elastic{on} 2017

Stop Guessing, Start Measuring

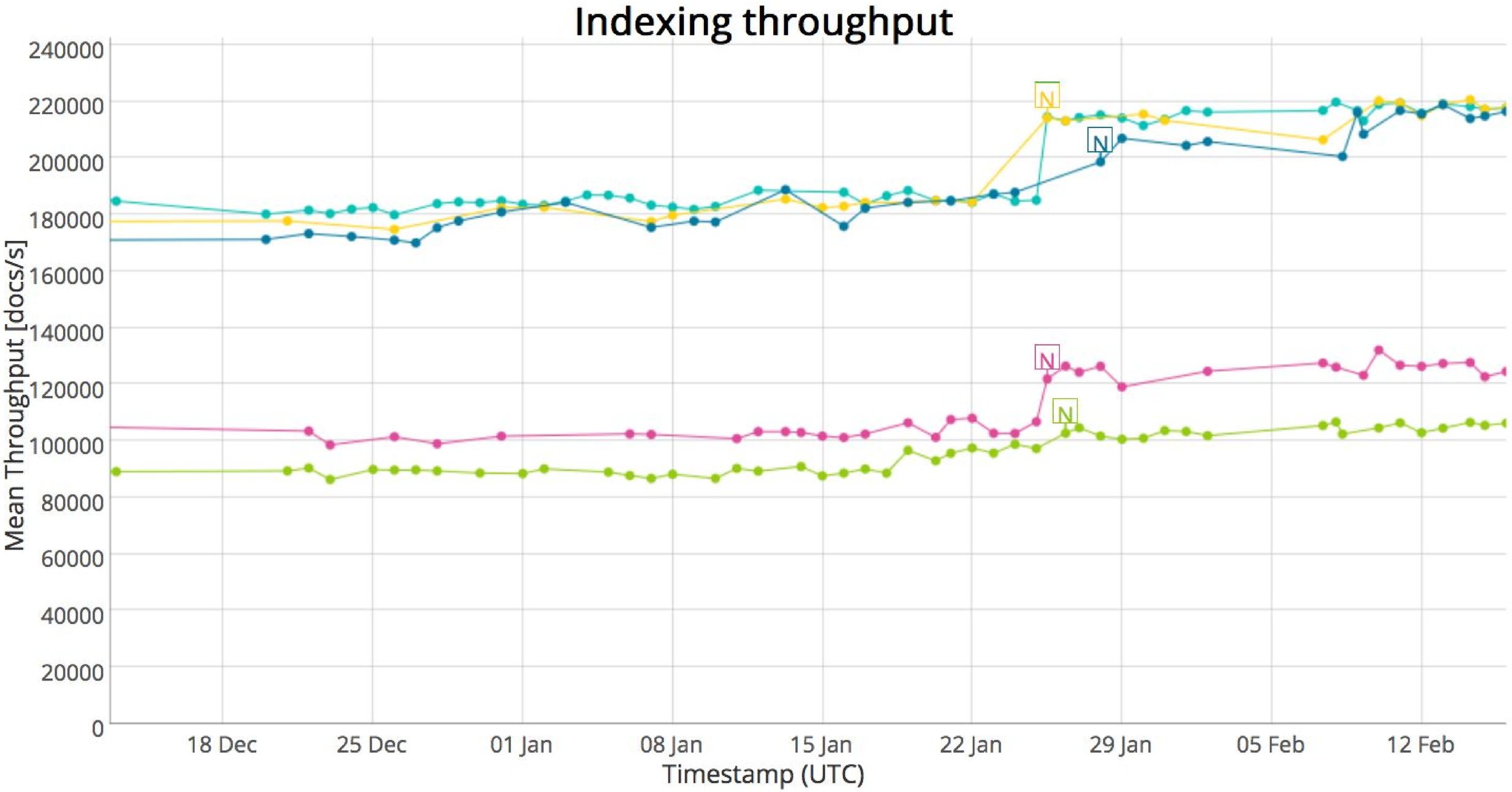
Getting Your Cluster Size Right with Rally Benchmarks

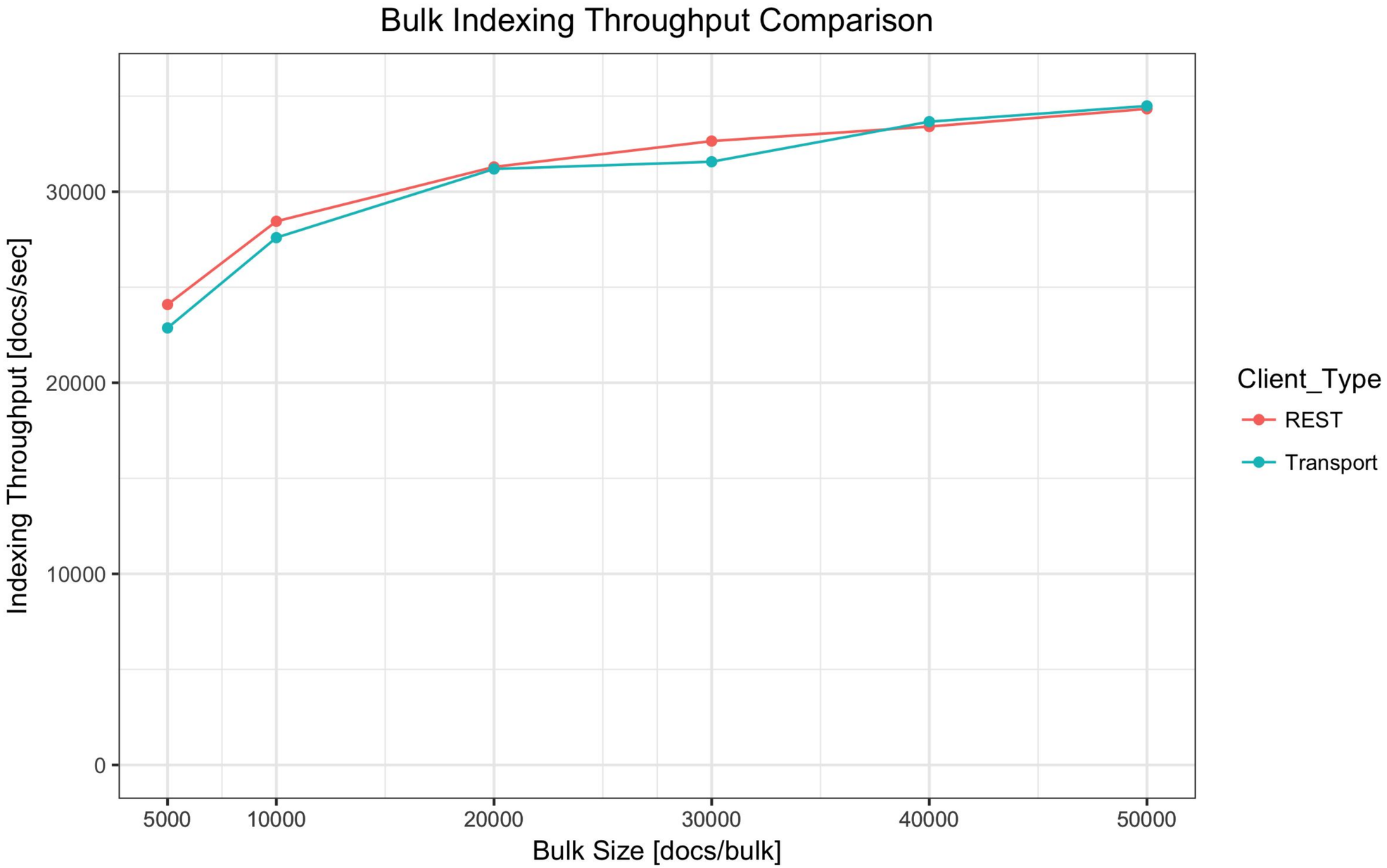
Christian Dahlqvist and Daniel Mitterdorfer

Agenda

1. Benchmarking at Elastic
2. A Whirlwind Tour of Rally
3. Rally in Practice

You Know, for Search


Indexing throughput
Mean Throughput [docs/s]
240000
220000
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
18 Dec
25 Dec
01 Jan
08 Jan
15 Jan
22 Jan
29 Jan
05 Feb
12 Feb
Timestamp (UTC)
N
N
N
N


Bulk Indexing Throughput Comparison
Indexing Throughput [docs/sec]
0
10000
20000
30000
5000
10000
20000
30000
40000
50000
Bulk Size [docs/bulk]
Client_Type
REST
Transport

Allocations

General | Allocation in New TLAB | Allocation Outside TLABs

Allocation by Class | Allocation by Thread | Allocation Profile

Allocation Profile

Stack Trace	Average Object Size	Total TLAB size	Pressure
java.lang.Thread.run()	974 bytes	6.89 GB	95.89%
java.util.concurrent.ThreadPoolExecutor$Worker.run()	975 bytes	6.88 GB	95.75%
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor$Worker)	975 bytes	6.88 GB	95.75%
org.elasticsearch.common.util.concurrent.AbstractRunnable.run()	997 bytes	6.62 GB	92.07%
org.elasticsearch.transport.TransportService$4.doRun()	998 bytes	6.62 GB	92.03%
org.elasticsearch.transport.RequestHandlerRegistry.processMessageReceived(TransportRequest, Transpo...	998 bytes	6.62 GB	92.03%
org.elasticsearch.transport.TransportRequestHandler.messageReceived(TransportRequest, TransportC...	1,002 bytes	6.59 GB	91.64%
org.elasticsearch.search.action.SearchServiceTransportAction$SearchQueryTransportHandler.mess...	1,002 bytes	6.59 GB	91.64%
org.elasticsearch.search.action.SearchServiceTransportAction$SearchQueryTransportHandler.me...	1,002 bytes	6.59 GB	91.64%
org.elasticsearch.search.SearchService.executeQueryPhase(ShardSearchRequest)	1,002 bytes	6.59 GB	91.64%
org.elasticsearch.search.SearchService.loadOrExecuteQueryPhase(ShardSearchRequest, S...	213 bytes	3.84 GB	53.37%
org.elasticsearch.search.query.QueryPhase.execute(SearchContext)	213 bytes	3.84 GB	53.37%
org.elasticsearch.search.SearchService.createAndPutContext(ShardSearchRequest)	1.79 kB	2.75 GB	38.27%
org.elasticsearch.action.support.nodes.TransportNodesAction$NodeTransportHandler.messageReceive...	24 bytes	15.35 MB	0.21%
org.elasticsearch.tasks.TaskManager.register(String, String, TransportRequest)	27 bytes	13.95 MB	0.19%
org.elasticsearch.discovery.zen.ping.unicast.UnicastZenPing$2.doRun()	80 bytes	1.40 MB	0.02%
org.elasticsearch.action.search.SearchQueryThenFetchAsyncAction$2.doRun()	40 bytes	1.20 MB	0.02%
org.jboss.netty.util.internal.DeadLockProofWorker$1.run()	101 bytes	121.52 MB	1.65%
org.elasticsearch.common.util.concurrent.PrioritizedEsThreadPoolExecutor$TieBreakingPrioritizedRunnable.run()	1.02 kB	79.57 MB	1.08%
org.elasticsearch.common.util.concurrent.PrioritizedEsThreadPoolExecutor$TieBreakingPrioritizedRunnable.r...	1.04 kB	78.18 MB	1.06%
org.elasticsearch.index.shard.StoreRecoveryService$1.run()	3.52 kB	25.17 MB	0.34%
java.util.concurrent.ThreadPoolExecutor.getTask()	38 bytes	17.14 MB	0.23%
org.elasticsearch.discovery.zen.ping.unicast.UnicastZenPing$3.run()	24 bytes	12.56 MB	0.17%
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run()	16 bytes	4.31 MB	0.06%
org.elasticsearch.transport.netty.NettyTransport$5.run()	53 bytes	4.19 MB	0.06%

General
Memory
Code
Threads
I/O
System
Events


12
OJB 93B

What is Rally?

Macrobenchmarking for Elasticsearch

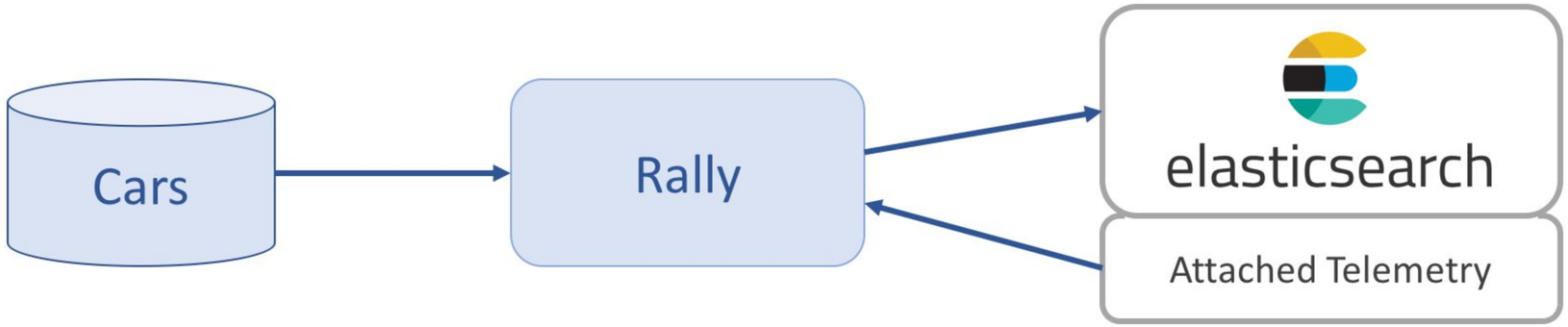
Think “JMeter on Steroids”

- Execute benchmarks based on Elasticsearch API
- Gather system metrics (CPU usage, disk I/O, GC ...) and attach “telemetry” for more insights
- Manage and provision Elasticsearch instances
- Structured storage for metrics

github.com/elastic/rally

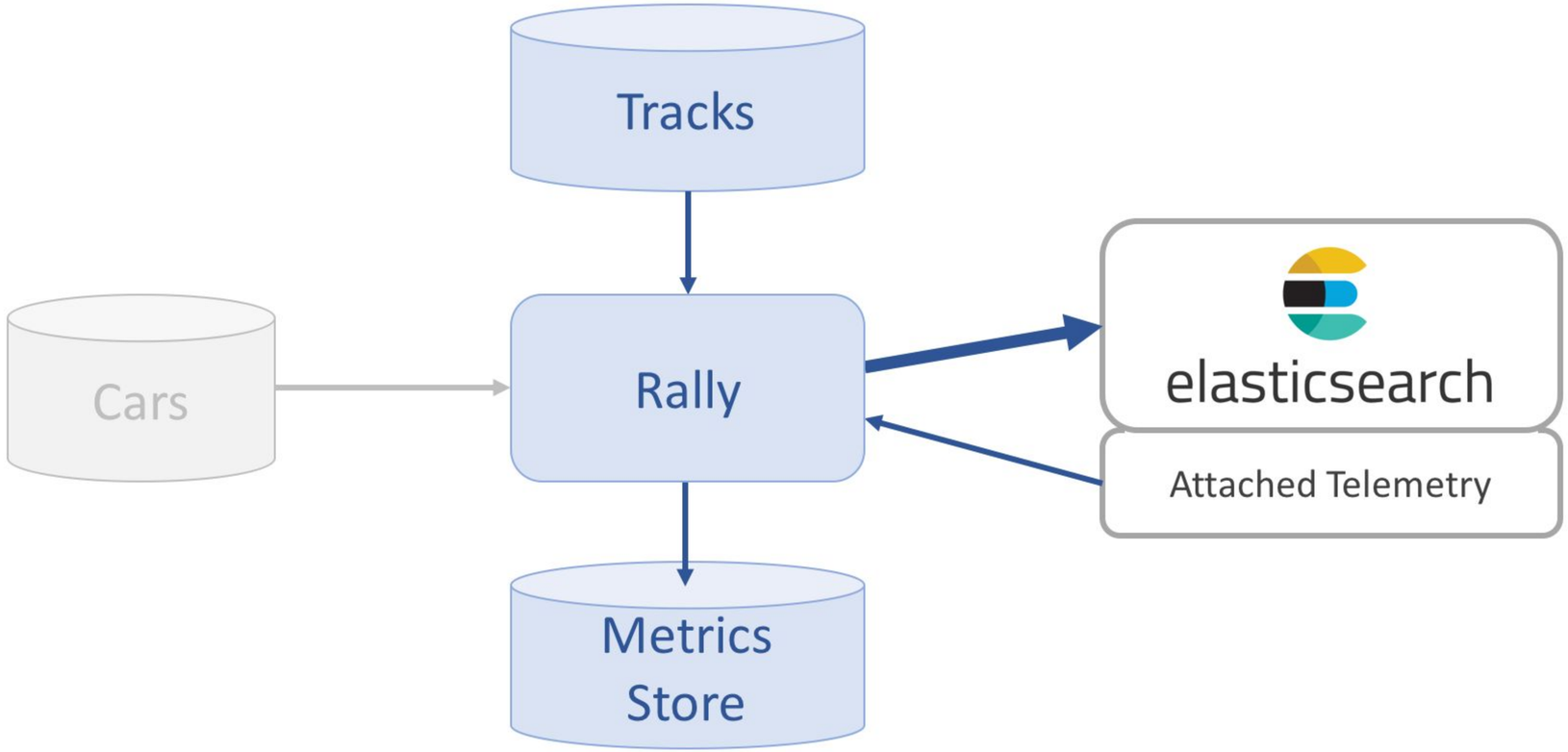
How does Rally work?

Part 1: Provisioning a cluster



How does Rally work?

Part 2: Running a benchmark



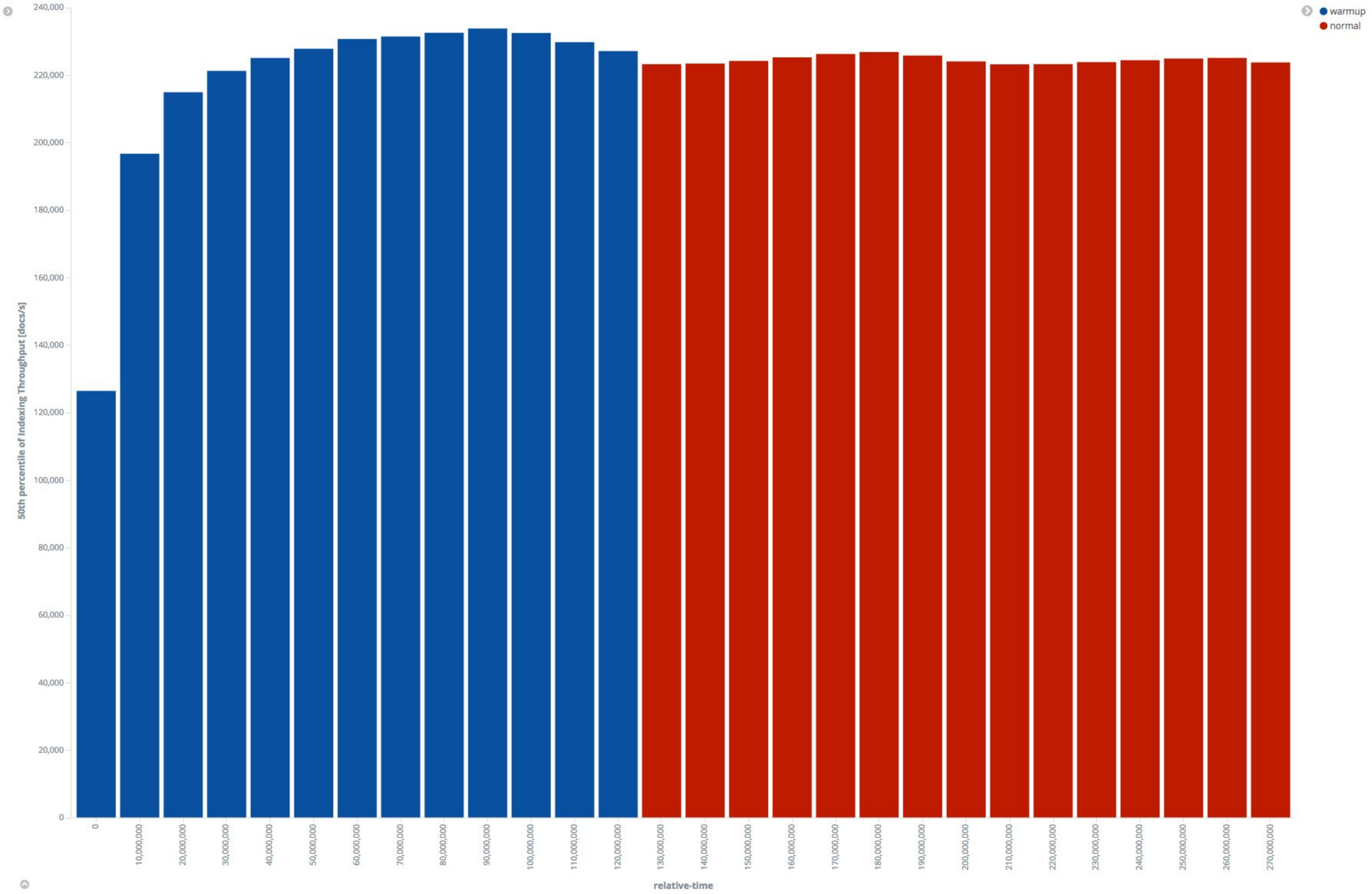
Inspecting Results

Summary Report

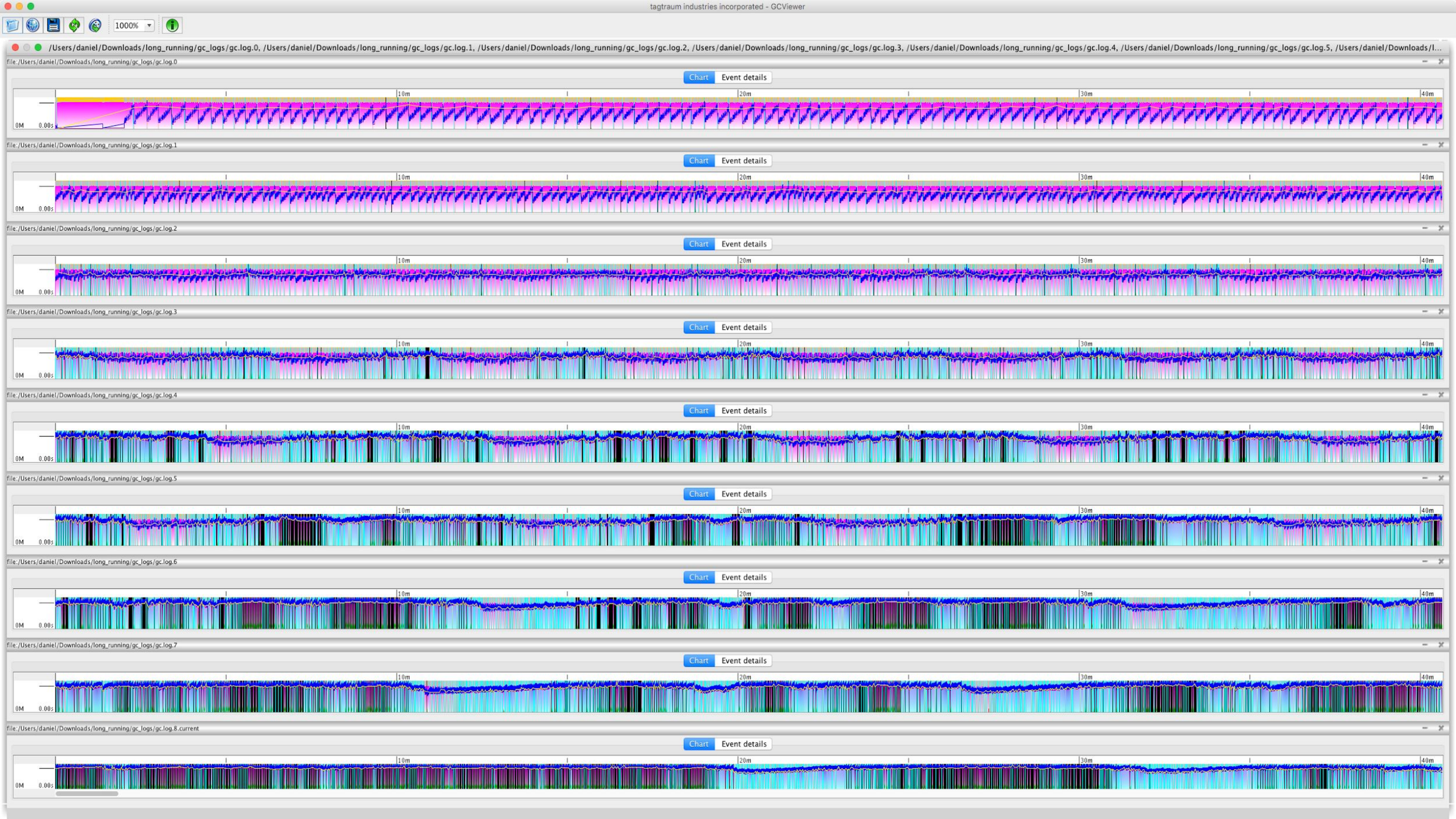
Metric	Operation	Value	Unit
Indexing time		124.712	min
Merge time		21.8604	min
Refresh time		4.49527	min
Merge throttle time		0.120433	min
Median CPU usage		546.5	%
Total Young Gen GC		72.078	s
Total Old Gen GC		3.426	s
Index size		2.26661	GB
Totally written		30.083	GB
…	…	…	…
99.9th percentile latency	index-update	2972.96	ms
99.99th percentile latency	index-update	4106.91	ms
100th percentile latency	index-update	4542.84	ms
99.9th percentile service time	index-update	2972.96	ms
99.99th percentile service time	index-update	4106.91	ms
100th percentile service time	index-update	4542.84	ms

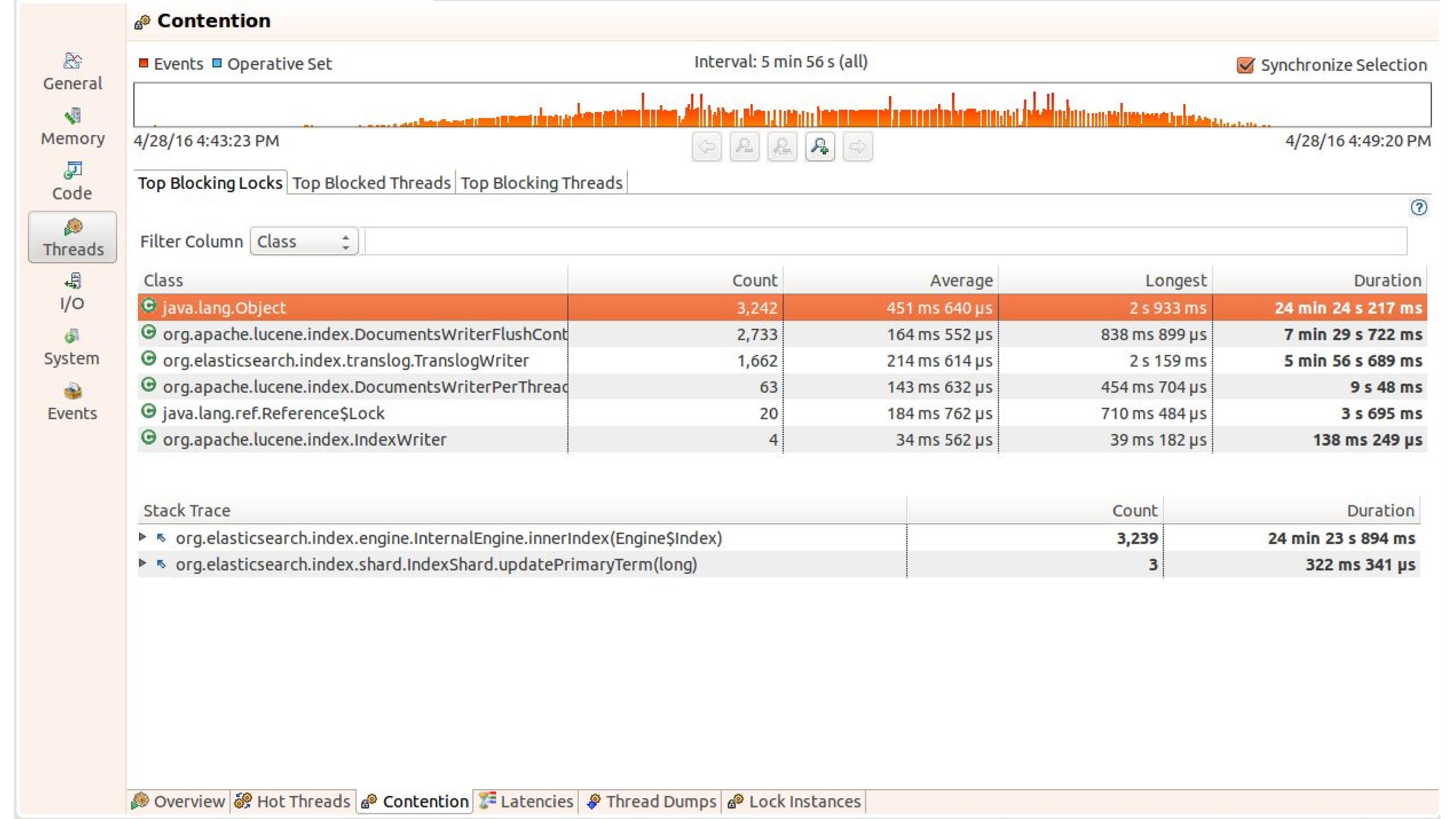
Metrics Records


```
{
  "trial-timestamp": "20170223T000046Z",
  "@timestamp": 1487811668093,
  "relative-time": 150148201,
  "track": "geonames",
  "challenge": "append-no-conflicts-index-only",
  "car": "4gheap",
  "sample-type": "normal",
  "name": "disk_io_write_bytes",
  "value": 12355731456,
  "unit": "byte",
  "meta": {
    "node_name": "rally-node0",
    "cpu_model": "Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz",
    "os_name": "Linux",
    "os_version": "4.4.0-38-generic",
    "jvm_vendor": "Oracle Corporation",
    "jvm_version": "1.8.0_101",
    "distribution_version": "6.0.0-alpha1",
    "source_revision": "18f57c0"
  }
}
```


warmup
normal
50th percentile of indexing Throughput [docs/s]
240,000
220,000
200,000
180,000
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
70,000,000
80,000,000
90,000,000
100,000,000
110,000,000
120,000,000
130,000,000
140,000,000
150,000,000
160,000,000
170,000,000
180,000,000
190,000,000
200,000,000
210,000,000
220,000,000
230,000,000
240,000,000
250,000,000
260,000,000
270,000,000
relative-time

Going Deeper:
Analyze Performance Issues


tagtraum industries incorporated - GCViewer
1000%
/Users/daniel/Downloads/long_running/gc_logs/gc.log.0, /Users/daniel/Downloads/long_running/gc_logs/gc.log.1, /Users/daniel/Downloads/long_running/gc_logs/gc.log.2, /Users/daniel/Downloads/long_running/gc_logs/gc.log.3, /Users/daniel/Downloads/long_running/gc_logs/gc.log.4, /Users/daniel/Downloads/long_running/gc_logs/gc.log.5, /Users/daniel/Downloads/l...
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.0
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.1
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.2
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.3
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.4
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.5
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.6
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.7
Chart
Event details
10m
20m
30m
40m
0M
0.00s
file:/Users/daniel/Downloads/long_running/gc_logs/gc.log.8.current
Chart
Event details
10m
20m
30m
40m
0M
0.00s


Contention
General
Memory
Code
Threads
I/O
System
Events
Events
Operative Set
Interval: 5 min 56 s (all)
Synchronize Selection
4/28/16 4:43:23 PM
4/28/16 4:49:20 PM
Top Blocking Locks
Top Blocked Threads
Top Blocking Threads
Filter Column
Class
Class
Count
Average
Longest
Duration
java.lang.Object
3,242
451 ms 640 µs
2 s 933 ms
24 min 24 s 217 ms
org.apache.lucene.index.DocumentsWriterFlushCont
2,733
164 ms 552 µs
838 ms 899 µs
7 min 29 s 722 ms
org.elasticsearch.index.translog.TranslogWriter
1,662
214 ms 614 µs
2 s 159 ms
5 min 56 s 689 ms
org.apache.lucene.index.DocumentsWriterPerThread
63
143 ms 632 µs
454 ms 704 µs
9 s 48 ms
java.lang.ref.Reference$Lock
20
184 ms 762 µs
710 ms 484 µs
3 s 695 ms
org.apache.lucene.index.IndexWriter
4
34 ms 562 µs
39 ms 182 µs
138 ms 249 µs
Stack Trace
Count
Duration
org.elasticsearch.index.engine.InternalEngine.innerIndex(Engine$Index)
3,239
24 min 23 s 894 ms
org.elasticsearch.index.shard.IndexShard.updatePrimaryTerm(long)
3
322 ms 341 µs
Overview
Hot Threads
Contention
Latencies
Thread Dumps
Lock Instances

Rally in Practice

How to use and extend for ‘realistic’ benchmarks

Why benchmark?

What insights are we looking for?

Cluster size required
to support use-case

What hardware to use

Cluster behaviour
under varying load

Optimal cluster
configuration

Benchmarking and use-cases

Common patterns

Search use-cases

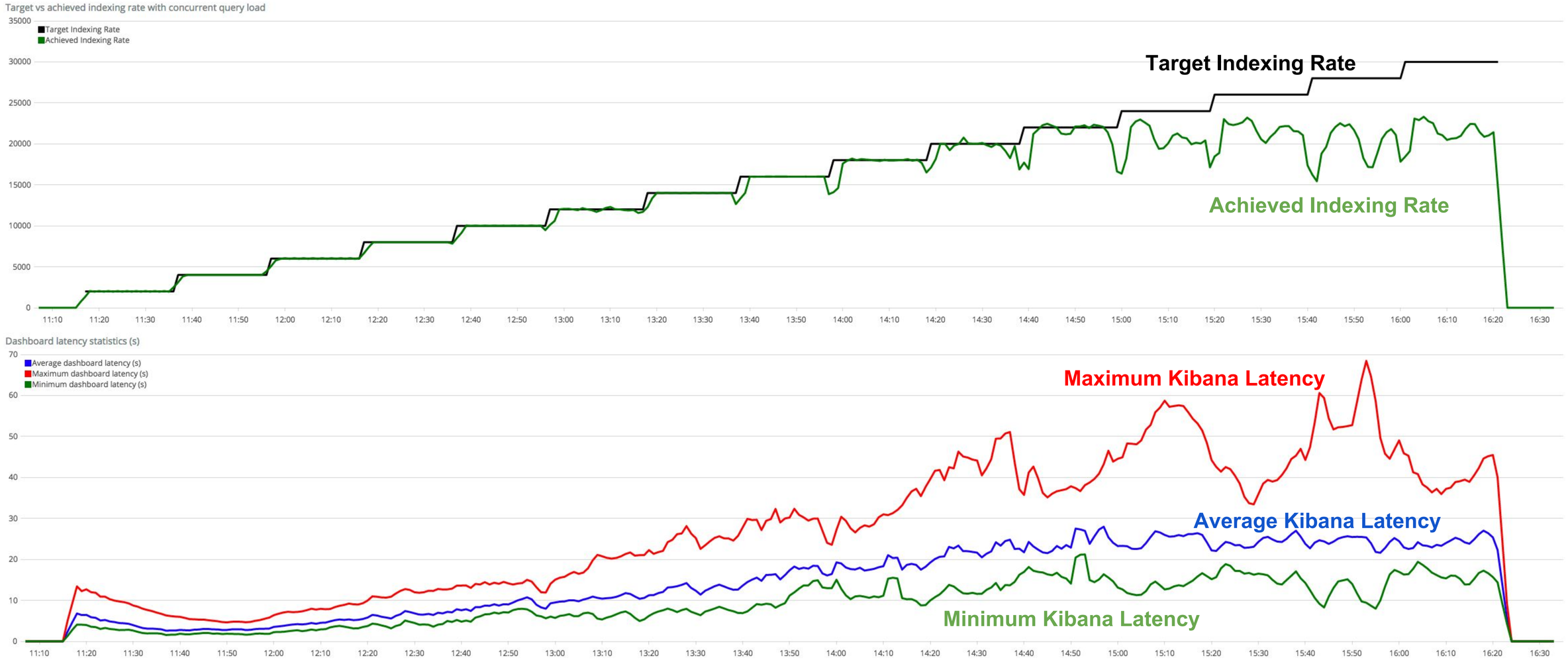
- Complex queries
- Complex data models
- Limited indexing
- Latency sensitive

Event-based use-cases

- Indexing heavy
- Flat data model
- Analysis through Kibana
- Limited other querying

Why more complex benchmarks?

How does different types of load interact?



Introducing rally-eventdata-track

(www.github.com/elastic/rally-eventdata-track)

What do we need?

Data generation

- Support long benchmarks
- Rate-limiting
- Configurable timestamp

Simulate Kibana usage

- Configurable
- More realistic load patterns
- Easy to get started

Easy to use and extend

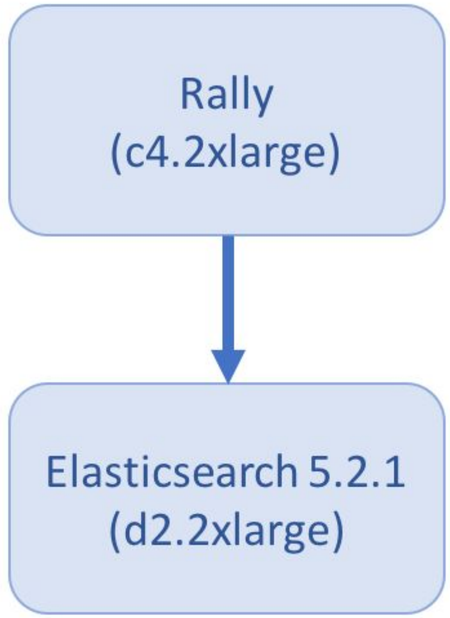
- Run it as-is
- Adapt to your scenario
- Use as inspiration

Example: Using the track to evaluate hardware

How performant are d2.2xlarge instances?

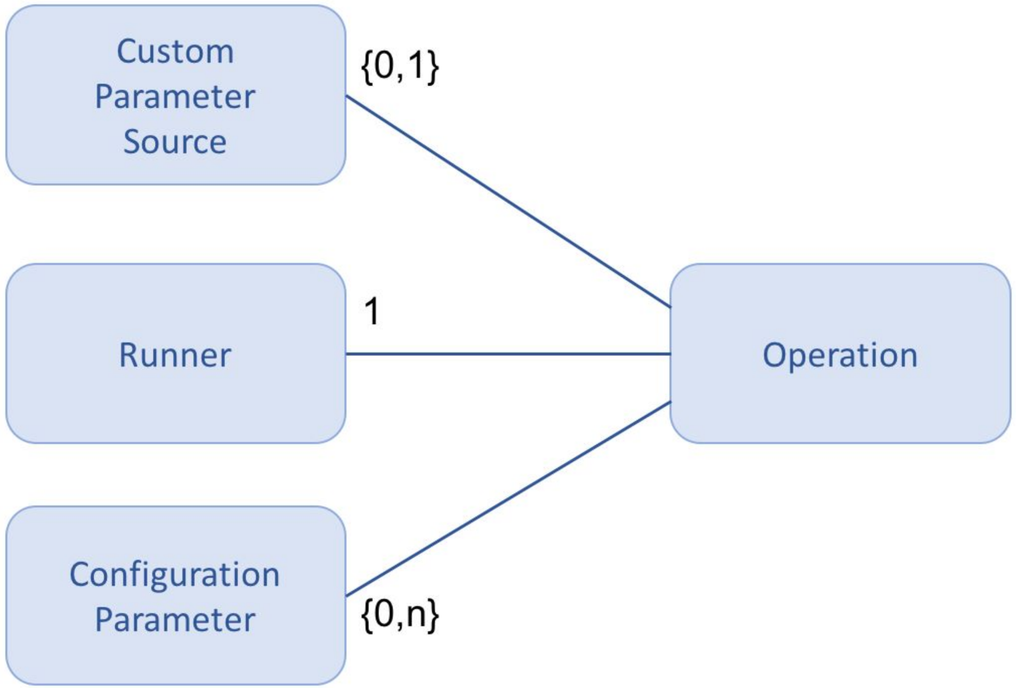
Why d2 instances?

- **_shrink** and **_rollover** APIs add flexibility
- 8 CPU cores, 61GB RAM
- 6 2TB disks in RAID10 => ~6TB storage
- Separate instance for Rally - CPU intensive



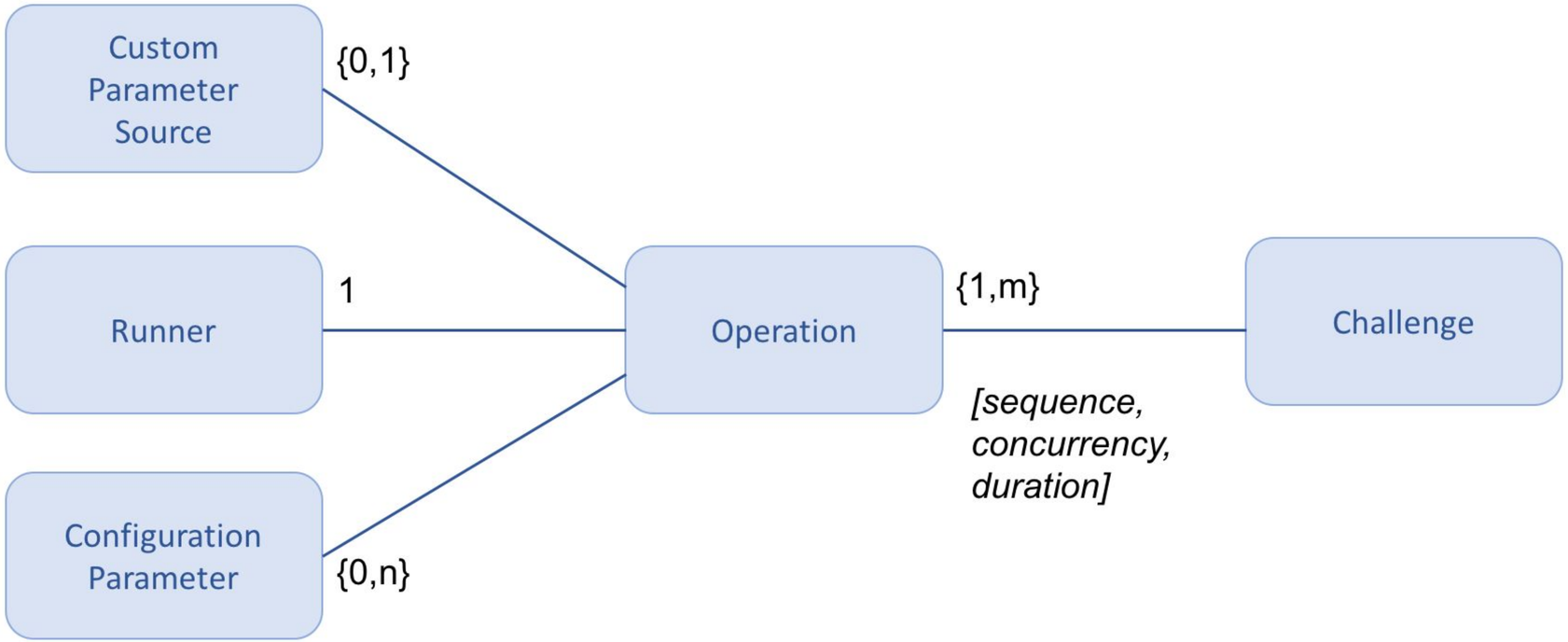
Important Rally concepts

The structure behind the benchmarks



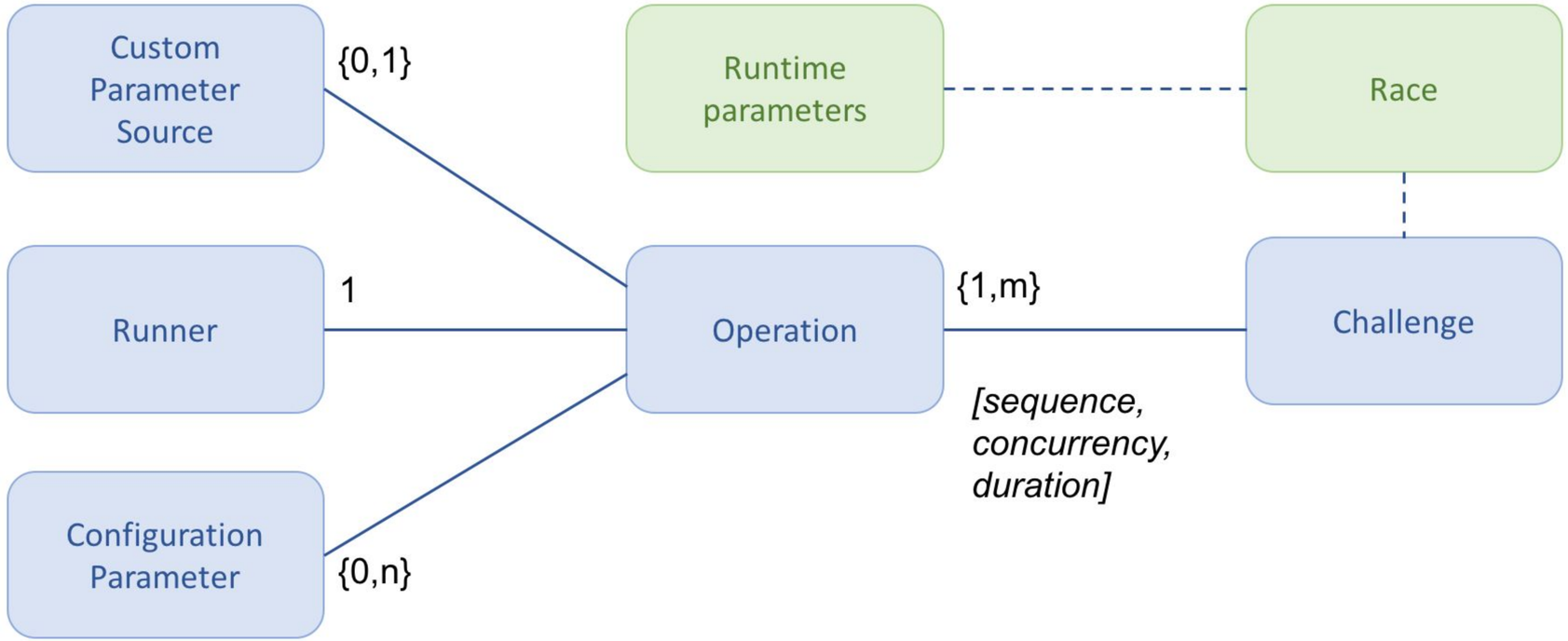
Important Rally concepts

The structure behind the benchmarks



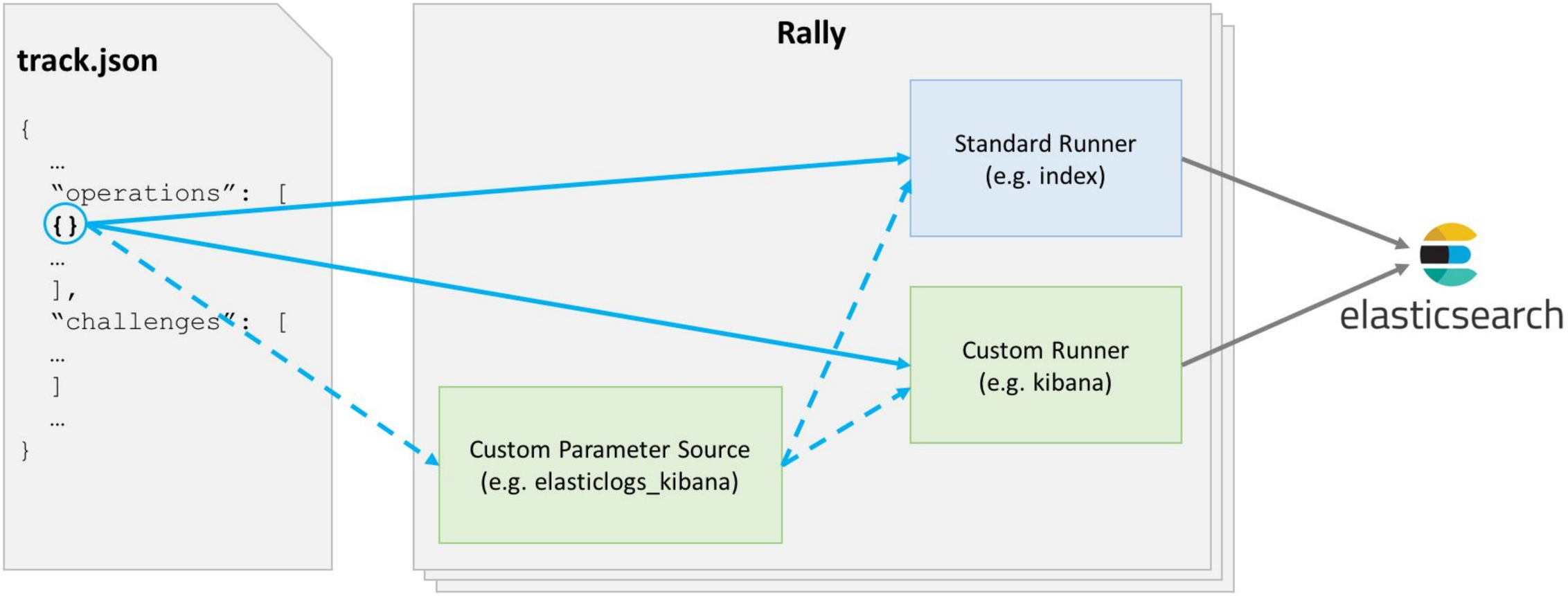
Important Rally concepts

The structure behind the benchmarks



Flow of data and configuration

Anatomy of a track



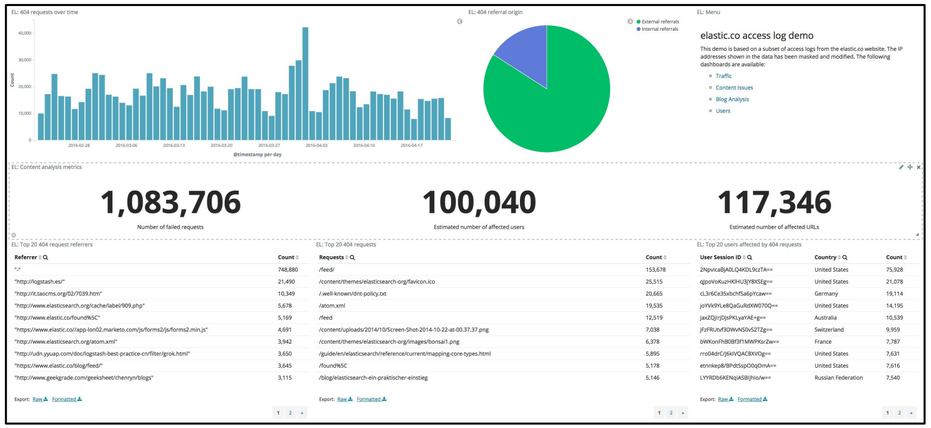
Bulk index data generator

Unbounded volumes of access log data

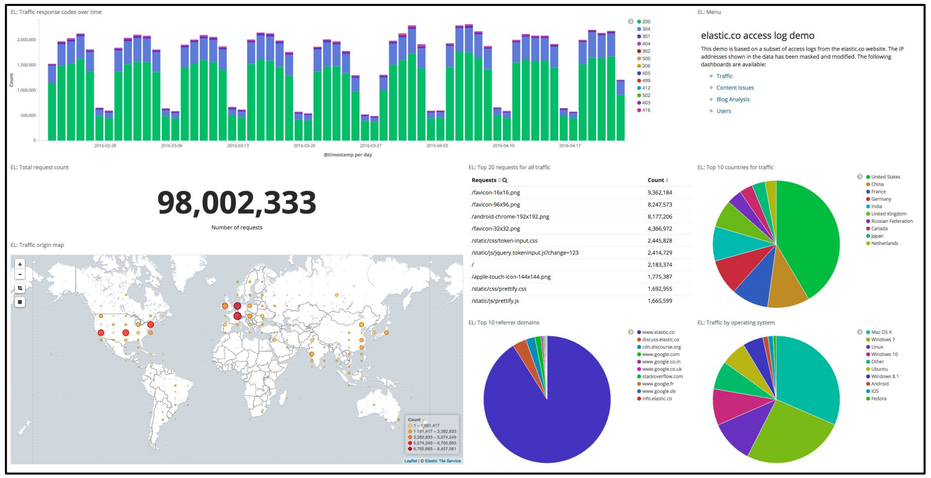

```
{
  "agent": "Mozilla/5.0 (Windows NT 6.3; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0",
  "useragent": {
    "os": "Windows 8.1",
    "os_name": "Windows 8.1",
    "name": "Firefox"
  },
  "geoip": {
    "country_name": "Canada",
    "location": [-95, 60]
  },
  "clientip": "184.151.239.181",
  "referrer": "-",
  "request": "/favicon-16x16.png?change=123",
  "bytes": 1763,
  "verb": "GET",
  "response": 200,
  "httpversion": "1.1",
  "@timestamp": "2017-02-22T13:09:06.343Z",
  "message": "184.151.239.181 - - [2017-02-22T13:09:06.343Z] \"GET /favicon-16x16.png?change=123
HTTP/1.1\" 200 1763 \"-\" \"-\" \"Mozilla/5.0 (Windows NT 6.3; WOW64; rv:42.0) Gecko/20100101
Firefox/42.0\""
}
```

Simulating Kibana queries

2 Out-of-the-box simulated Kibana dashboards



- **Content Issues** Dashboard
- Internal/external missing link analysis
- Analyses subset of data
- Lightweight



- **Traffic** Dashboard
- Traffic pattern analysis
- Analyses all data
- Heavyweight

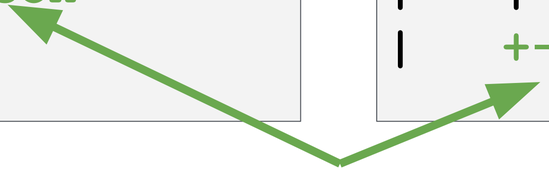
How should I use it?

Fork and extend the track

Dynamically loads files from directories

```
eventdata
|-- track.json
|-- track.py
|-- mappings.json
|-- operations
|    |-- indexing.json
|    |-- querying.json
|    |-- stats.json
|    +-- my_operations.json
...
```

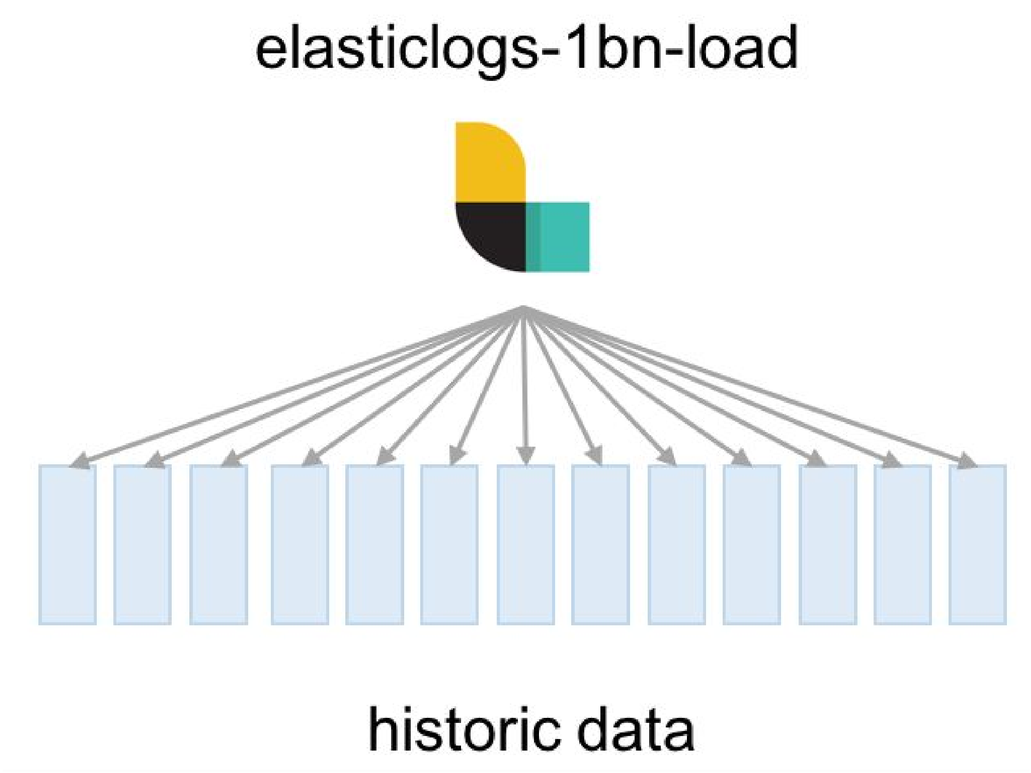
```
...
|-- parameter_sources
|    +-- [custom parameter sources]
|-- runners
|    +-- [custom runners]
|-- challenges
|    |-- bulk-size-evaluation.json
|    |-- elasticlogs-1bn-load.json
|    |-- shard-sizing.json
|    +-- my_challenges.json
```



Add files with new operation and challenge definition files - no conflicts

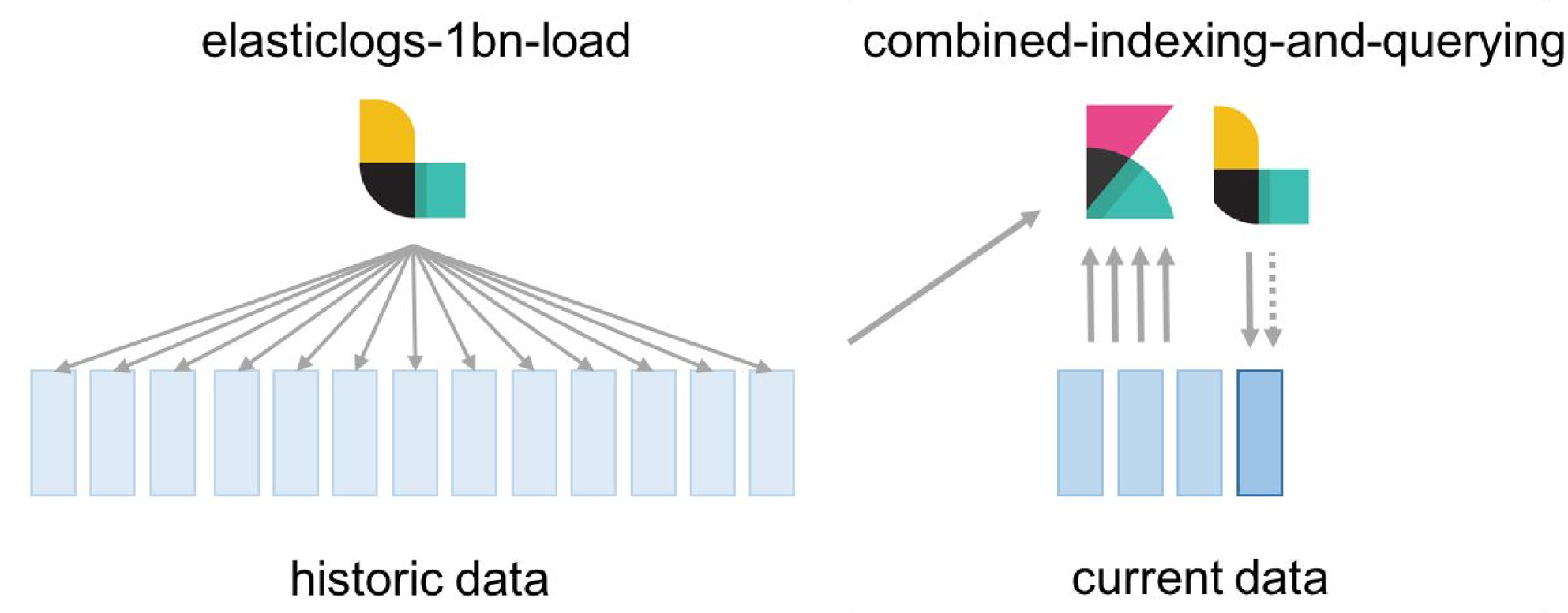
Example Challenges

Combining indexing and querying



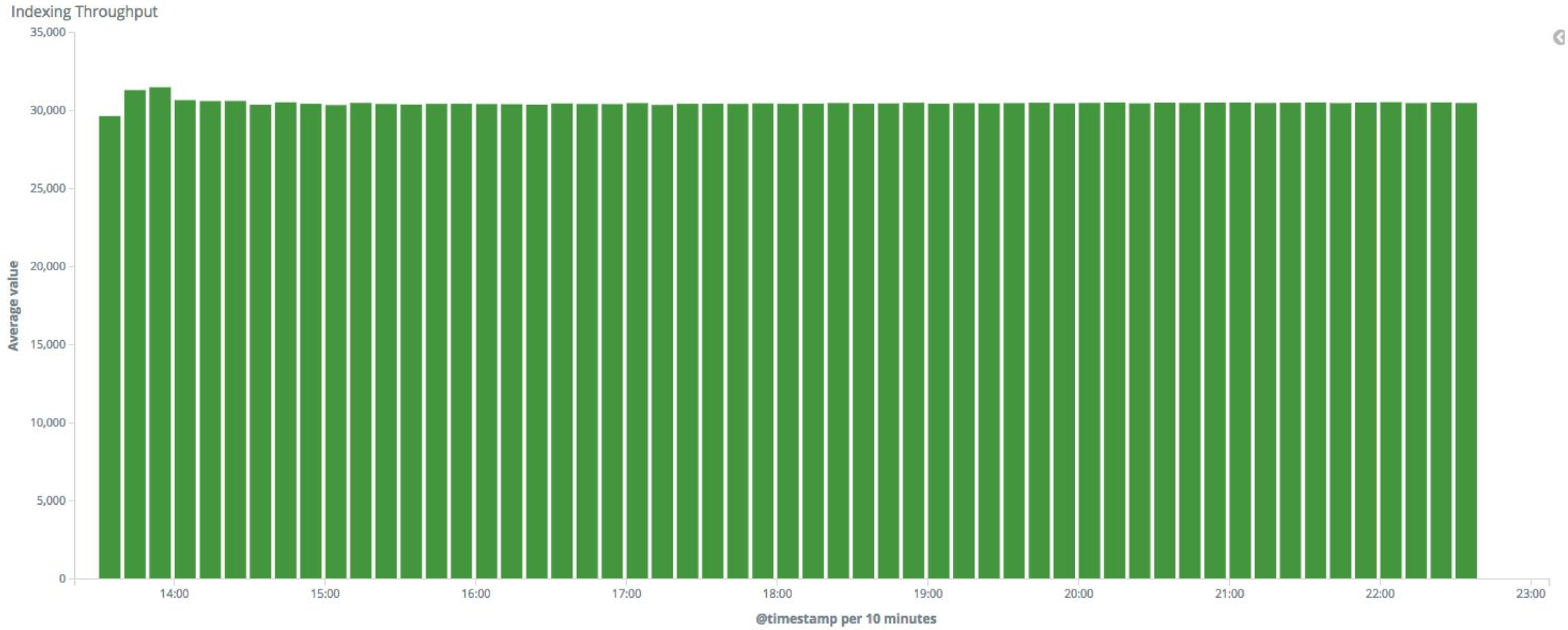
Example Challenges

Combining indexing and querying



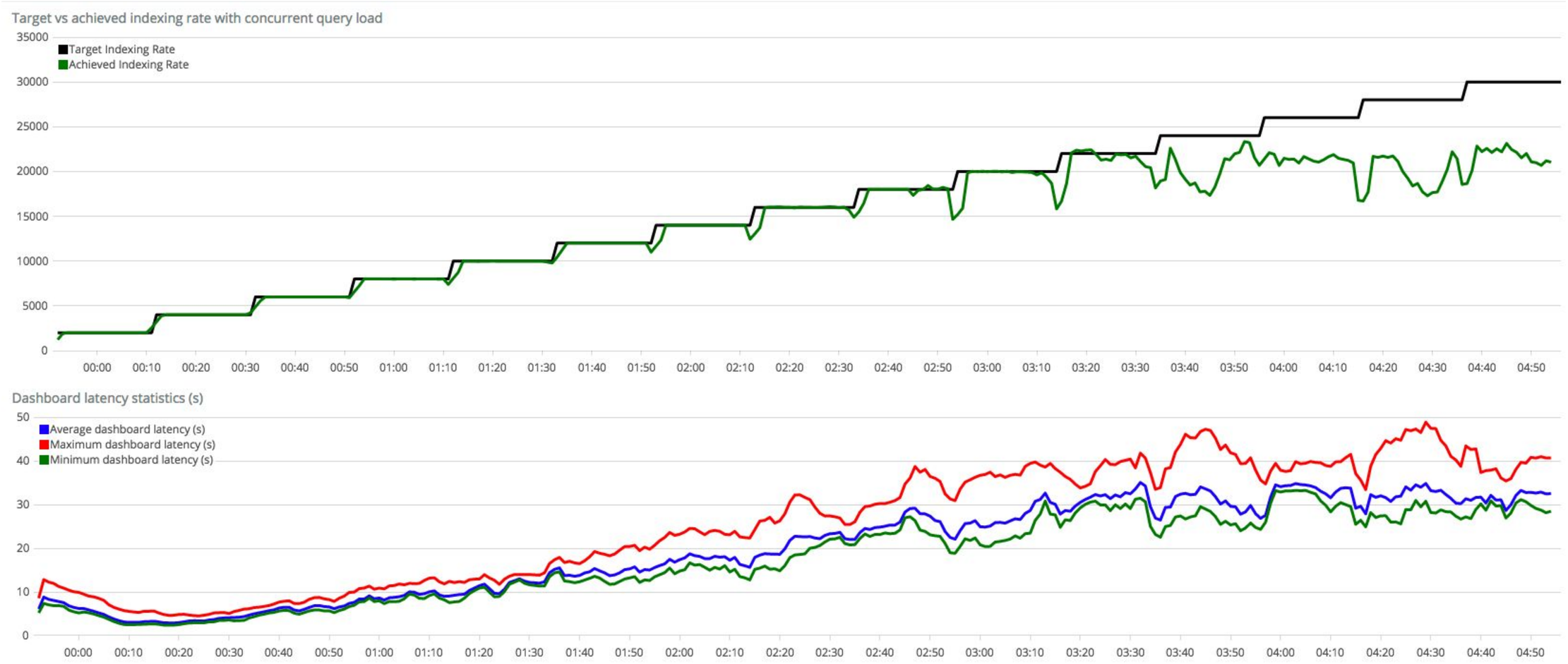
Indexing performance

elasticlogs-1bn-load benchmark



Combined indexing and querying

combined-indexing-and-querying benchmark



Take it for a spin!!

Help us take it to the next level!

More Questions?

Visit us at the AMA
or
Discuss in “BoF: Benchmarking Elasticsearch” today at 12:45

www.elastic.c
o

Image Credits

- “measuring tape” by Sean MacEntee: https://www.flickr.com/photos/smemon/14618772953/ (CC BY 2.0)
- “Works Mini Cooper S DJB 93B” by Andrew Basterfield:

  https://www.flickr.com/photos/andrewbasterfield/4759364589/ (CC BY-SA 2.0)

Please attribute Elastic with a link to elastic.co