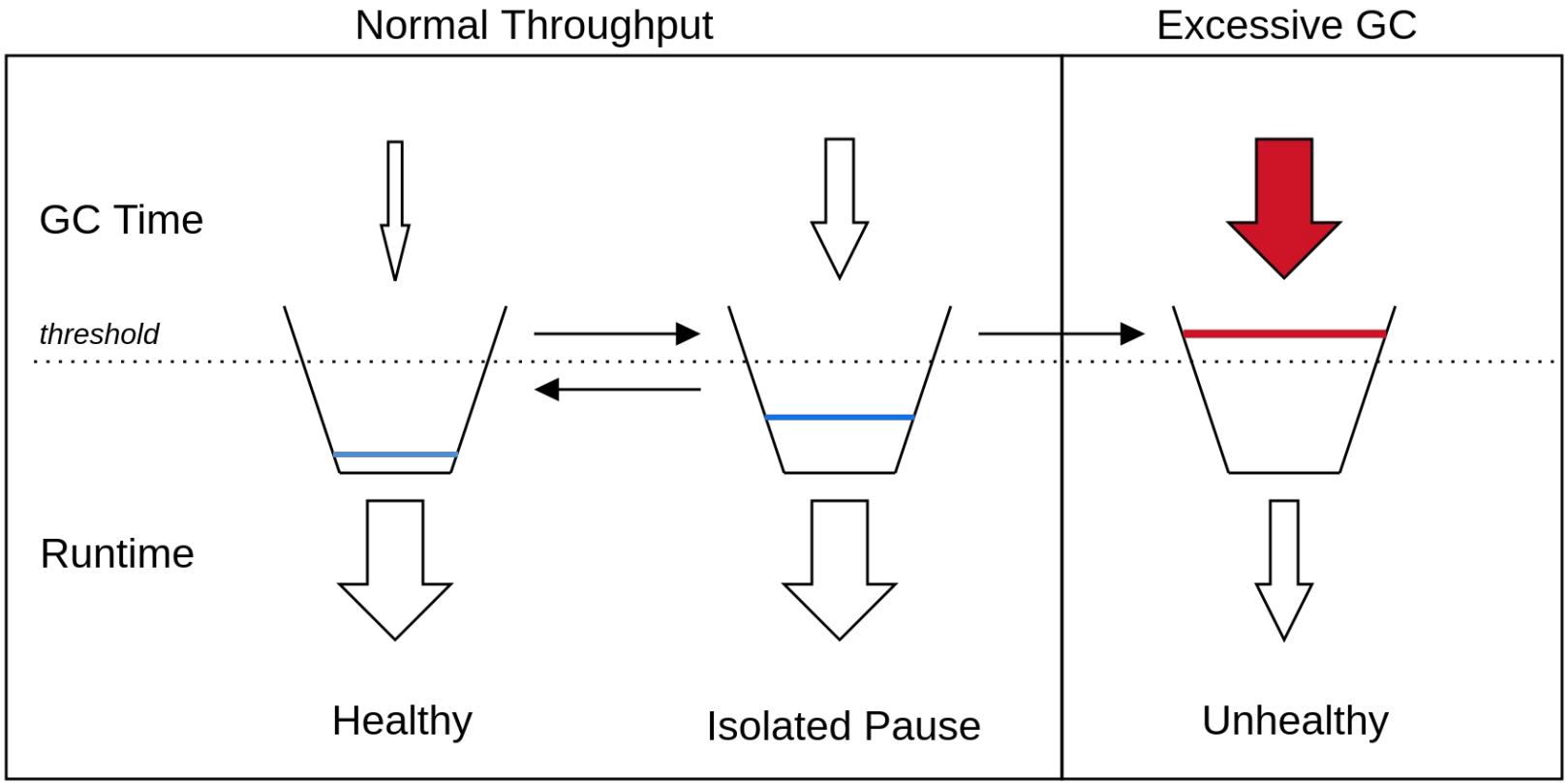
```
$ jstat -gcutil $(pgrep -f Cassandra) 1s
 S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
100.00   0.00 100.00 100.00  97.79  94.95  14042  624.634    10   88.798  713.432
100.00   0.00 100.00 100.00  97.79  94.95  14042  624.634    10   88.798  713.432
100.00   0.00 100.00 100.00  97.79  94.95  14042  624.634    10   88.798  713.432

# Might have to run this in a while loop if you're really stuck ...
$ sudo -u cassandra jmap -dump:live,format=b,file=core.hprof $(pgrep -f Cassandra)
Dumping heap to /mnt/data/cassandra/dumps/core.hprof …

# Since it will be O(heap size) large, you probably need to take it off box
$ aws s3 cp core.hprof s3://bucket/to/stash/coredumps

# Mitigate, probably need to forcibly kill since C* has a sigterm handler
$ sudo kill -9 $(pgrep -f Cassandra)
```

100% heap utilization is Bad™


rdesktop
Applications
Terminal
01:34 sag
- YourKit Java Profiler 2017.02-b66 - 64-bit
File View Memory CPU Settings Tools Help
Welcome
Memory
Retained by Instances of 'Cell[]'
Threads
Inspections
Summary
All objects (reachable and unreachable)
Reachability scopes
Objects: 344,121,776 / shallow size: 11.4 GB / retained size: 11.4 GB
Strong reachable among them: 344,079,779 (99%) / shallow size: 11.4 GB (99%) / retained size: 11.4 GB (99%)
Objects by category
Class
Class and package
Class loader
Web application
Generation
Reachability
Shallow size
Object explorer
Biggest objects – Dominators
Inspections
Allocations
Not available in HPROF snapshots
How to record
Memory & GC telemetry
Class name, string value, thread name or ID (Press "Enter" to apply / ?):
Name
Retained Size
Shallow Size
java.lang.Thread [Thread, Stack Local] "SharedPool-Worker-1" daemon tid=181 [RUNNABLE] 11,715,271,224 120
<local variable> java.util.ArrayList [Stack Local] size = 1,447,806 11,713,263,360 24
contextClassLoader sun.misc.Launcher$AppClassLoader [Stack Local] 6,199,459 88
<local variable> org.apache.cassandra.utils.MergeIterator$ManyToOne [Stack Local] 1,922,816 40
threadLocals java.lang.ThreadLocal$ThreadLocalMap 9,064 24
group java.lang.ThreadGroup 8,304 48
<local variable> org.apache.cassandra.db.ArrayBackedSortedColumns [Stack Local] 8,160 40
<local variable> org.apache.cassandra.db.ColumnFamilyStore [Stack Local] 4,392 88
<local variable> org.apache.cassandra.concurrent.JMXEnabledSharedExecutorPool$JMXEnabledSEPExecut 448 72
<local variable> org.apache.cassandra.net.MessageIn [Stack Local] 288 32
payload org.apache.cassandra.db.RangeSliceCommand 208 56
predicate org.apache.cassandra.db.filter.SliceQueryFilter [Stack Local] 112 32
<class> org.apache.cassandra.db.filter.SliceQueryFilter sun.misc.Launcher$AppClassLoader 77 77
slices org.apache.cassandra.db.filter.ColumnSlice[1] 48 24
columnCounter org.apache.cassandra.db.filter.ColumnCounter 32 32
compositesToGroup = int -1 0xFFFFFFFF 4
count = int 2,125,964,767 0x7EB7A5DF 4
reversed = boolean false 1
<class> org.apache.cassandra.db.RangeSliceCommand sun.misc.Launcher$AppClassLoader 72 72
<loader> sun.misc.Launcher$AppClassLoader [Stack Local] 6,199,459 88
<protection domain> java.security.ProtectionDomain 848 40
serializer org.apache.cassandra.db.RangeSliceCommandSerializer 16 16
columnFamily java.lang.String 64 24
keyspace java.lang.String 64 24
keyRange org.apache.cassandra.dht.Range 24
left org.apache.cassandra.dht.Token$KeyBound 24
right org.apache.cassandra.dht.Token$KeyBound 112 24
<class> org.apache.cassandra.dht.Range sun.misc.Launcher$AppClassLoader 73 73
huge result set
RangeSlice on
KS.TABLE

Automate

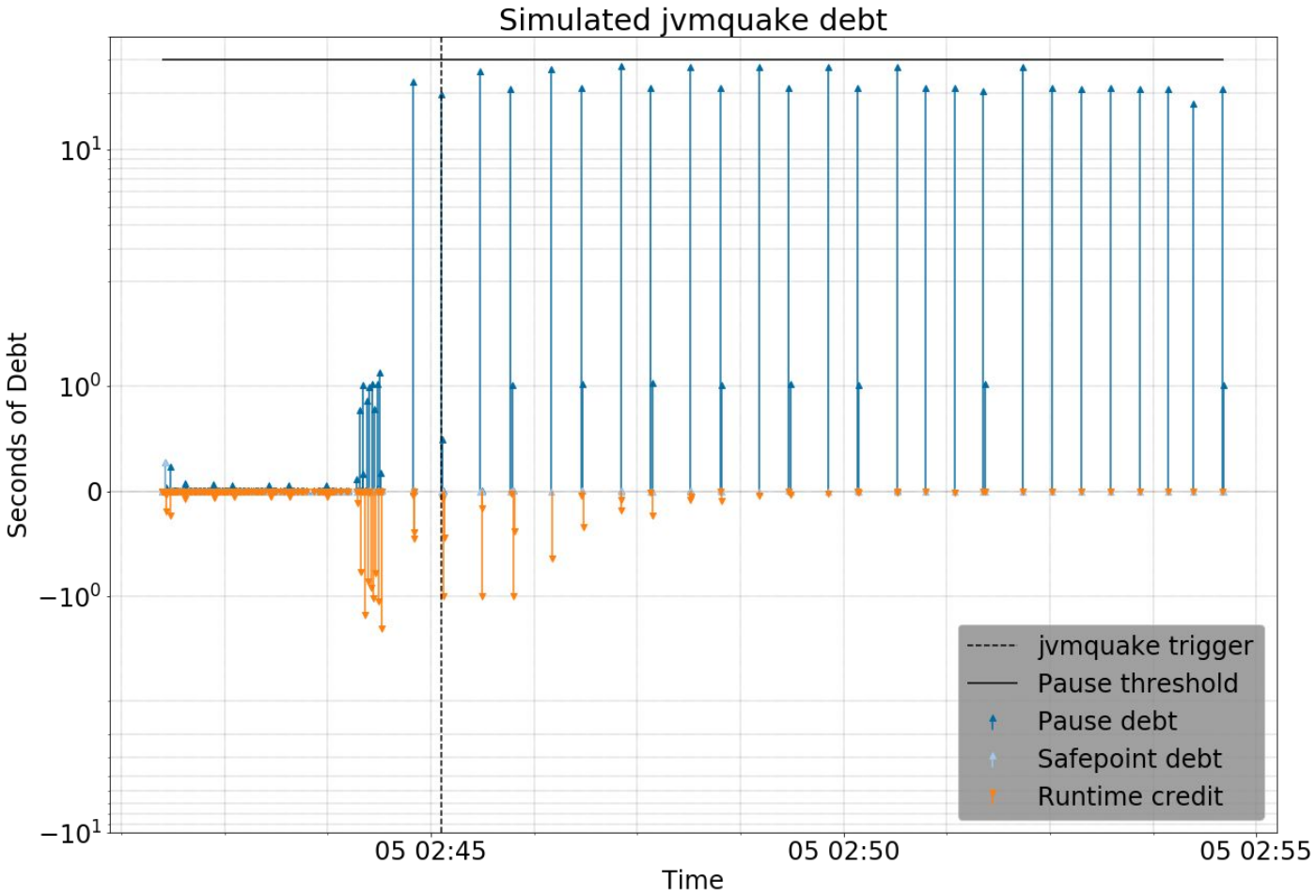
GC Spiral Of Death



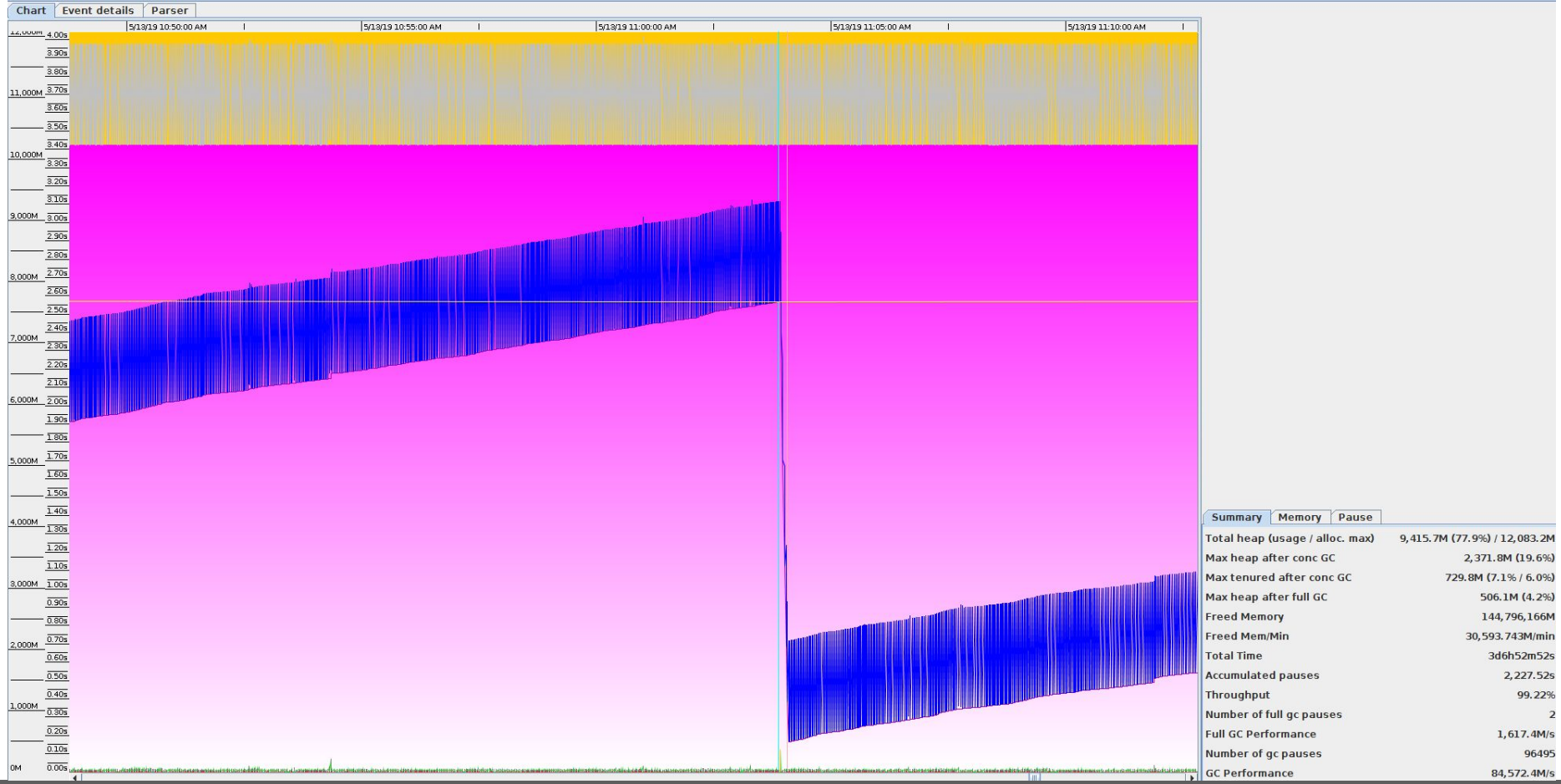
https://github.com/Netflix-Skunkworks/jvmquake

Automate

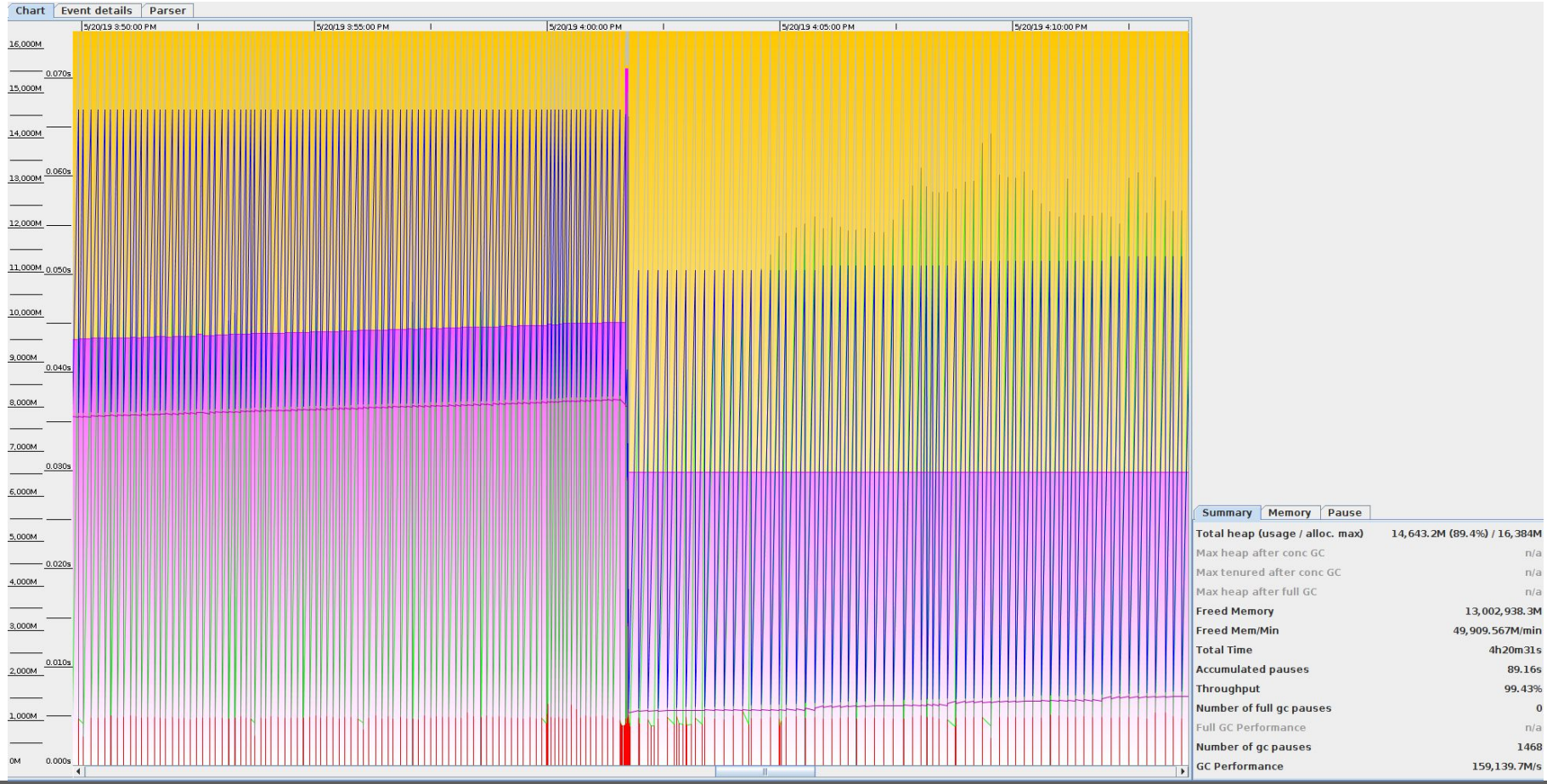
GC Spiral Of Death



https://github.com/Netflix-Skunkworks/jvmquake



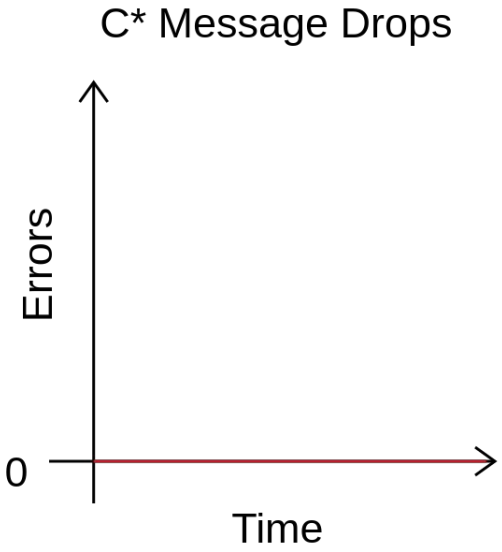
This is Normal CMS

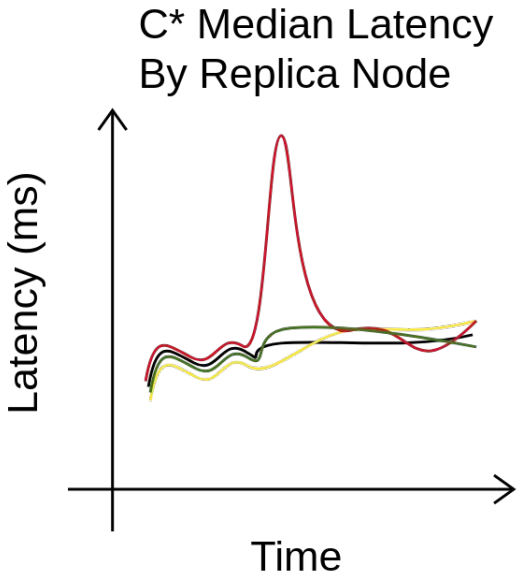


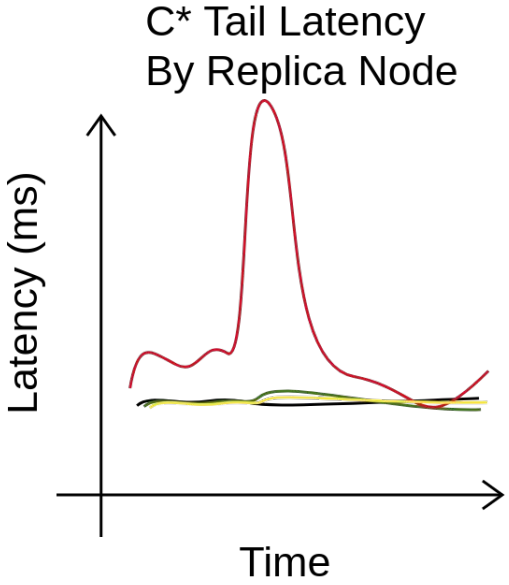
This is Normal G1GC

“Anything wrong with Cassandra during XY:ZZ minute? We saw errors?”

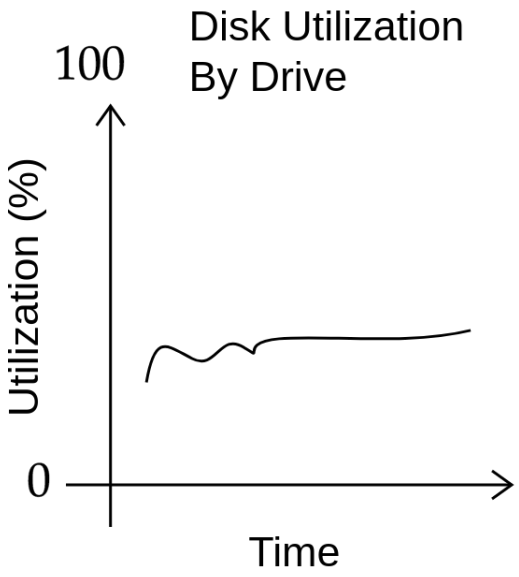
-Many a developer

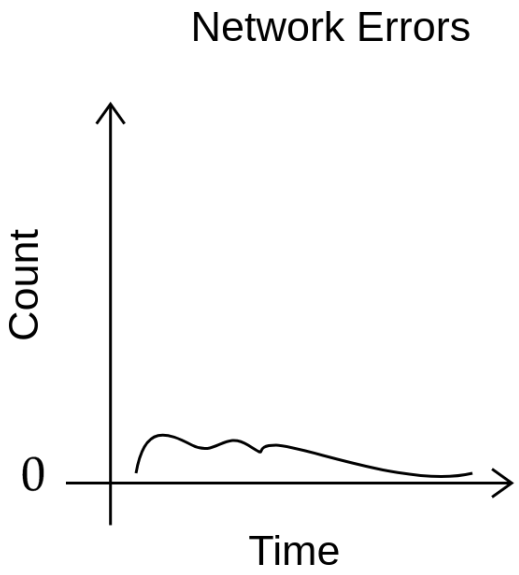

C* Message Drops
Errors
0
Time

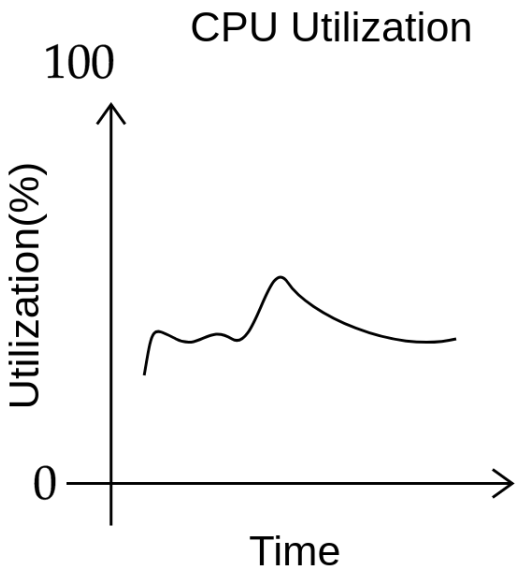

C* Median Latency
By Replica Node
Latency (ms)
Time


C* Tail Latency
By Replica Node
Latency (ms)
Time

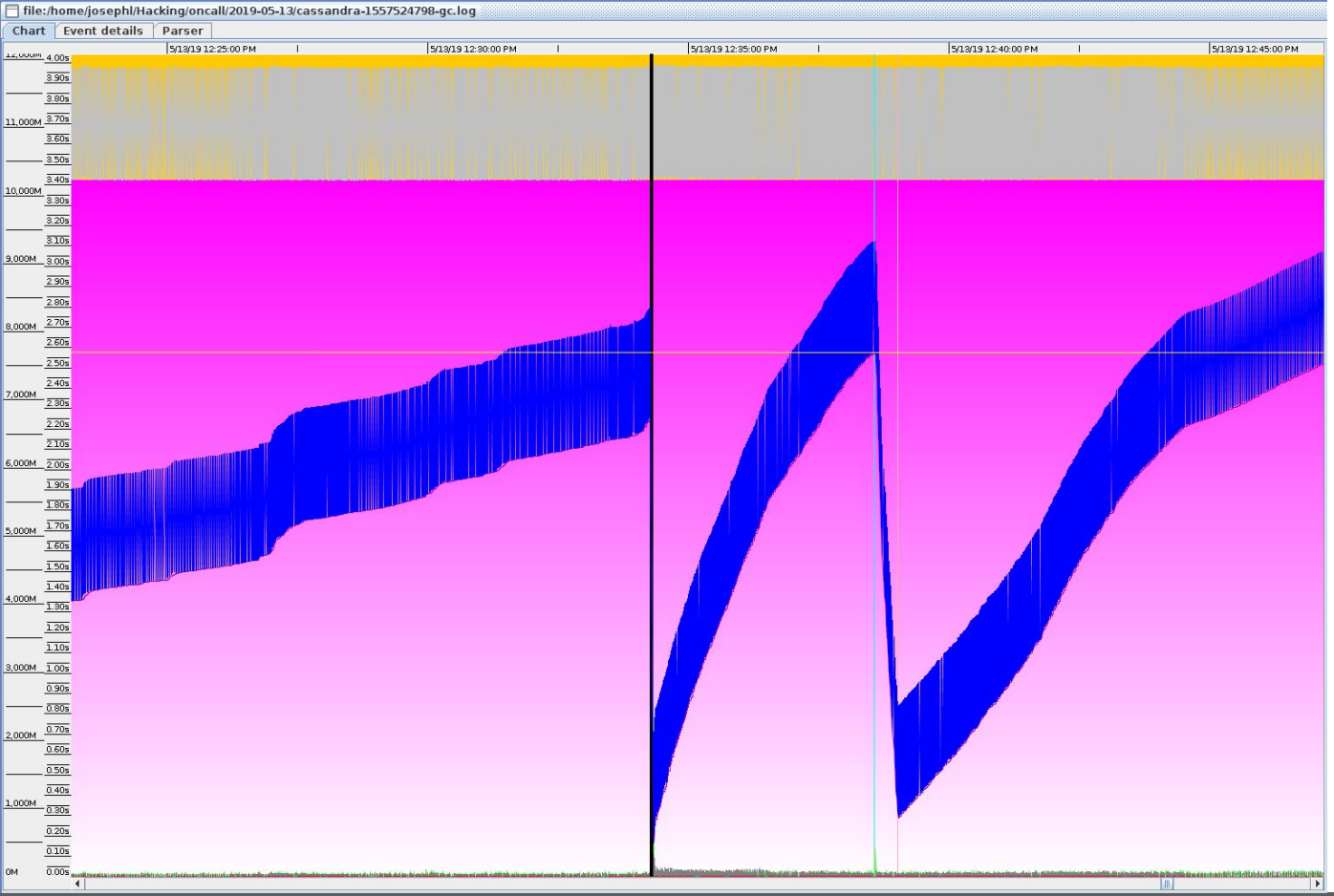
Initial Theories	Action Plan
Drive blip	Drive latency metrics
Network blip	Network metrics
Garbage Collection	GC Analysis
Cassandra bug	Let’s hope it’s not this


Disk Utilization
By Drive
100
Utilization (%)
0
Time


Network Errors
Count
0
Time


CPU Utilization
100
Utilization(%)
0
Time

Initial Theories	Reasoning
~~Drive blip~~	Drive latency metrics
~~Network blip~~	Network metrics
Garbage Collection	GC Analysis
Cassandra bug	Let’s hope it’s not this



Context

CMS is not a compacting collector

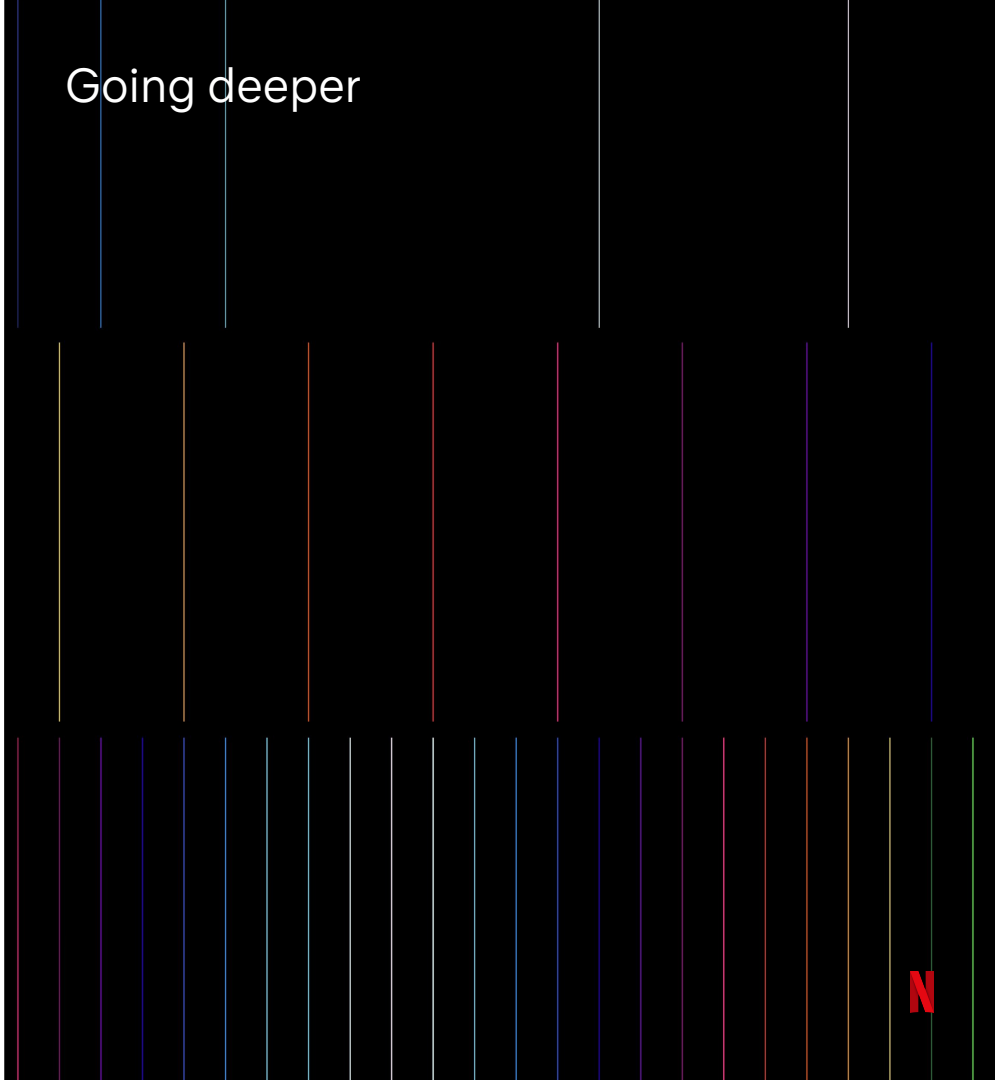
Say hi to a STW compacting GC

Proposed Experiment	Reasoning
Switch to G1GC	It works now, just use it.

What if it wasn’t that easy?

Advanced Maneuvers

Going deeper



“We tried quorum and it exploded.”

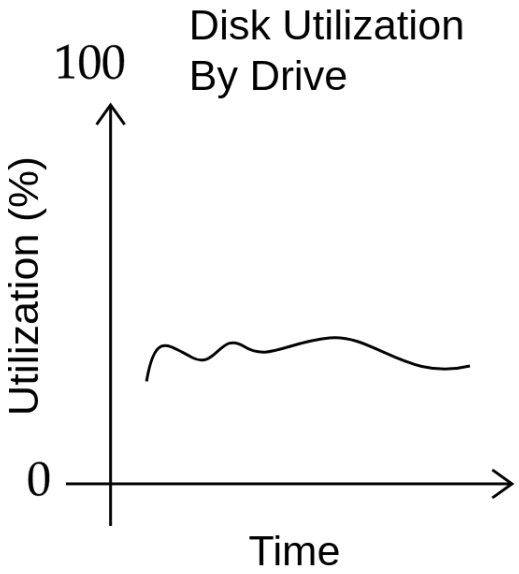
- Developer of new application

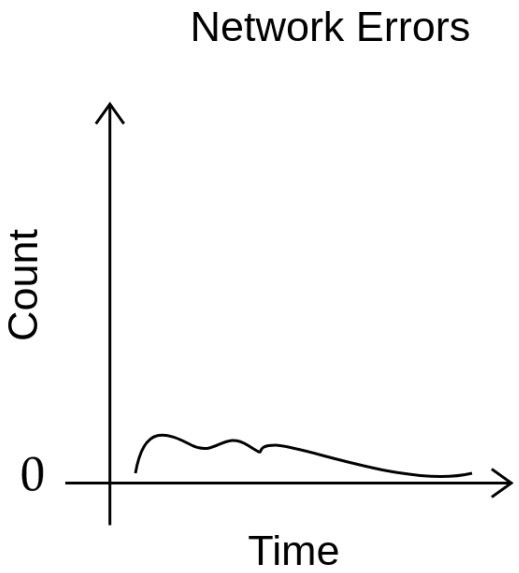
Initial Theories

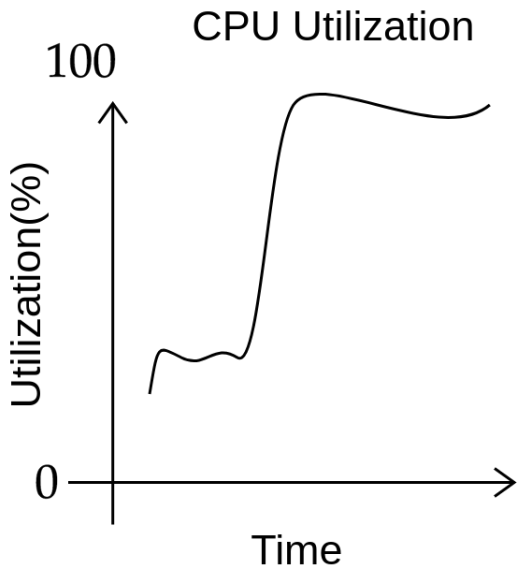
??

Action Plan

Find out what resource we ran out of.


Disk Utilization
By Drive
100
Utilization (%)
0
Time


Network Errors
Count
0
Time

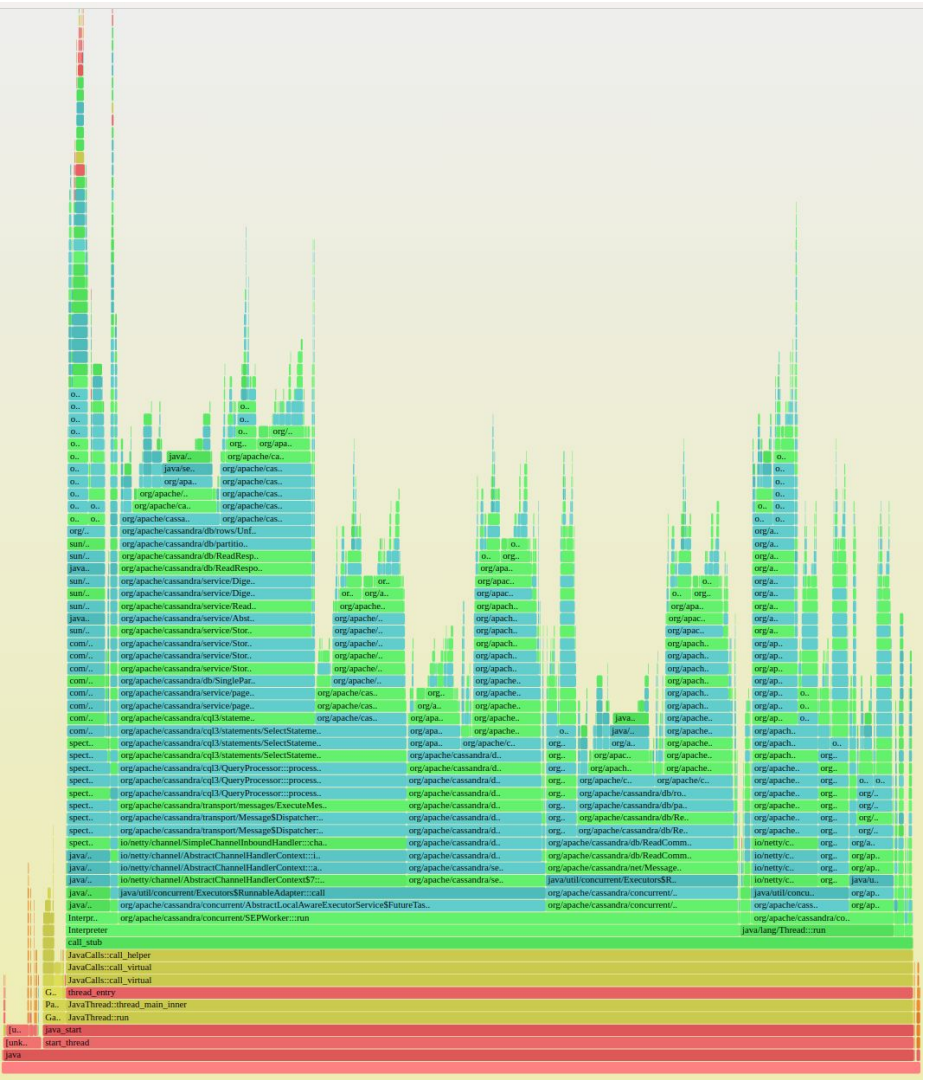

CPU Utilization
100
Utilization(%)
0
Time

Initial Theories

Cassandra bug?

Action Plan

Profile, then turn off quorum.

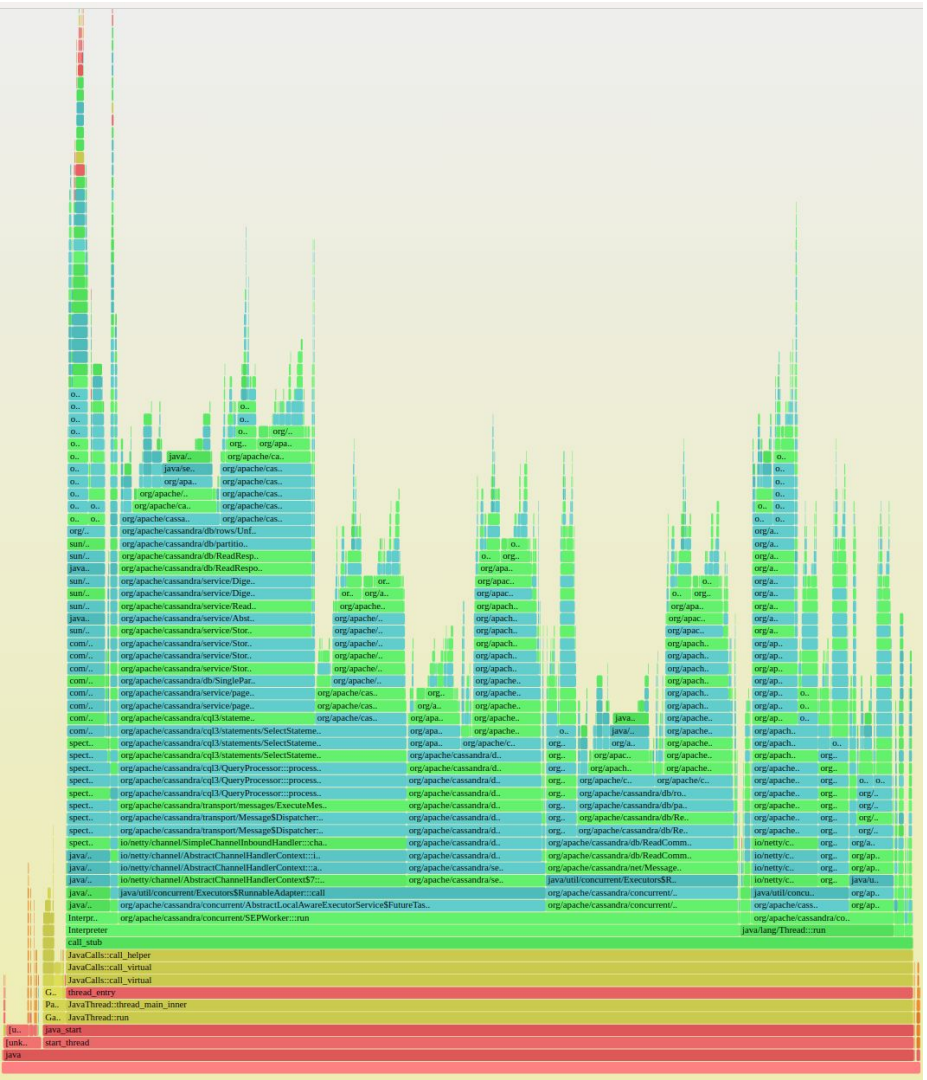


CPU Flamegraphs

Great way to visualize CPU usage across the OS,

Searchable by Java function name

https://medium.com/netflix-techblog/java-in-flames-e763b3d32166

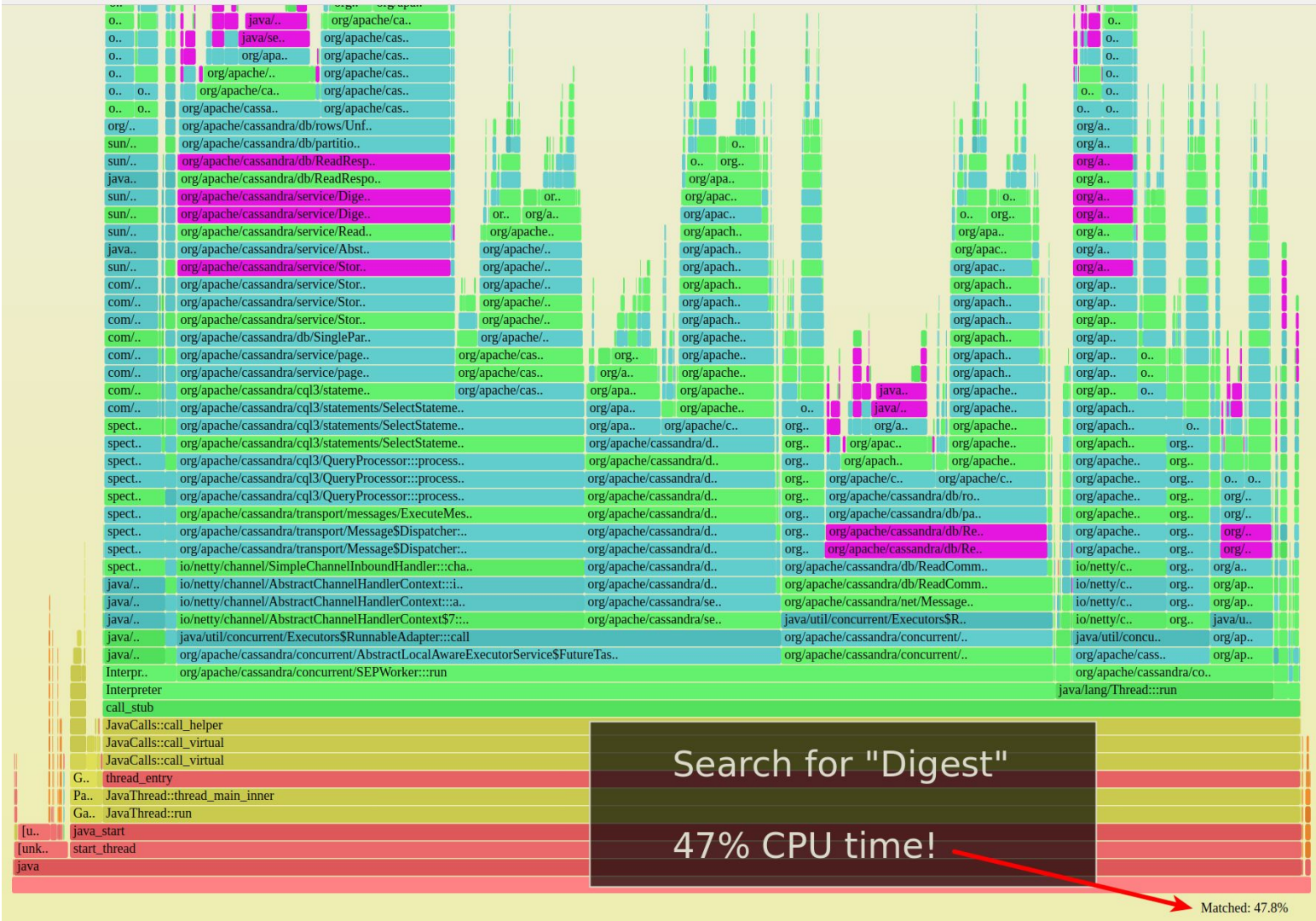


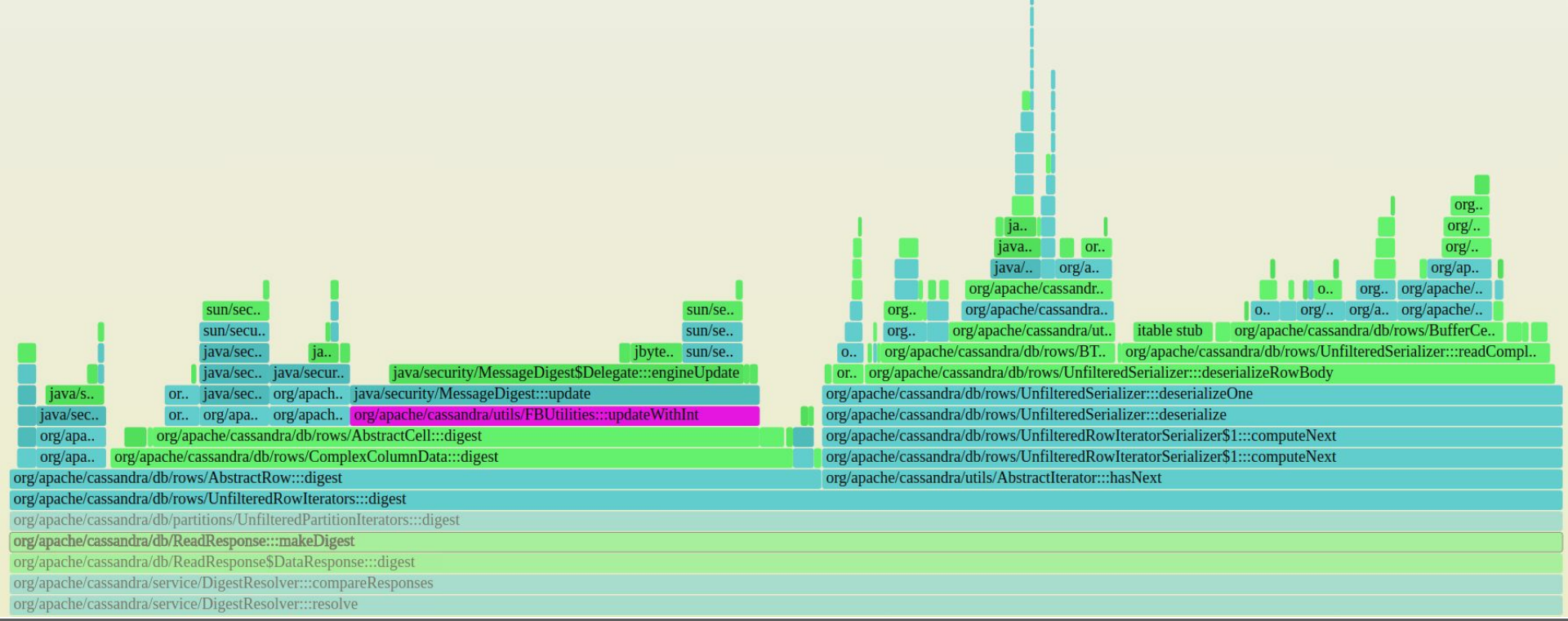
Dive deep by searching

Read quorums do cause a digest, but that should be cheap right?

What is the cost of that hash?

Search for “Digest”

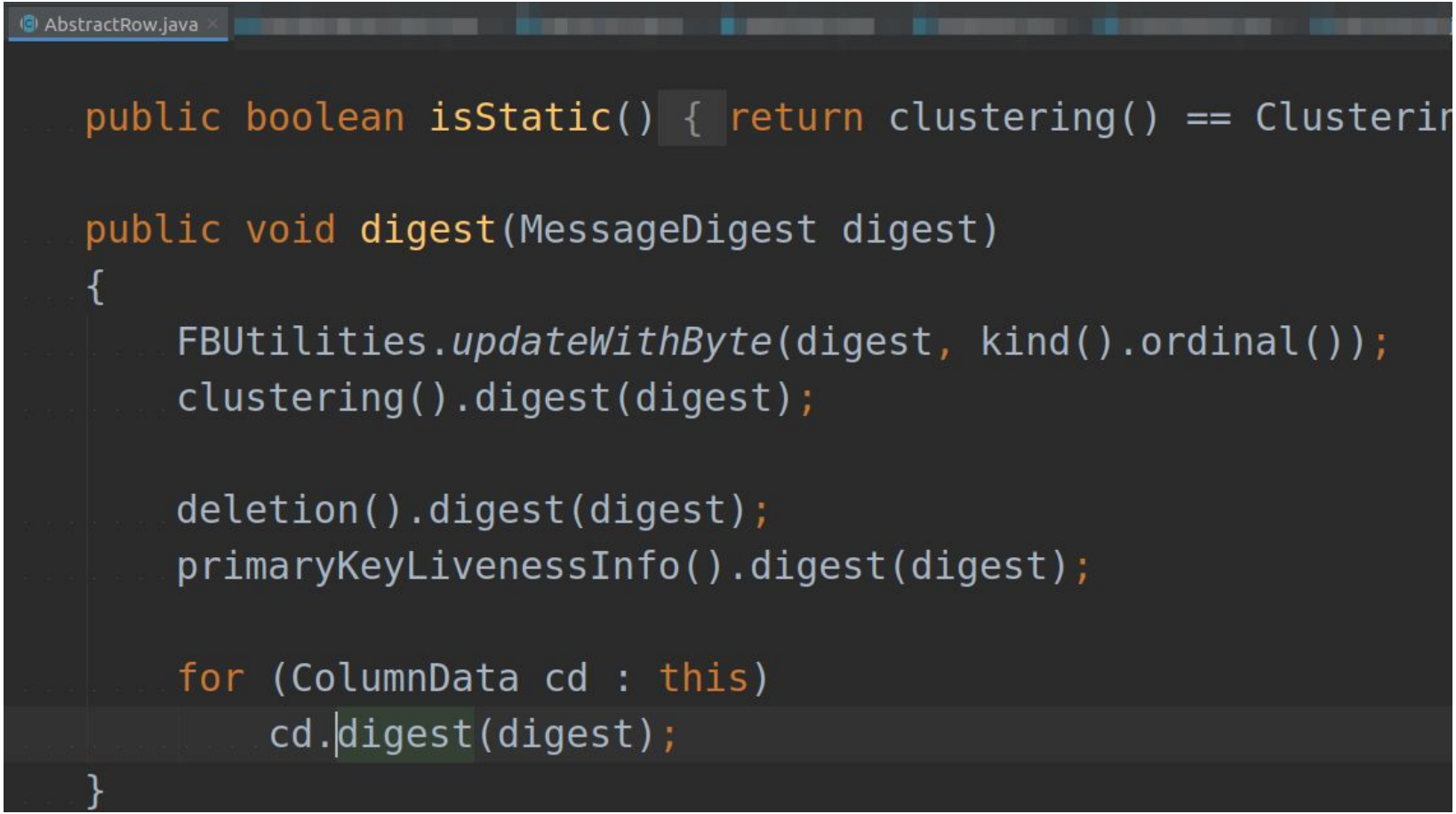

Search for "Digest"
47% CPU time!
Matched: 47.8%
java
start_thread
java_start
JavaThread::run
JavaThread::thread_main_inner
thread_entry
JavaCalls::call_virtual
JavaCalls::call_virtual
JavaCalls::call_helper
call_stub
Interpreter
java/lang/Thread:::run



A lot of `updateWithInt`

```java
public static void updateWithInt(MessageDigest digest, int val)
{
    digest.update((byte) ((val >>> 24) & 0xFF));
    digest.update((byte) ((val >>> 16) & 0xFF));
    digest.update((byte) ((val >>>  8) & 0xFF));
    digest.update((byte) ((val >>> 0) & 0xFF));
}
```

```
public void digest(MessageDigest digest)
{
    if (isCounterCell())
    {
        CounterContext.instance().updateDigest(digest, value());
    }
    else
    {
        digest.update(value().duplicate());
    }

    FBUtilities.updateWithLong(digest, timestamp());
    FBUtilities.updateWithInt(digest, ttl());
    FBUtilities.updateWithBoolean(digest, isCounterCell());
    if (path() != null)
        path().digest(digest);
}
```


AbstractRow.java
public boolean isStatic() { return clustering() == Clusterin
public void digest(MessageDigest digest)
{
FBUtilities.updateWithByte(digest, kind().ordinal());
clustering().digest(digest);
deletion().digest(digest);
primaryKeyLivenessInfo().digest(digest);
for (ColumnData cd : this)
cd.digest(digest);
}

What do we know?

Quorum “cost” 50% here, but why?

Action Plan

Which hash is C* using?

What does the data model look like?

ReadResponse.java

```java
protected static ByteBuffer makeDigest(UnfilteredPartitionIterator iterator, ReadCommand command)
{
    MessageDigest digest = FBUtilities.threadLocalMD5Digest();
    UnfilteredPartitionIterators.digest(command, iterator, digest, command.digestVersion());
    return ByteBuffer.wrap(digest.digest());
}
```

```
$ du -h file
1.0G file

$ time md5sum file
7d4f8791eb687cd923e0f90ee987d757  file
real    0m2.431s
user    0m2.266s
sys     0m0.148s

$ time xxhsum file
9e6ecc169707df6c  file

real    0m0.299s
user    0m0.105s
sys     0m0.194s
```

Yes, MD5 is a slow hash function.

Especially one byte at a time

```
CREATE TABLE ks.table (
    id bigint,
    key text,
    event_ts set<bigint>,
    PRIMARY KEY (id, key)
)
```

Uh oh, this could be bad

Proposed Experiment

Observe partition size distribution with `tablehistograms`

Reasoning

Wide partitions might be very expensive to digest

```
$ nt tablehistograms ks table
ks/table histograms
Percentile  SSTables     Write Latency      Read Latency    Partition Size    Cell Count
                              (micros)          (micros)           (bytes)
50%             3.00            24.60           1629.72              1916            42
75%             4.00            24.60           4866.32             20501           446
95%             4.00            35.43          17436.92            219342          5722
98%             4.00            35.43          25109.16            545791         14237
99%             4.00            42.51          30130.99            785939         20501
Min             0.00             6.87             24.60                73             0
Max             4.00            88.15         107964.79           2346799         73457
```

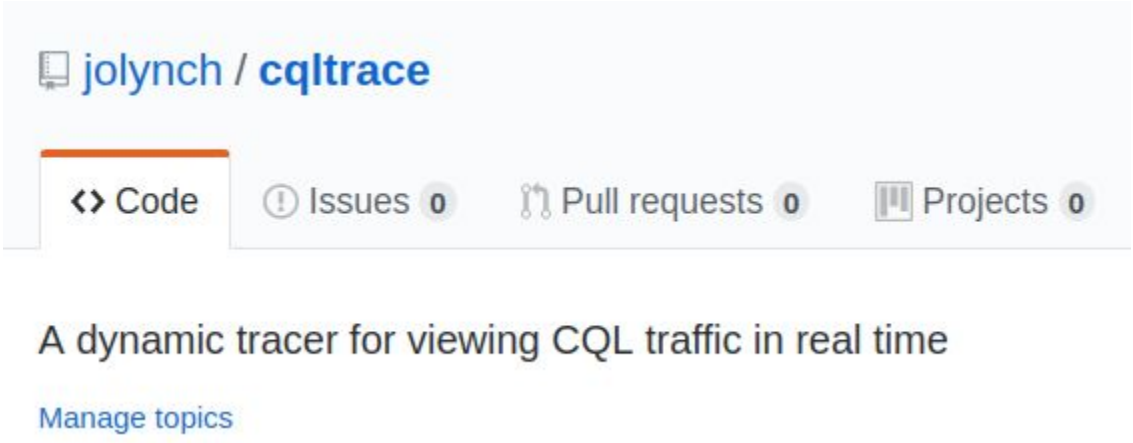
This doesn’t seem that bad

Proposed Experiment

Capture live traffic with `cqltrace`, see actual query distribution

Reasoning

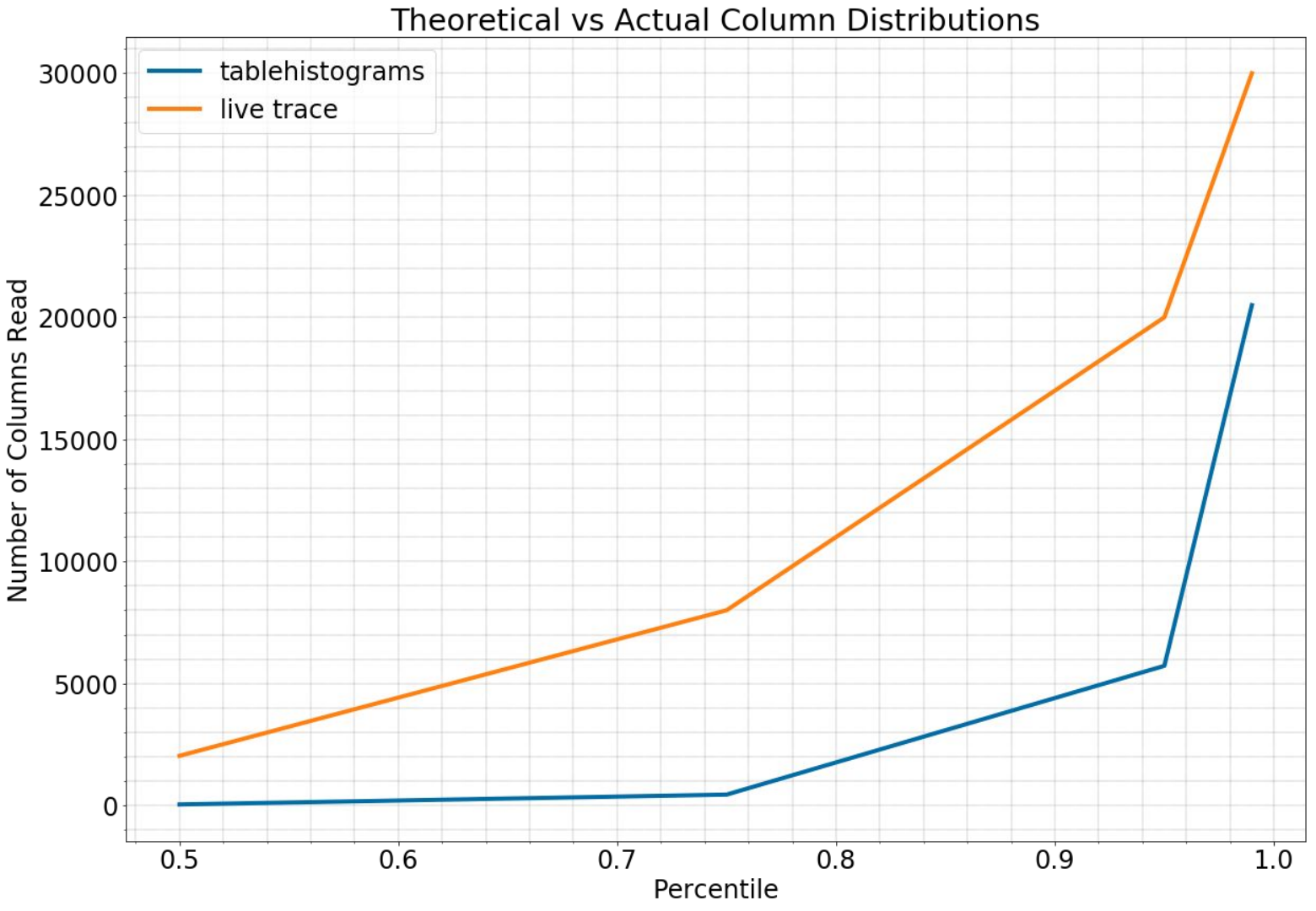
`tablehistograms` only shows on disk distribution... we need the actual distribution



```
# Basically just optimized tcpdump and tshark
$ sudo ./cqlcap -i eth0 -p 9042 data.pcap

$ ./cqltrace -H -p 9042 -b -k data.pcap > trace


$ head trace
Loading CQL Latency Tracer
SOURCE IP|STATEMENT ID|CONSISTENCY LEVEL|BOUND VARIABLES|LATENCY
100.12.34.235:15196|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS=030011D98345A13E|0.003676000
100.12.34.76:33114|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS =030011A0FEF22711|0.002695000
100.12.34.206:62418|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS=0300BF123D6F7FFE|0.002129000
100.12.34.248:33310|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS=0400B1D98604B269|0.000787000
100.12.34.163:33378|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS=0500B4A78B8115FB|0.002818000
100.12.34.60:46278|FB175FB7E427A725E3DE5534DCF3169A|LOCAL_ONE|BINDS =0000000012345659|0.001590000


$ cat trace | cut -f 4 -d '|' | tail -n 1
BINDS=030011D98345A13E
```

Nice, we have the keys

```
# Write a script which queries the keys being requested, observe #
$ cat columns_by_row | histogram.py -p -b 20
# NumSamples = 836; Min = 0.00; Max = 52673.00
# Mean = 5368.320574; Variance = 67827677.595797; SD = 8235.756043; Median 2035.000000
# each ∎ represents a count of 6
    0.0000 -  2633.6500 [   473]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (56.58%)
 2633.6500 -  5267.3000 [   135]: ∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎∎ (16.15%)
 5267.3000 -  7900.9500 [    59]: ∎∎∎∎∎∎∎∎∎ (7.06%)
 7900.9500 - 10534.6000 [    32]: ∎∎∎∎∎ (3.83%)
10534.6000 - 13168.2500 [    38]: ∎∎∎∎∎∎ (4.55%)
13168.2500 - 15801.9000 [    19]: ∎∎∎ (2.27%)
15801.9000 - 18435.5500 [    15]: ∎∎ (1.79%)
18435.5500 - 21069.2000 [    16]: ∎∎ (1.91%)
21069.2000 - 23702.8500 [     8]: ∎ (0.96%)
23702.8500 - 26336.5000 [     8]: ∎ (0.96%)
26336.5000 - 28970.1500 [     6]: ∎ (0.72%)
28970.1500 - 31603.8000 [     9]: ∎ (1.08%)
31603.8000 - 34237.4500 [     2]:  (0.24%)
34237.4500 - 36871.1000 [     3]:  (0.36%)
36871.1000 - 39504.7500 [     2]:  (0.24%)
39504.7500 - 42138.4000 [     1]:  (0.12%)
42138.4000 - 44772.0500 [     5]:  (0.60%)
44772.0500 - 47405.7000 [     4]:  (0.48%)
47405.7000 - 50039.3500 [     0]:  (0.00%)
50039.3500 - 52673.0000 [     1]:  (0.12%)
```


Theoretical vs Actual Column Distributions
tablehistograms
live trace
Number of Columns Read
30000
25000
20000
15000
10000
5000
0
0.5
0.6
0.7
0.8
0.9
1.0
Percentile

```
CREATE TABLE ks.table (
    id bigint,
    key text,
    event_ts text, // json blob
    PRIMARY KEY (id, key)
)
```

This has less hashing
(tradeoff consistency)

Automate

1. Education

— Explain context to developer, was able to change data model

2. Improve the product

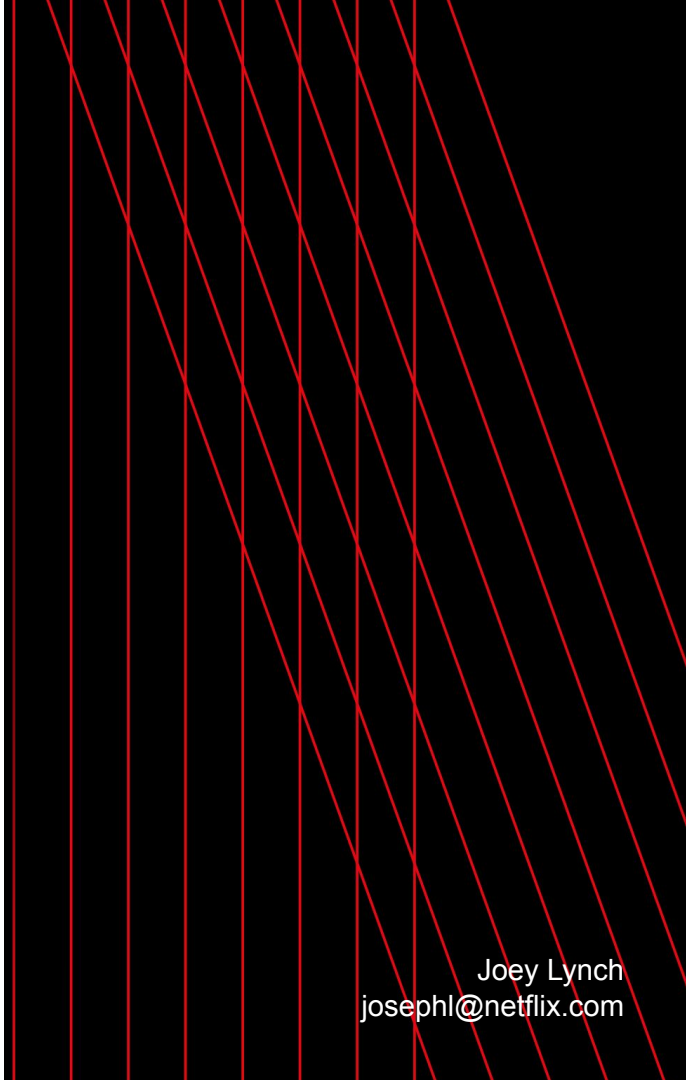
— File bug report upstream, work with community to fix the problem.

- CASSANDRA-13292: Better hash
- CASSANDRA-14611: Fix single byte update slowness
- CASSANDRA-15294: Use ACCP

Take Away

Focus on building mental models of the systems you hold a pager for. Debugging follows from science.

Thank You.



Joey Lynch
josephl@netflix.com