The Road to 20 Terabytes per Node: Overcoming Cassandra's Storage Density Challenges

Jon Haddad
RustyRazorBlade Consulting

Is Cassandra Storage Bound?

Past Present and Future

About Me

- > 10 years with Cassandra
- Committer / PMC
- Startups, DataStax, The Last Pickle, Apple, Netflix
- Rustyrazorblade Consulting
- Cassandra Consulting, Training
- Platform Eng, Observability

Improving Storage Density Can Decrease Costs Significantly

What Affects Density?

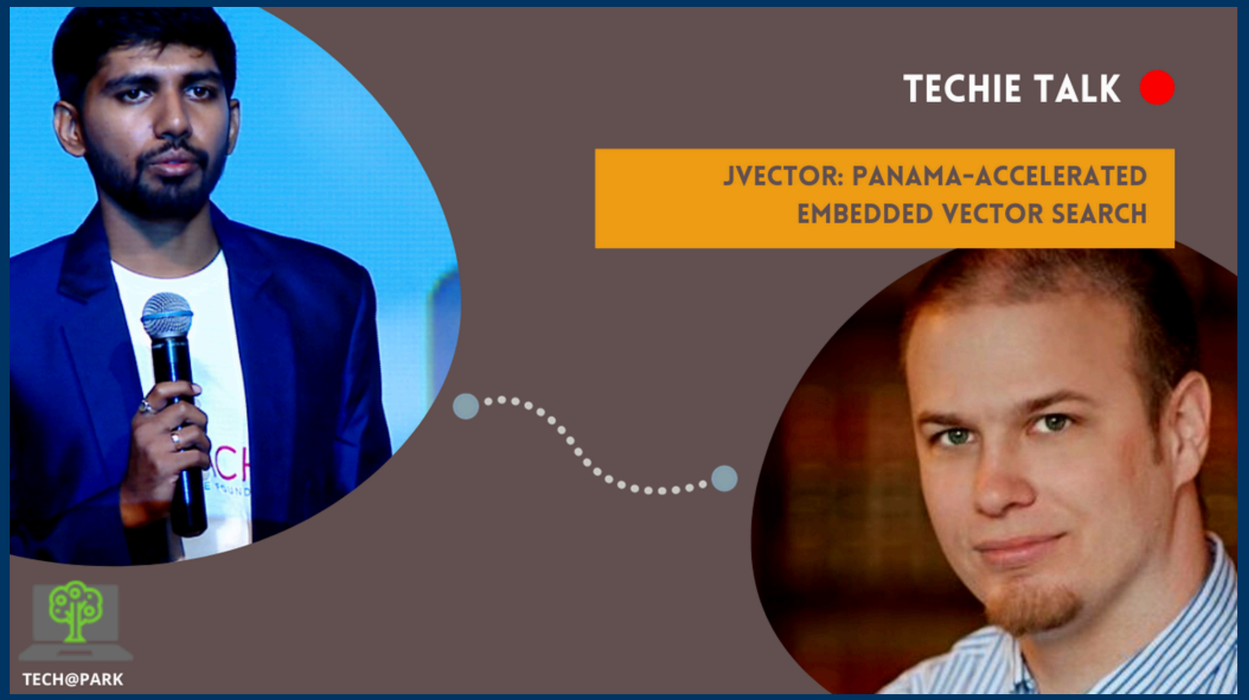
- Hardware capabilities
- Workload
- Compression
- Compaction
- Streaming
- Repair

Modern Hardware is Insane

- NVMe
- Hundreds of cores
- Hundreds of GB of RAM
- 10GB+ Networks

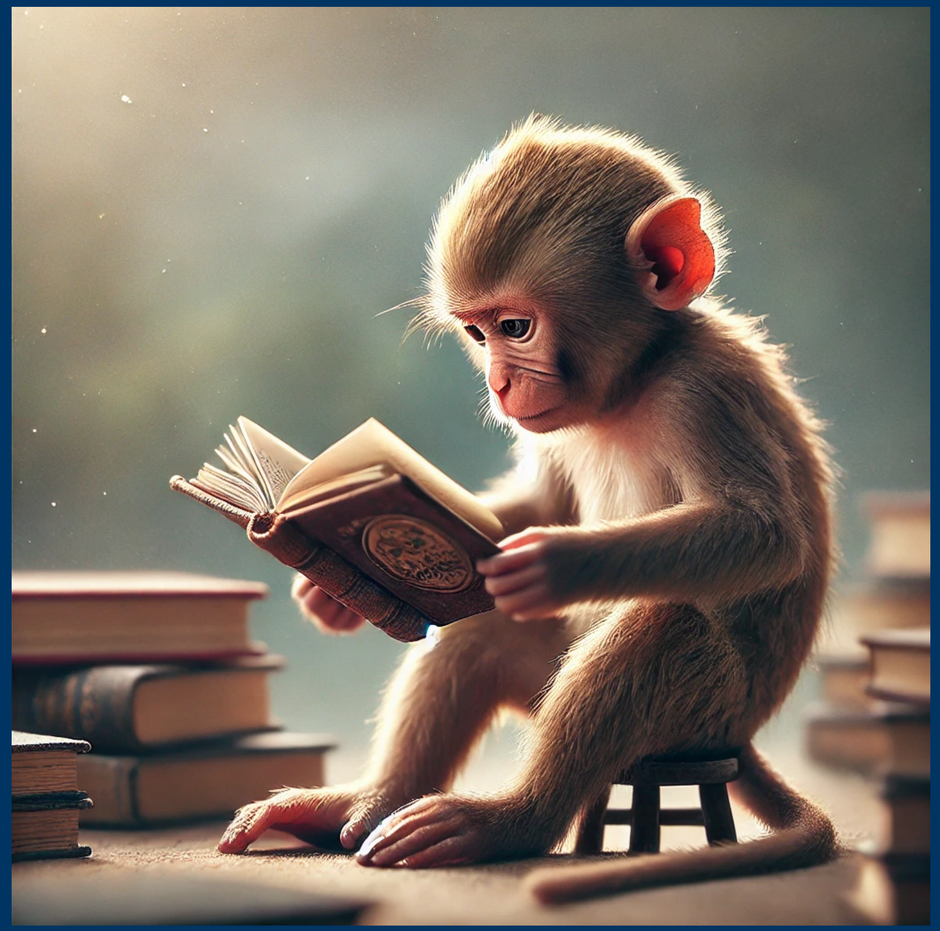
Query Throughput

- CPU & IO Constraints
- JDK 21+
- Generational ZGC
- Project Panama / Vector Api
  - Found in jvector!



More Efficient Read / Write IO

- BTI - New SSTable w/ Trie Indexes
- Trie Memtables
- Do our own read ahead (streaming, compaction)
- Direct I/O
- Virtual threads?



Compaction

- We need to keep up with writes
- Every release gets faster
- More CPU & memory efficient
- Unified Compaction Strategy is awesome

Compaction w/ Disaggregated Storage

- Separating Compute from Disk (EBS)
- Needed for K8 / Stateful Sets
- Today: Terrible Performance
- EBS: 1 op = read up to 256KB

```
TIME(s)     COMM           PID     DISK      T SECTOR     BYTES  LAT(ms)
0.000000    CompactionExec 26988   xvdb      R 48340000   16384     0.26
0.002350    CompactionExec 26988   xvdb      R 48339842   512       0.23
0.003560    CompactionExec 26988   xvdb      R 48339872   4096      0.22
0.003788    CompactionExec 26988   xvdb      R 48339864   4096      0.18
0.004550    CompactionExec 26988   xvdb      R 48339841   512       0.16
0.004719    CompactionExec 26988   xvdb      R 48339843   512       0.14
0.004906    CompactionExec 26988   xvdb      R 48339856   4096      0.15
0.005077    CompactionExec 26988   xvdb      R 48339848   4096      0.14
0.005304    CompactionExec 26988   xvdb      R 48340288   16384     0.17
0.009510    CompactionExec 26988   xvdb      R 48340320   16384     0.19
0.016991    NonPeriodicTas 26988   xvdb      R 32258112   16384     0.20
0.028200    CompactionExec 26988   xvdb      R 32262176   16384     0.27
0.029466    CompactionExec 26988   xvdb      R 32258944   16384     0.26
0.031511    CompactionExec 26988   xvdb      R 32226562   512       0.21
0.038502    CompactionExec 26988   xvdb      R 32226592   4096      0.22
0.038848    CompactionExec 26988   xvdb      R 32226584   4096      0.24
0.039842    CompactionExec 26988   xvdb      R 32226561   512       0.14
0.040079    CompactionExec 26988   xvdb      R 32226563   512       0.14
```

Improve disk access patterns during compaction and streaming (big format)

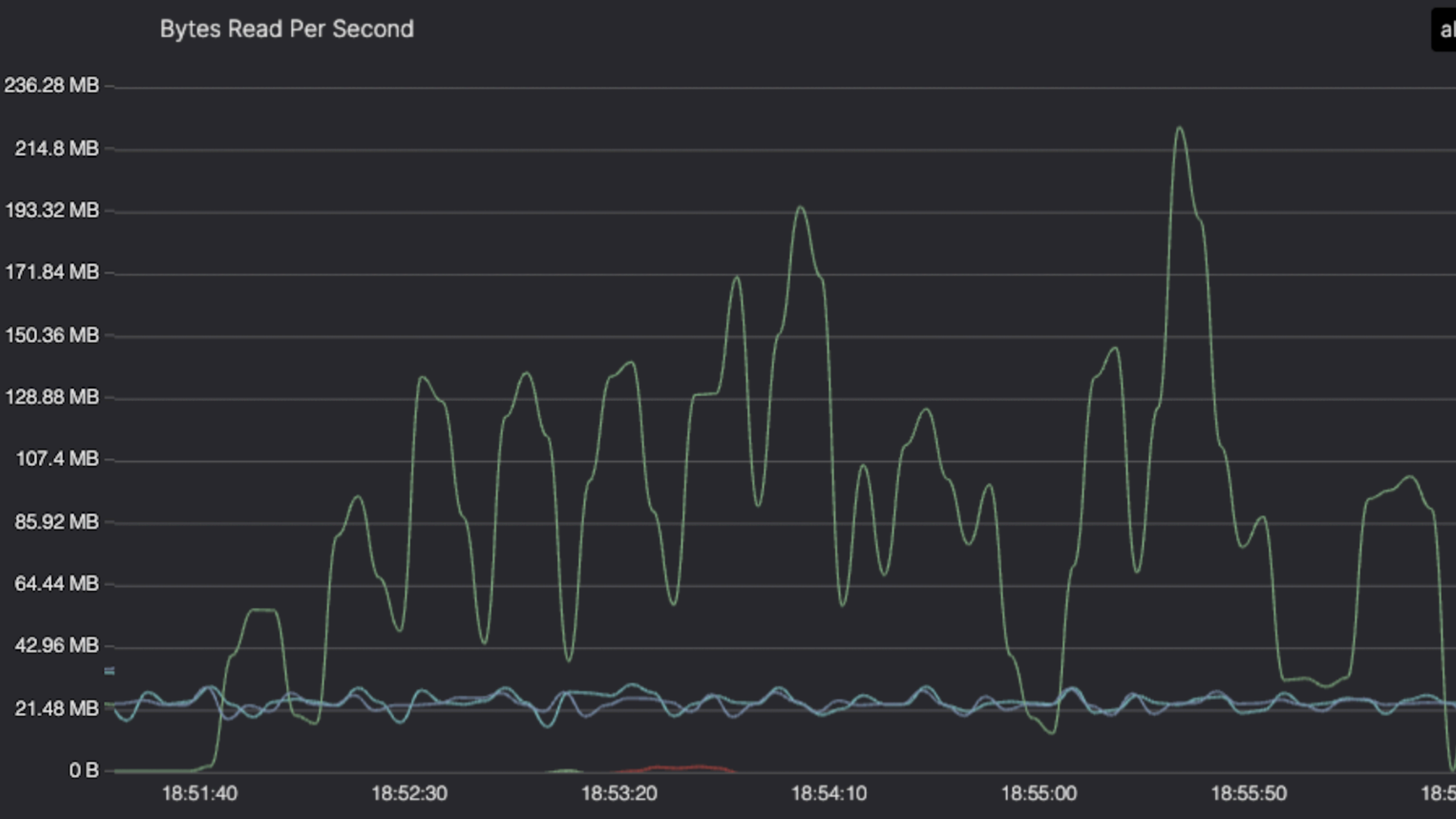
Details

Type:	Improvement	Status:	OPEN
Priority:	Normal	Resolution:	Unresolved
Component/s:	Legacy/Local Write-Read Paths, ... (1)	Fix Version/s:	None
Labels:	None		
Change Category:	Performance		
Complexity:	Normal		
Platform:	All		
Impacts:	None		

Description

On read heavy workloads Cassandra performs much better when using a low read ahead setting. In my tests I've seen an 5x improvement in throughput and more than a 50% reduction in latency. However, I've also observed that it can have a negative impact on compaction and streaming throughput. It especially negatively impacts cloud environments where small reads incur high costs in IOPS due to tiny requests.

1. We should investigate using POSIX_FADV_DONTNEED on files we're compacting to see if we can improve performance and


Bytes Read Per Second
236.28 MB
214.8 MB
193.32 MB
171.84 MB
150.36 MB
128.88 MB
107.4 MB
85.92 MB
64.44 MB
42.96 MB
21.48 MB
0 B
18:51:40
18:52:30
18:53:20
18:54:10
18:55:00
18:55:50

5x w/ same instance type

Instance Type	Single Instance Cost	EBS Cost	Num	Total
c5.9xlarge	$1,116.9	260	100	$137,690
c5.9xlarge	$1,116.9	900	20	$40,338

Instance vs EBS, 20TB

Instance Type	Instance Cost	EBS Cost	Total
i4i.32xlarge	$8,017	0	$8,017
c5.12xlarge	$1,489	$3,328	$4,818

Better Scheduling

- Work based scheduling instead of concurrent_*
- Reject work that can't be completed
- Dynamic Queue Depth
- Related ML Discussion
  - Generic Purpose Rate Limiter in Cassandra

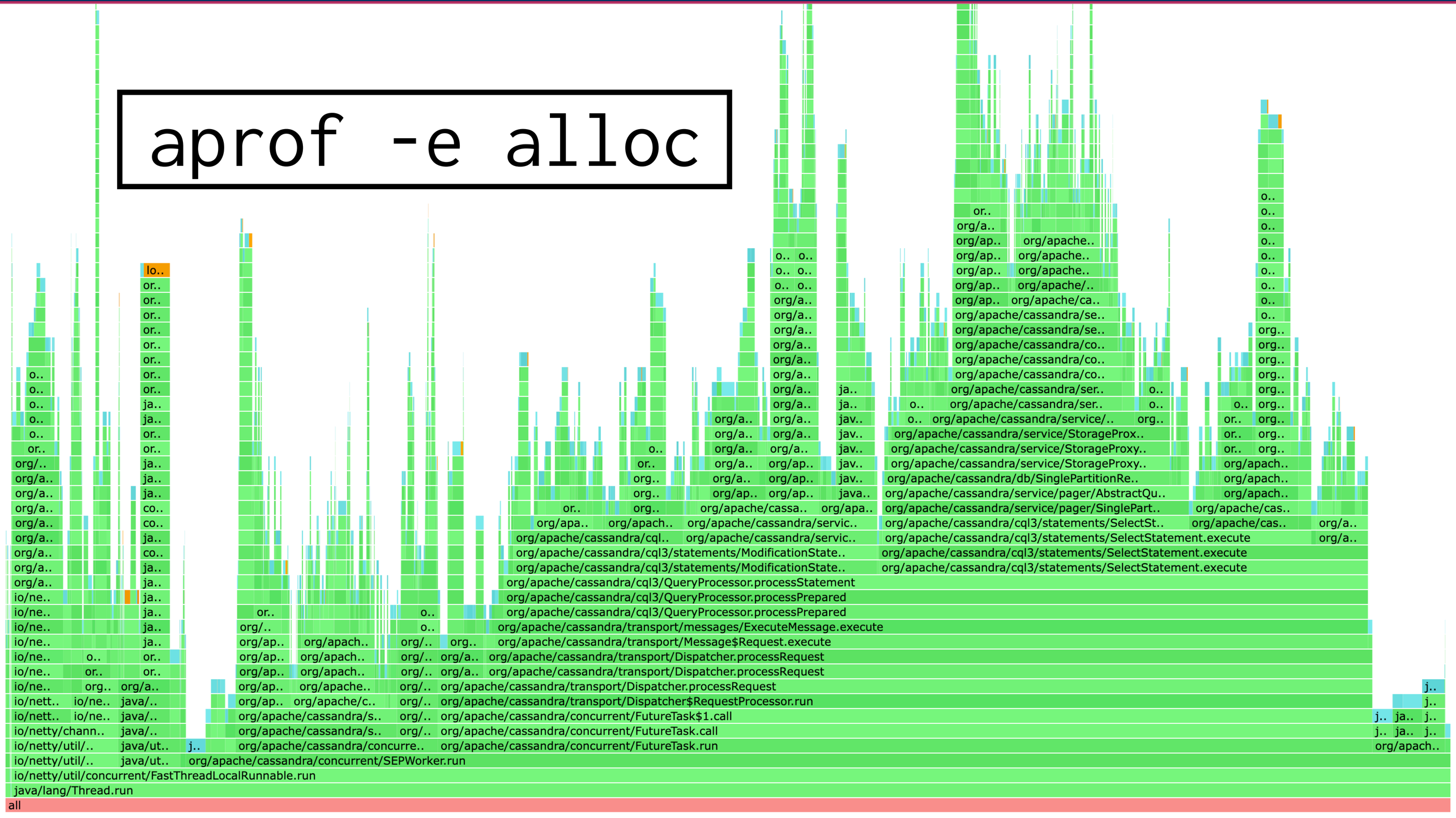
Multi-Tenancy

- It's Just Terrible
- Noisy Neighbors
- Thousands of tables
- We need per-resource rate limiting

Garbage!

- Individual allocations are cheap
- But.. allocations are not free
- Allocations on a hot path are EXPENSIVE
- GC still has to keep up
- TLAB is great but not huge or infinite

aprof -e alloc



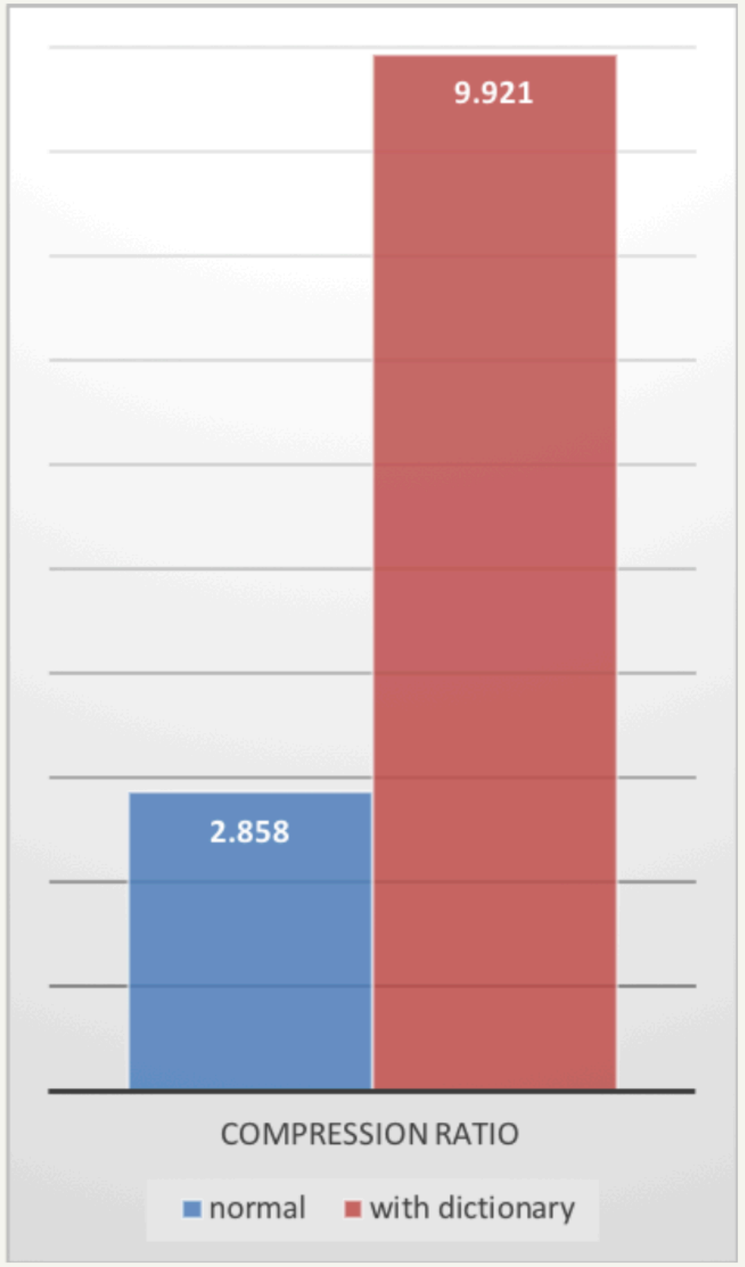
Arenas

- Java 21+
- MemorySegment, contiguous region of memory
- Shared or Confined
- On or off heap
- Isolate memory for requests, compaction

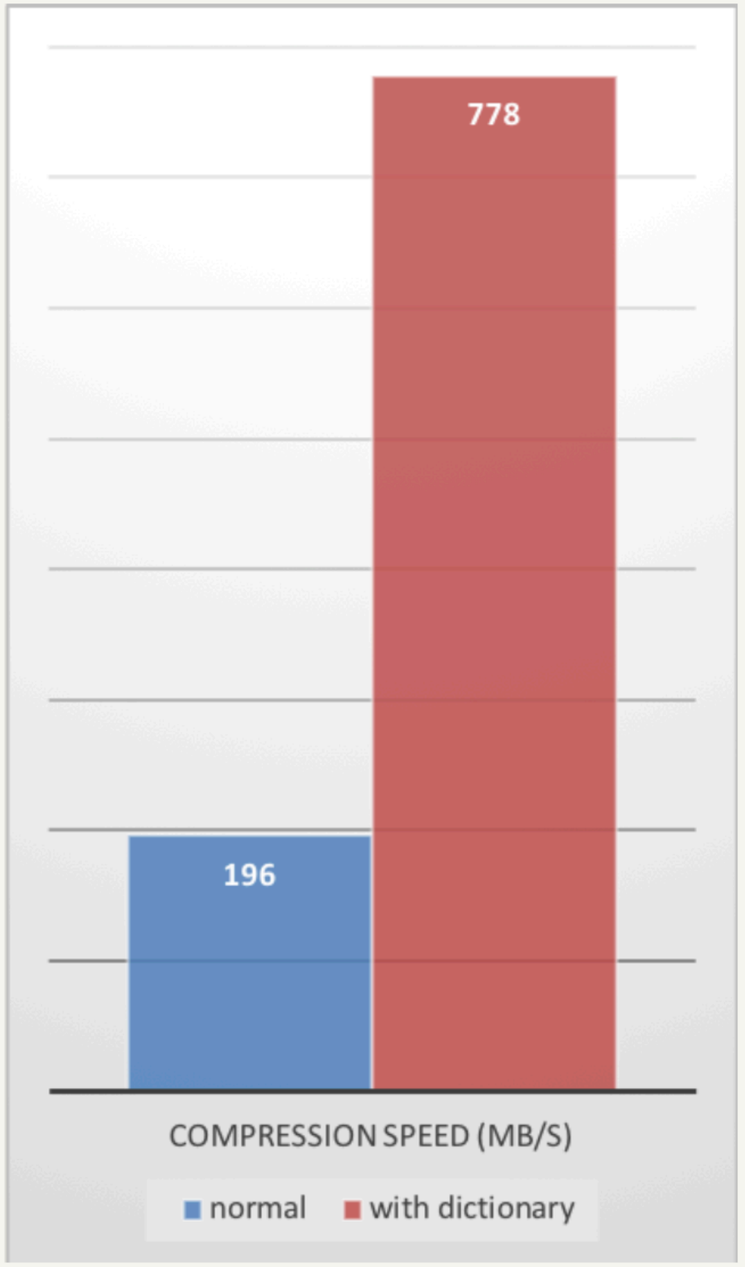
Compression

	LZ4	zSTD
BasicTimeSeries	0.95262	0.64973
KeyValue	0.96425	0.65043
KeyValue (book text)	0.73177	0.50157

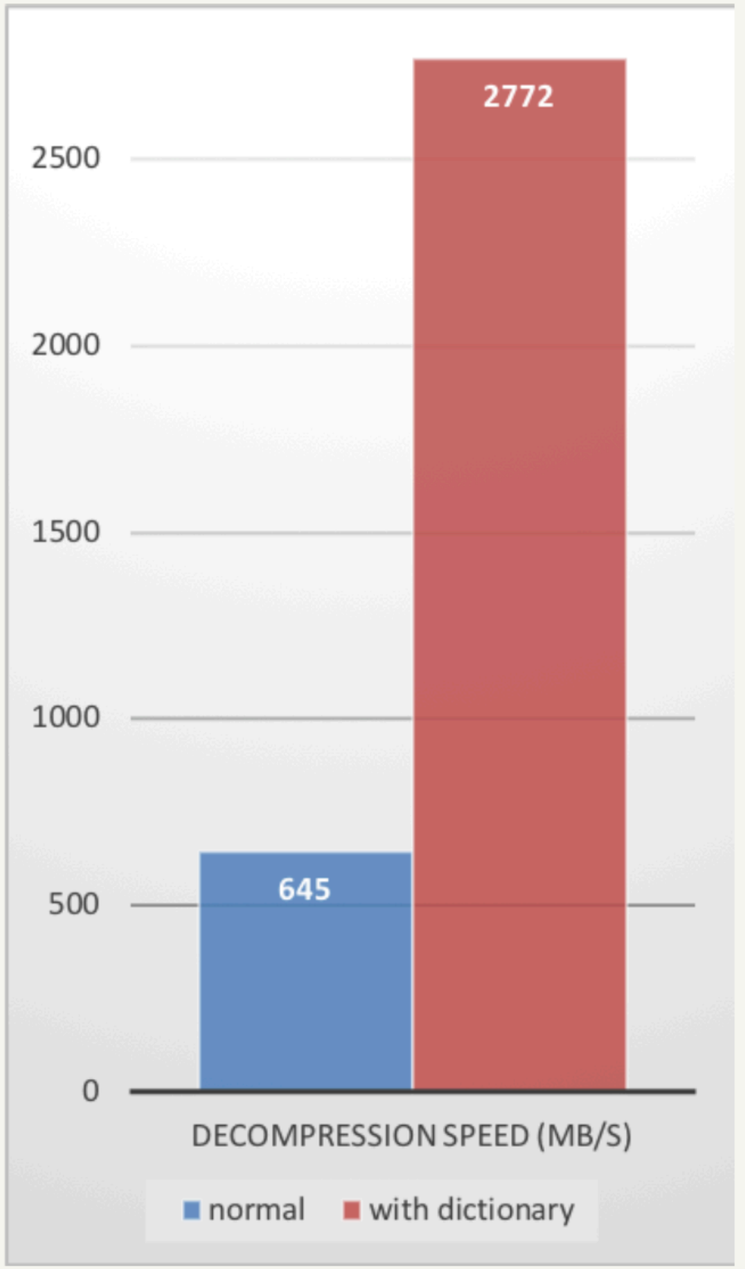
Compression Ratio



Compression Speed



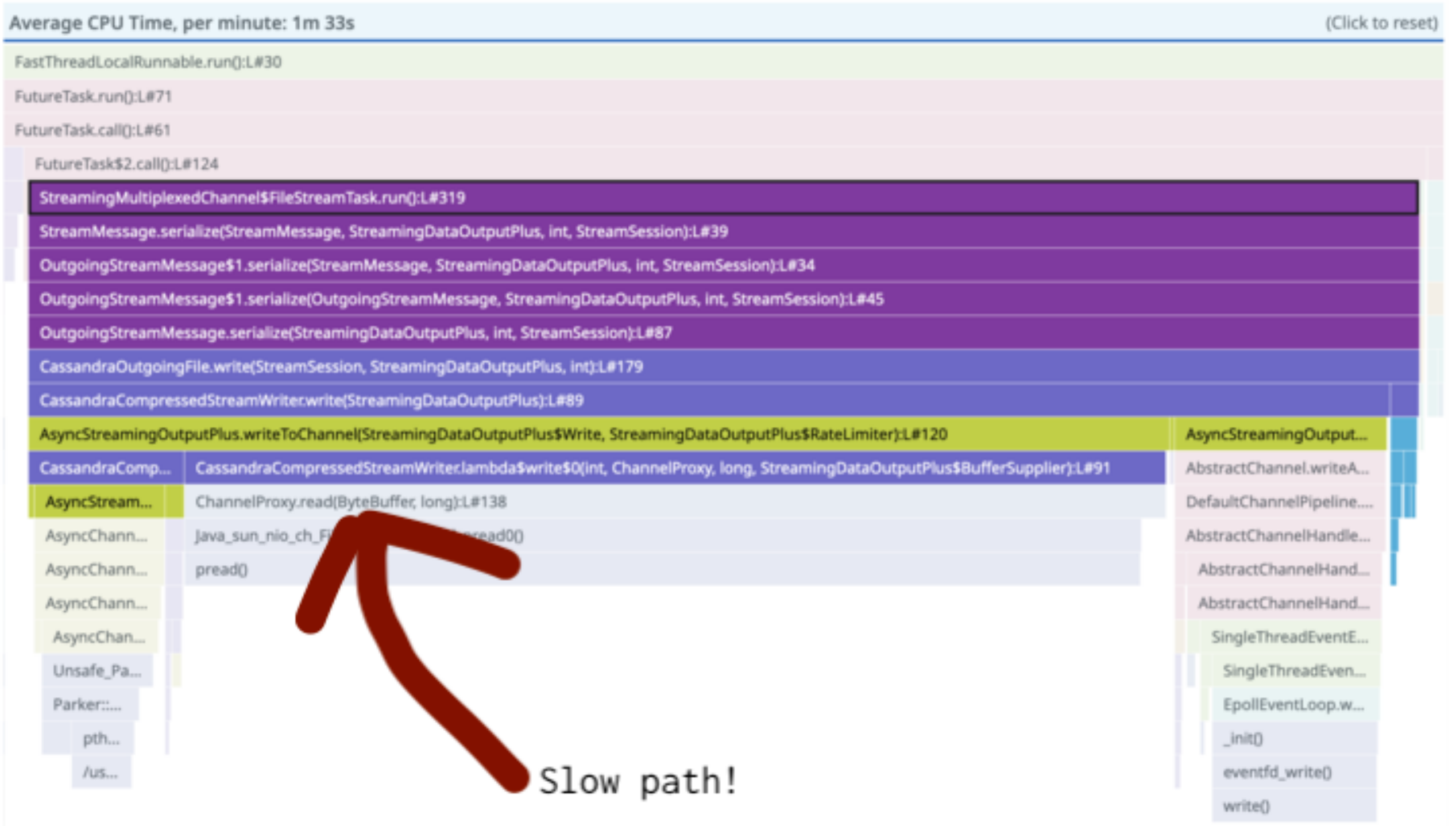
Decompression Speed



Streaming

Entire SSTable Streaming

- ZCS (Awesome perf, Unencrypted)
- Through JVM (Still awesome)
- Future: Kernel TLS!
- Best with LCS & UCS
- STCS / TWCS 😭 😭


Average CPU Time, per minute: 1m 33s
(Click to reset)
FastThreadLocalRunnable.run():L#30
FutureTask.run():L#71
FutureTask.call():L#61
FutureTask$2.call():L#124
StreamingMultiplexedChannel$FileStreamTask.run():L#319
StreamMessage.serialize(StreamMessage, StreamingDataOutputPlus, int, StreamSession):L#39
OutgoingStreamMessage$1.serialize(StreamMessage, StreamingDataOutputPlus, int, StreamSession):L#34
OutgoingStreamMessage$1.serialize(OutgoingStreamMessage, StreamingDataOutputPlus, int, StreamSession):L#45
OutgoingStreamMessage.serialize(StreamingDataOutputPlus, int, StreamSession):L#87
CassandraOutgoingFile.write(StreamSession, StreamingDataOutputPlus, int):L#179
CassandraCompressedStreamWriter.write(StreamingDataOutputPlus):L#89
AsyncStreamingOutputPlus.writeToChannel(StreamingDataOutputPlus$Write, StreamingDataOutputPlus$RateLimiter):L#120
AsyncStreamingOutput...
CassandraComp...
CassandraCompressedStreamWriter.lambda$write$0(int, ChannelProxy, long, StreamingDataOutputPlus$BufferSupplier):L#91
AbstractChannel.writeA...
AsyncStream...
ChannelProxy.read(ByteBuffer, long):L#138
DefaultChannelPipeline....
AsyncChann...
Java_sun_nio_ch_Fi...read0()
AbstractChannelHandle...
AsyncChann...
pread()
AbstractChannelHand...
AsyncChann...
AbstractChannelHand...
AsyncChan...
SingleThreadEventE...
Unsafe_Pa...
SingleThreadEven...
Parker::...
EpollEventLoop.w...
pth...
_init()
/us...
eventfd_write()
write()
Slow path!

Repair

- Same I/O problem as compaction!
- Incremental is great (>4.0)
- Full repair is still needed
- Should be direct IO

easy-cass-lab

Thank You!
Questions?