Seven Golden Rules for Benchmarking Elasticsearch

Daniel Mitterdorfer
@dmitterd


elastic
JOE
Cola

What is Benchmarking?

Characteristics

- **Run** a well-defined workload
- **Measure** performance metrics
- **Change** a parameter
- **Compare** results

Rule One

Think about System Setup

Relevancy

Be close to production

- Same hardware
- Same software
- Same configuration

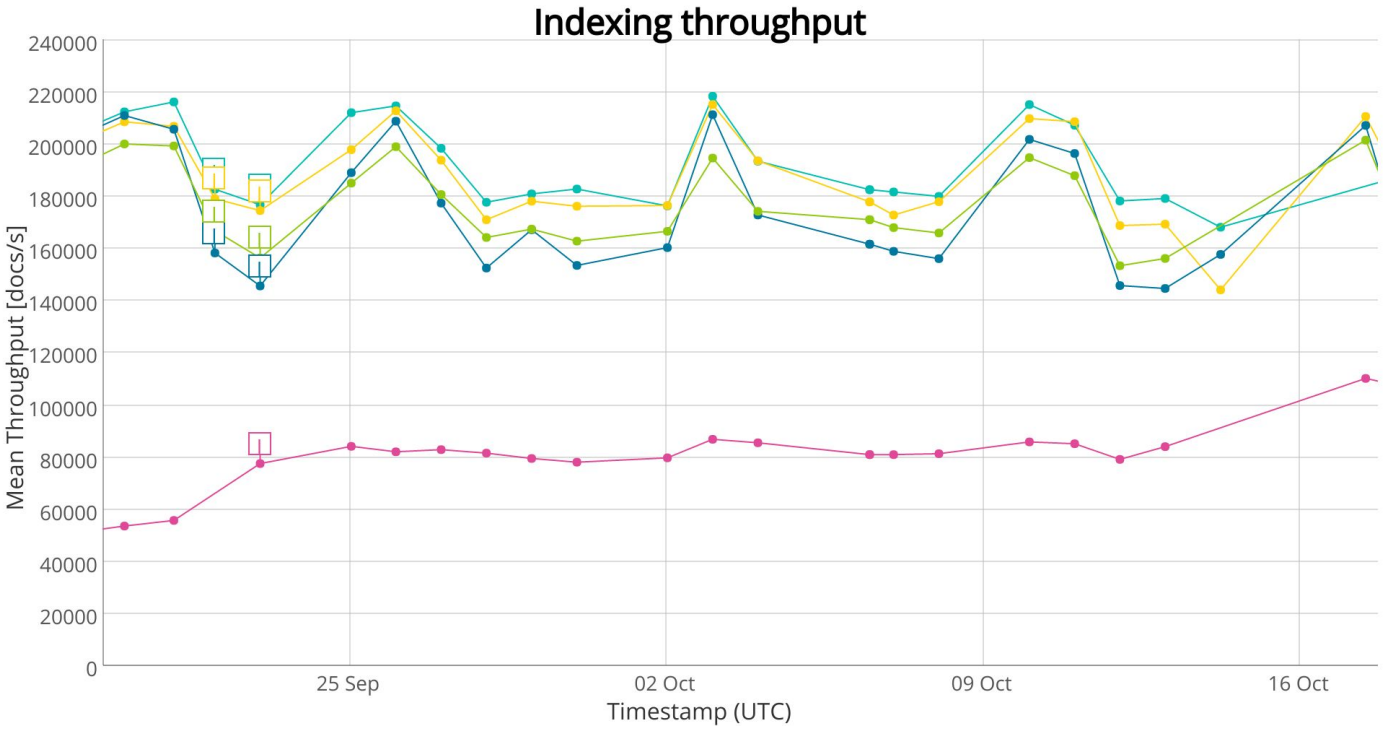
Reduce Noise

Better reproducible numbers

- Stable environment
- Turn off system daemons
- Load generator is on a separate machine
- Low-latency, high-throughput network
- No other traffic

Reduce Noise

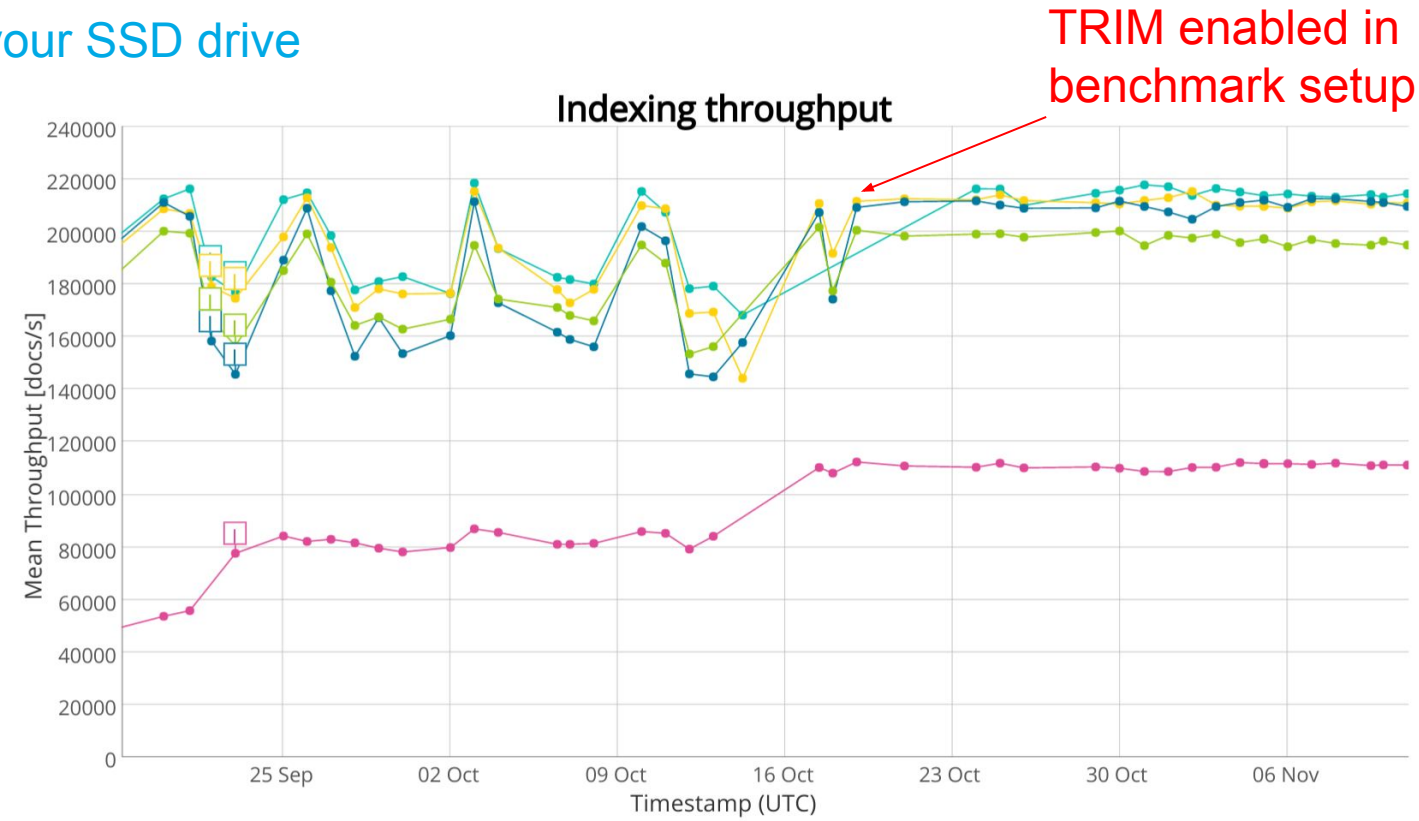
Weekly variation in throughput?



elastic

Reduce Noise

TRIM your SSD drive



elastic

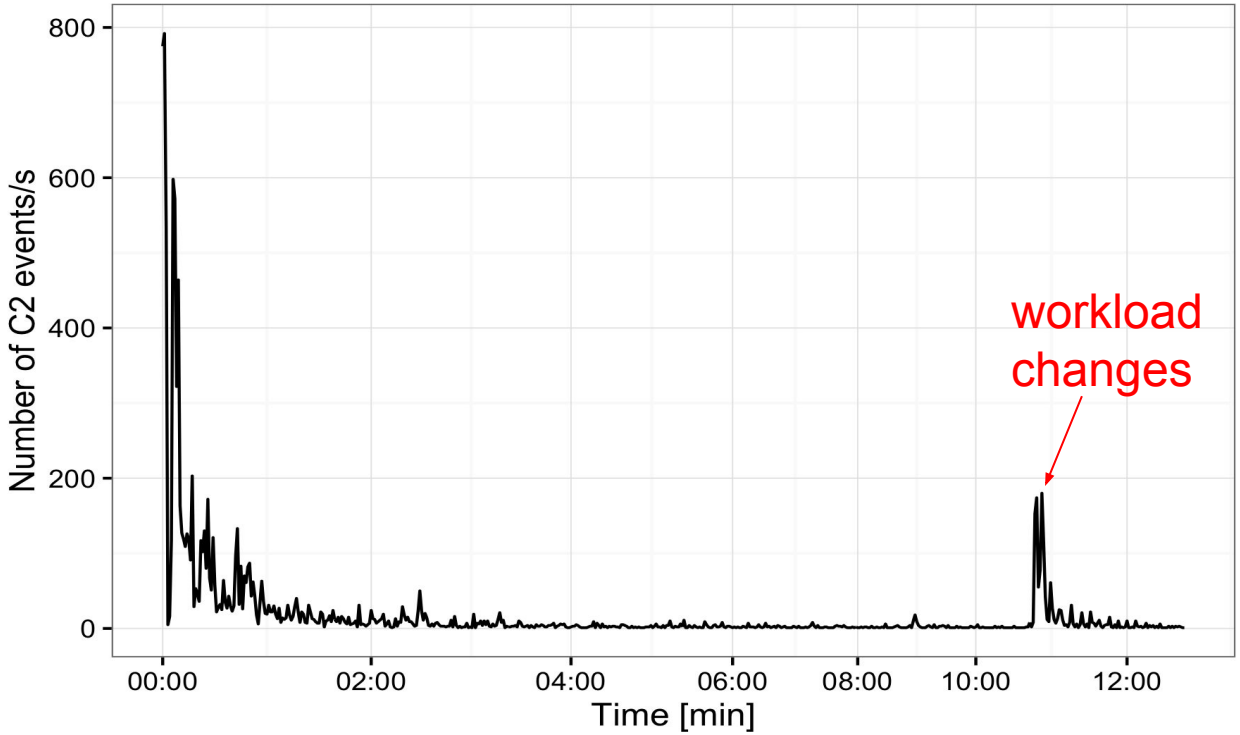
Rule Two

Warmup Properly

Are you awake before your first coffee?

Warmup Effects

JIT compilation



elastic

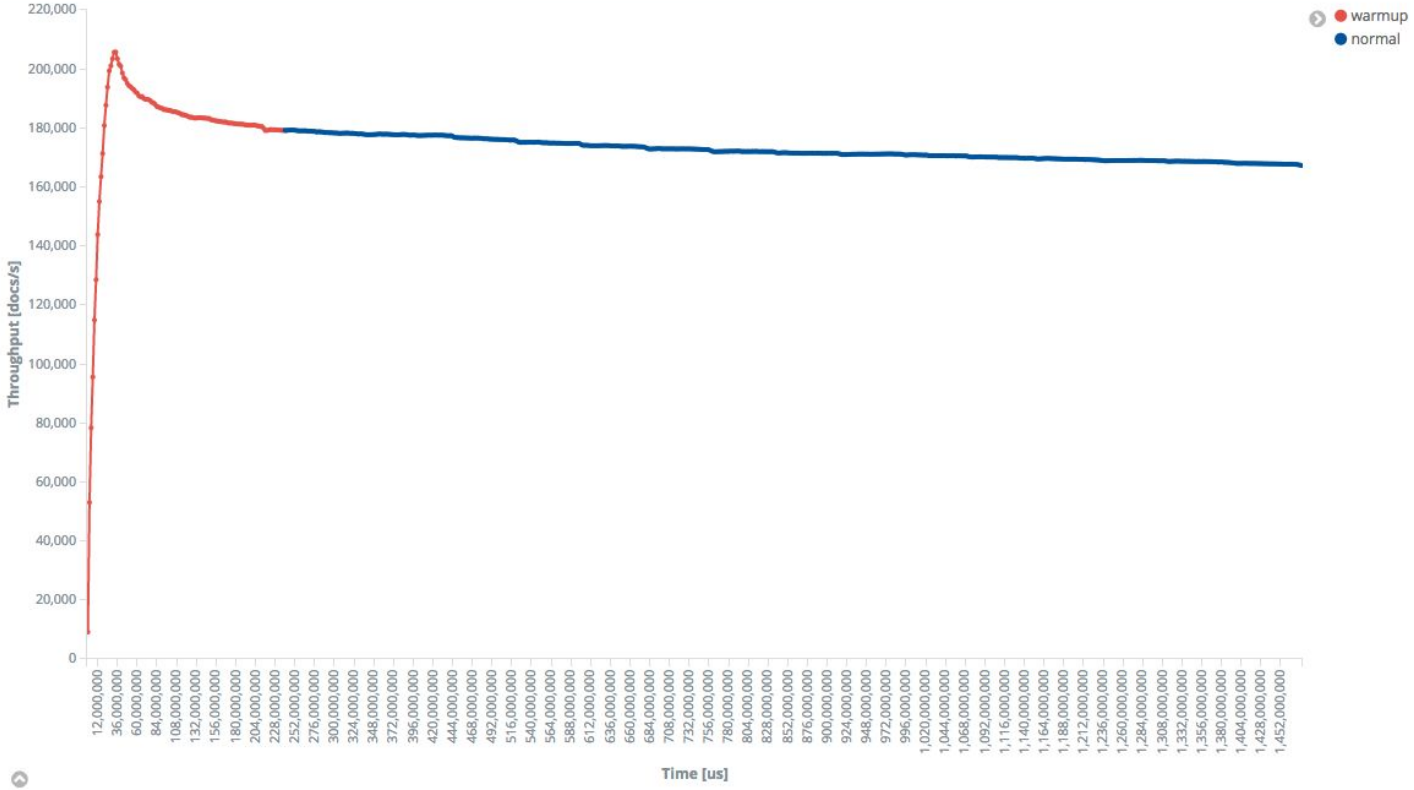
Caches Everywhere

Consider in Warmup and Workload Definition

- **CPU** L1 - L3 cache (incl. prefetching unit)
- **Disk**-internal cache (absorb I/O spikes)
- **OS** page cache (buffers writes to disk)
- **Application** caches: shard request cache, node query cache

Warmup Effects

Indexing Throughput



elastic

Rule Three

Model your Production Workload

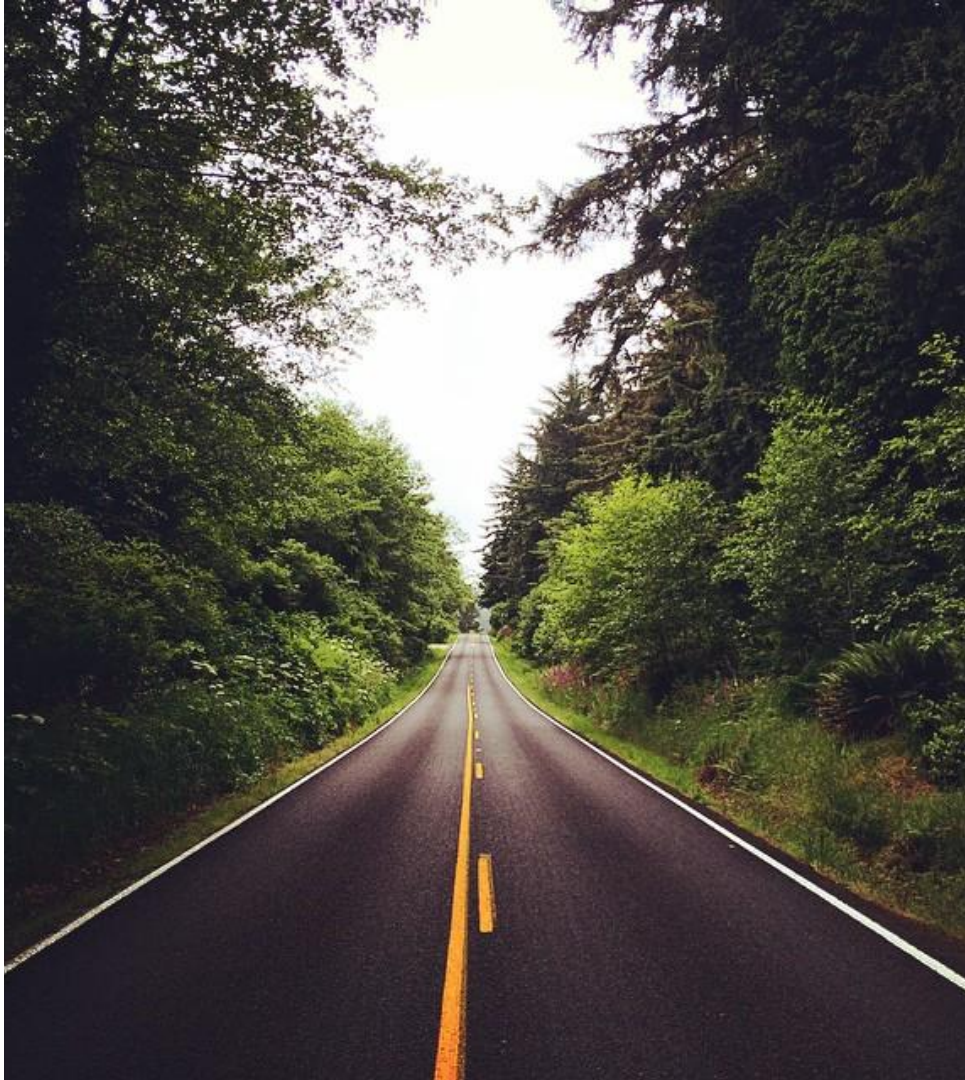
Waiting Time

Service Time

Latency = Waiting Time + Service Time

Utilisation

At 0%: No waiting time



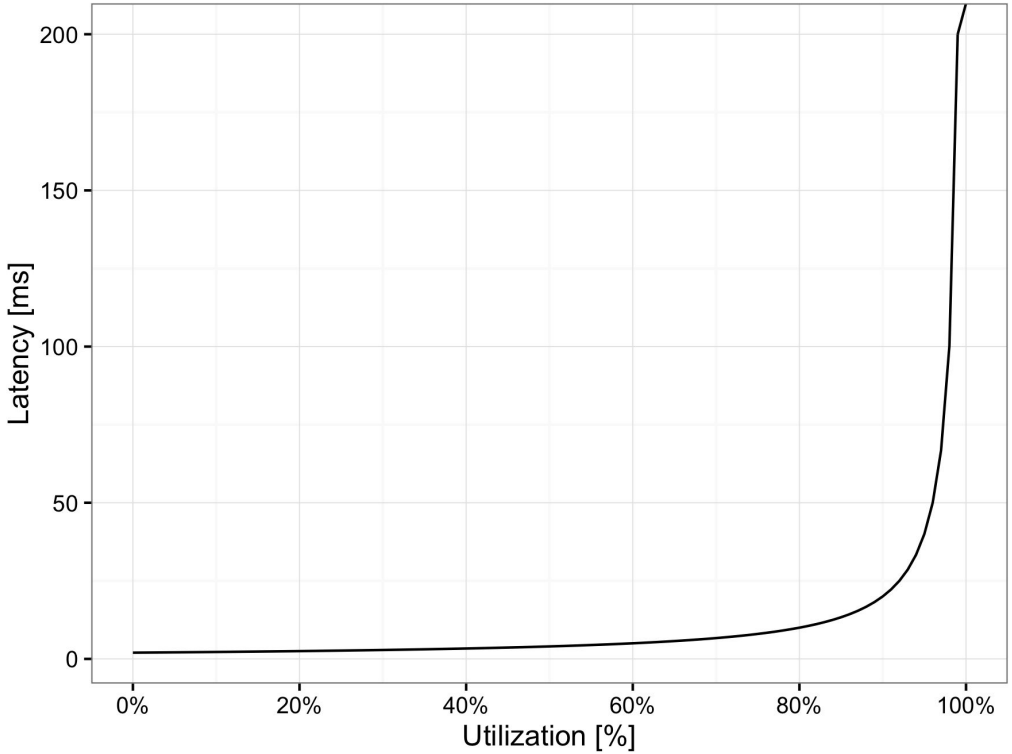
Utilisation

At 100%: High waiting time

Throughput and Utilisation

Latency...

… but at which throughput?



Created based on http://robharrop.github.io/maths/performance/2016/02/20/service-latency-and-utilisation.html

elastic

Tips

Batch Operations (e.g. bulk indexing)

- Important metrics: Throughput
- Run at maximum throughput
- Watch error rate (rejections, timeouts), reduce load if necessary

Tips

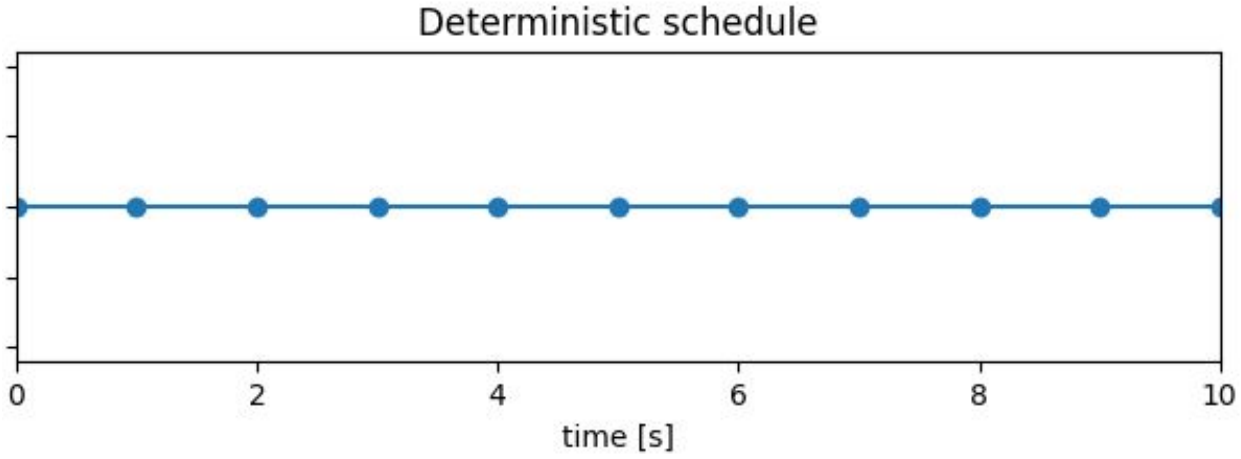
Interactive Operations (e.g. searches)

- Important metrics: Latency
- Run at a defined throughput (production metrics for guidance)
- Latency >> service time is a clear sign of saturation

Measuring Latency

Modelling Arrivals: Deterministic schedule at 1 query/s

- Simple, but often unrealistic (coordination between users)
- Latency spikes (requests pile up)

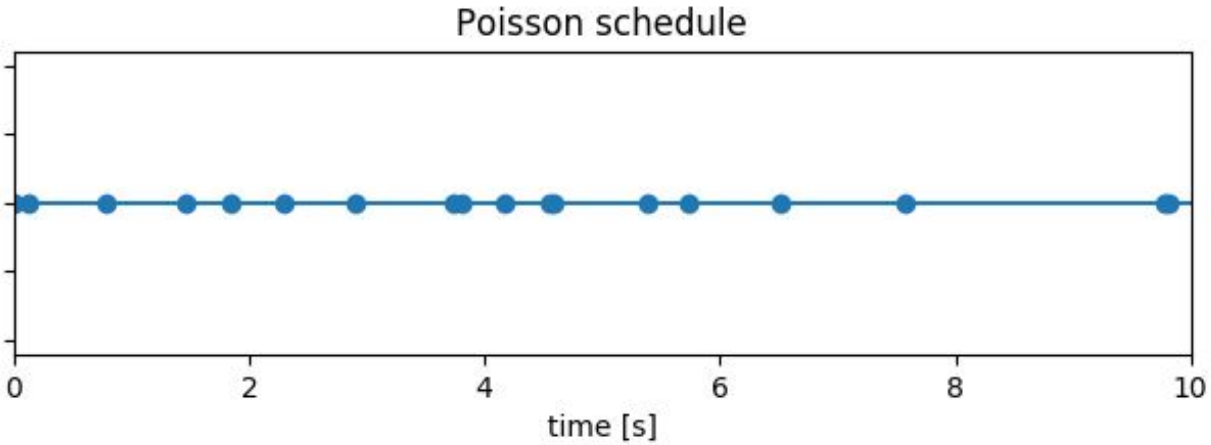


elastic

Measuring Latency

Modelling Arrivals: Poisson schedule at 1 query/s

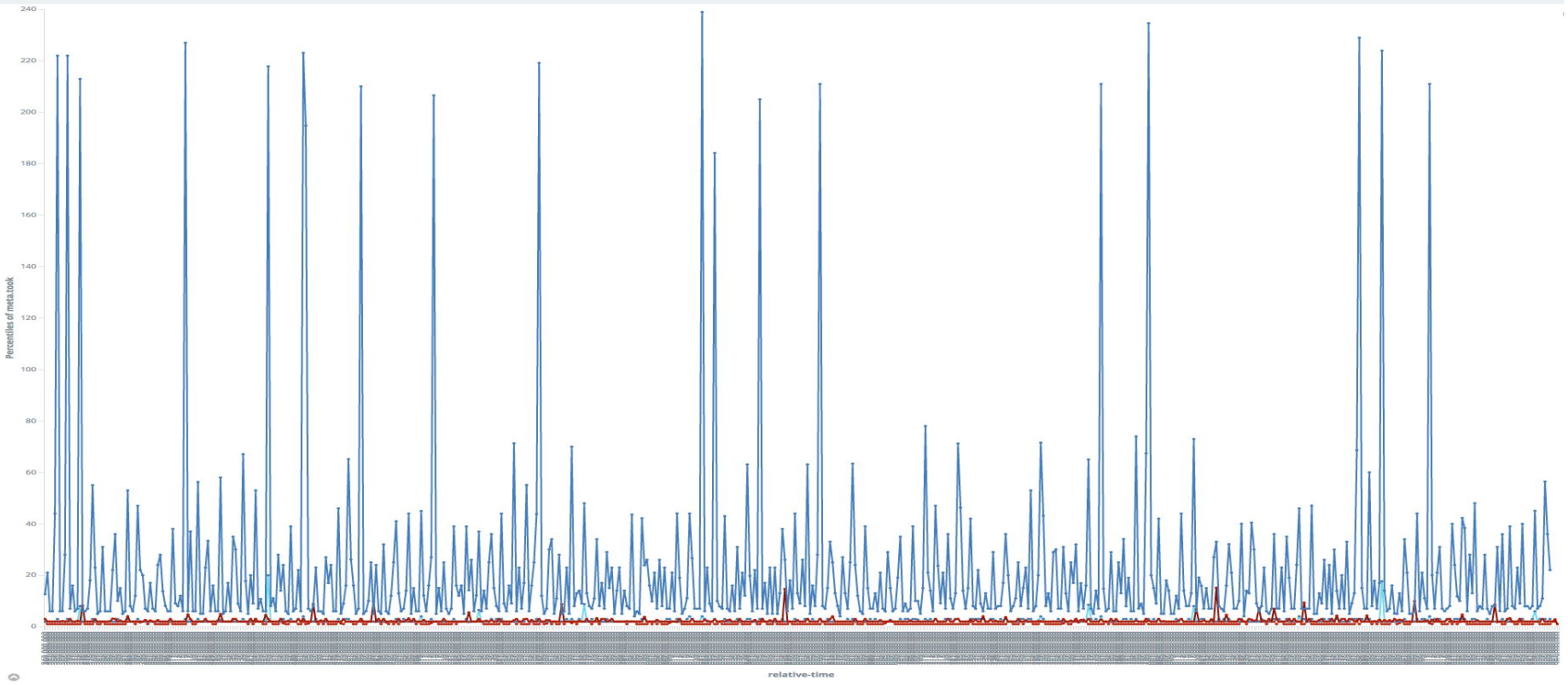
- Probabilistic: not intuitive at first
- Often more realistic (models **independent users**)



elastic

Measuring Latency

Deterministic (blue) vs. Poisson (red) with 300 concurrent clients



elastic

Rule Four

Test your Benchmarking Software

elastic

Newsflash: Benchmarking software has bugs

"It must be correct. After all, it produces numbers with 6 decimal places!"

- Response status code checks (the fast 404)?
- Maximum throughput of your load generator?

Example 1: Inappropriate Timeout

Overwhelming Elasticsearch

```
es = Elasticsearch(target_hosts)
while True:
    sendBulk(es)
```

Example 1: Inappropriate Timeout

Overwhelming Elasticsearch

```
# increase request timeout
es = Elasticsearch(target_hosts, timeout=60)
while True:
    sendBulk(es)
```

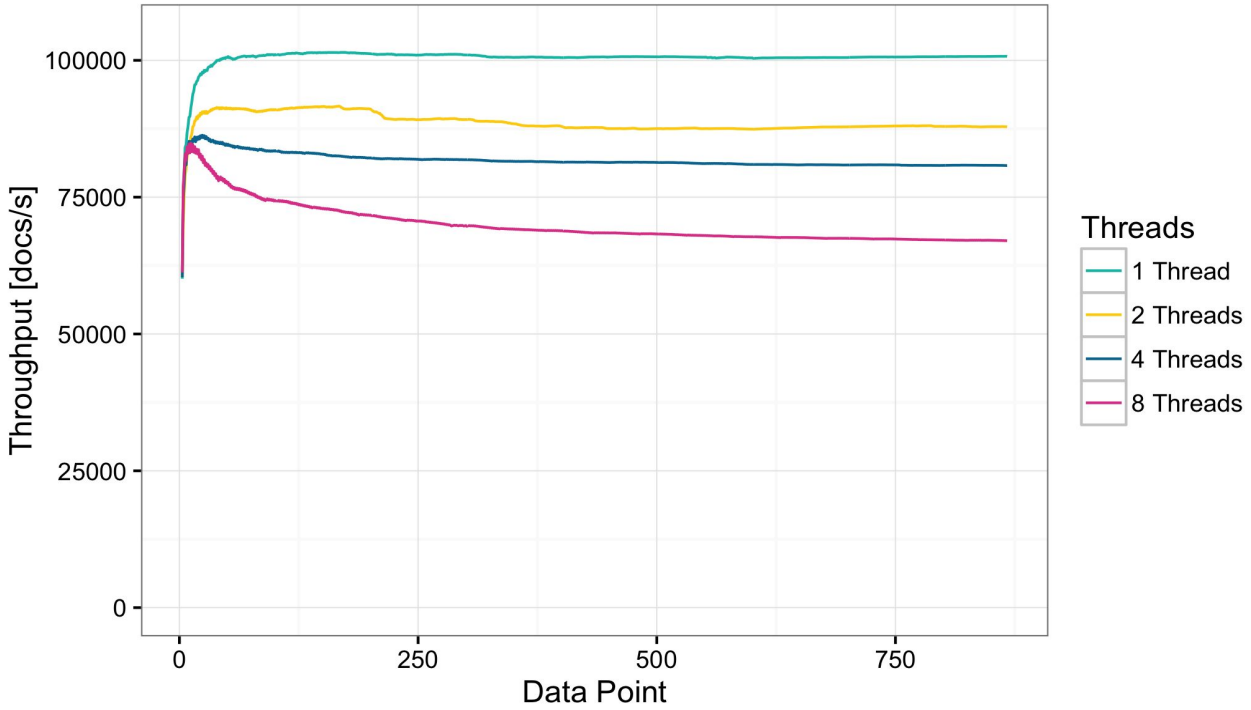
Example 2: Contention in Elasticsearch?

More clients, less load?

Client Count	Median Throughput [docs/s]
1	100.000
2	87.500
4	80.000
8	70.000

Example 2: Contention in the Load Generator!

More clients, less load?



elastic

Example 3: Let’s Query

```
while read -r query
do
    curl --data "${query}" "http://es:9200/cars/_search"&
done < popular_car_queries.txt
```

Be Critical

Check, check and then check again

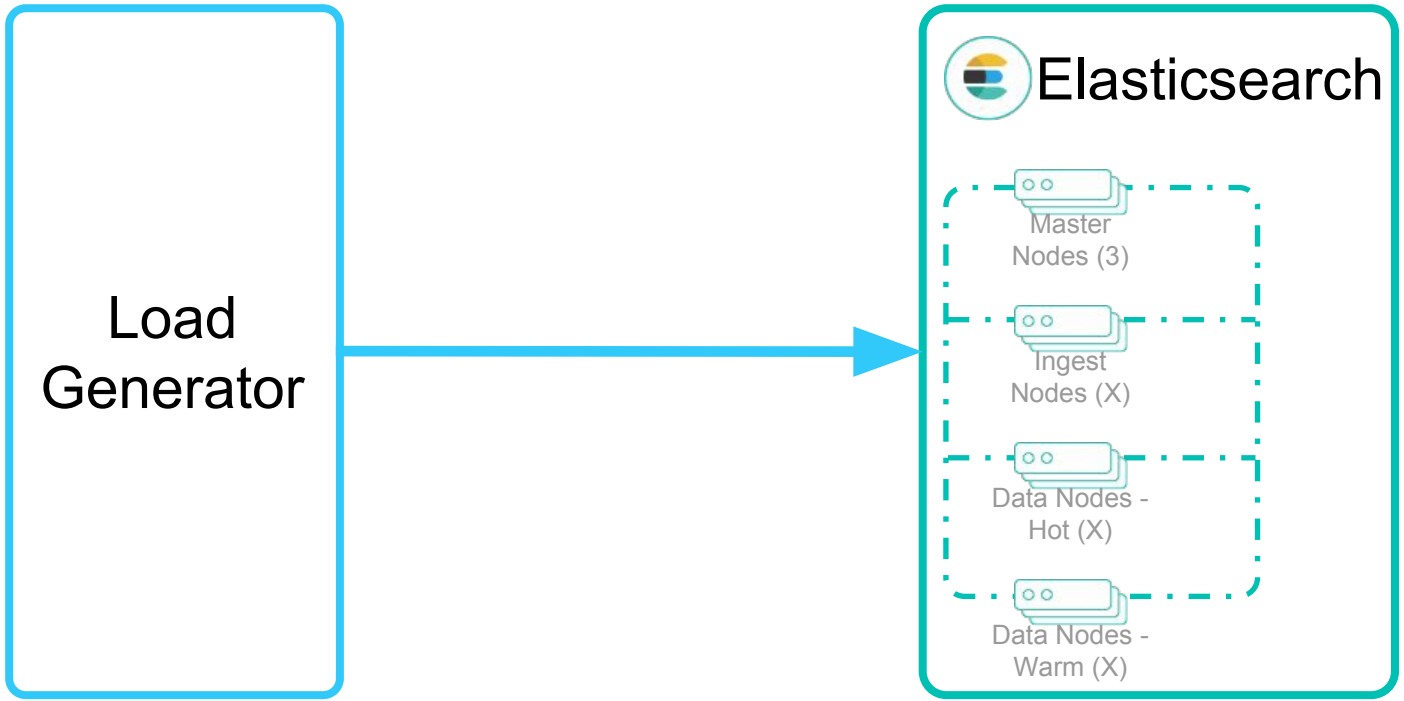
- Don’t trust any random script
- Stress-test your load generator
- Cross-check behavior on network level (Wireshark)
- Test error scenarios (e.g. 404s)

Rule Five

Eliminate Accidental Bottlenecks

Are you stressing the right component?

Check every subcomponent



Are you stressing the right component?

More nodes, no throughput gains?

Node Count	Median Throughput [docs/s]
1	1.300
2	2.600
3	2.600

Are you stressing the right component?

Example: Check network bandwidth with `ifstat`

```
Time              ens3
HH:MM:SS    KB/s in  KB/s out
10:07:12       0.11       0.21
10:07:13      34.71   45218.57
10:07:14     224.08   91764.32
10:07:15     821.85   127922.0
10:07:16    1612.70   127817.9
```

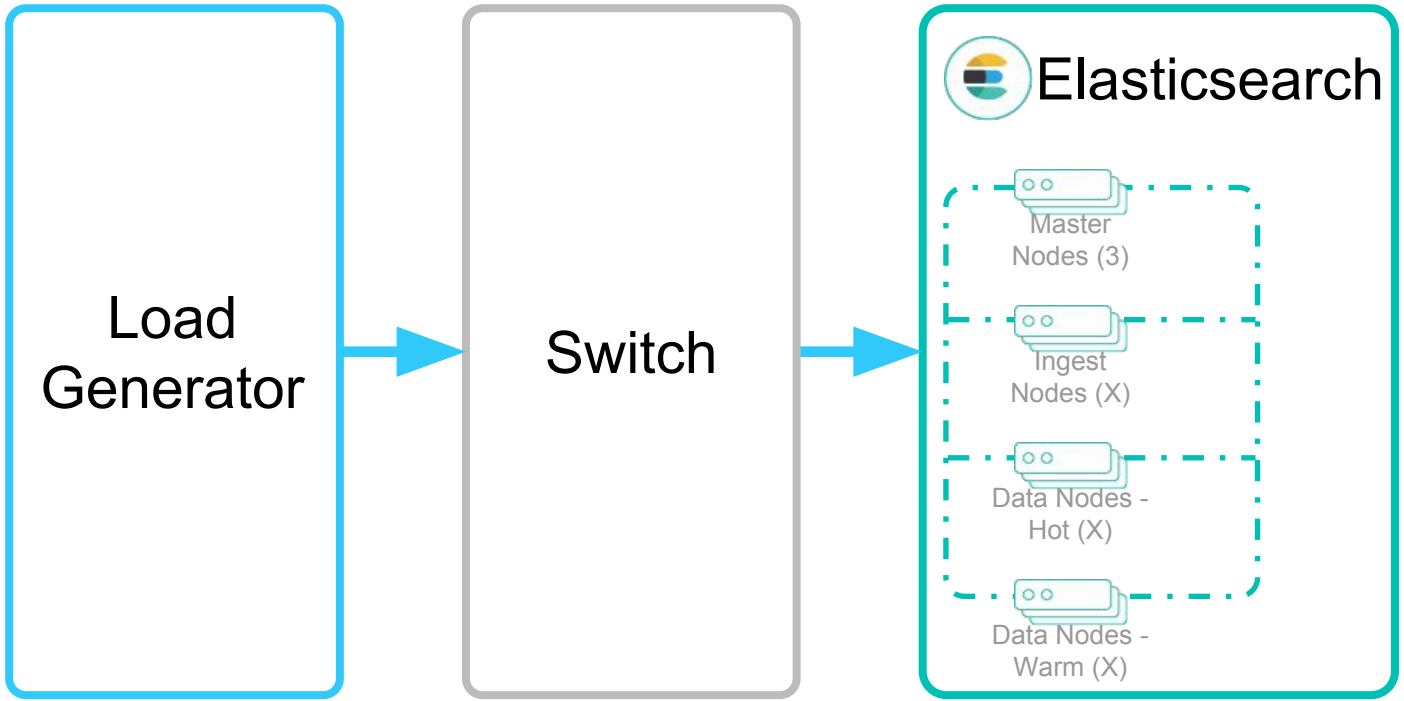
Are you stressing the right component?

Retry with a 10 Gbit card

```
  Time             ens3
HH:MM:SS    KB/s in   KB/s out
12:16:32       0.13       0.32
12:16:33      45.81   47114.57
12:16:34     354.18   96889.94
12:16:35     751.95  193469.0 # 1 Gbit link would be saturated
12:16:36    1722.80  271688.9
```

Are you stressing the right component?

Check **every** subcomponent



Are you stressing the right component?

Check methodically

- Example approach: USE method by Brendan Gregg (http://www.brendangregg.com/usemethod.html)
  - **U**tilization
  - **S**aturation
  - **E**rrors

Rule Six

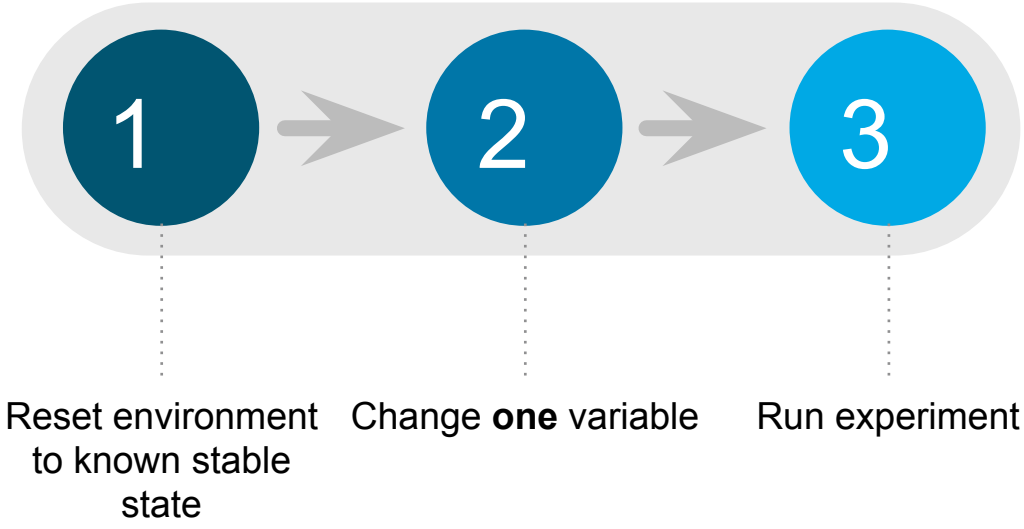
Use a structured process

I'll update Elasticsearch …
… and the Java version.

A recipe for disaster

One Step at a Time

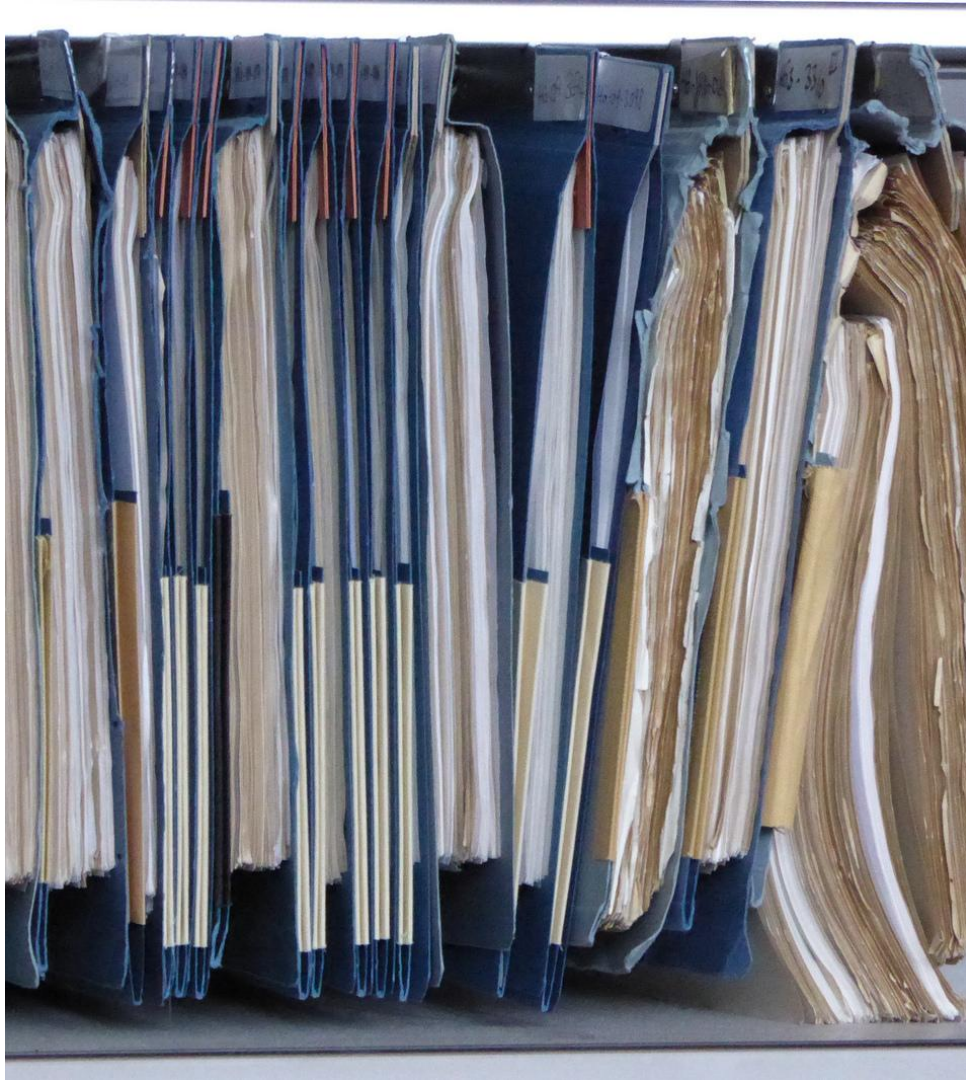
Benchmark Experiment Execution



elastic

What did I do to get these results?

Document Everything



Example metrics record

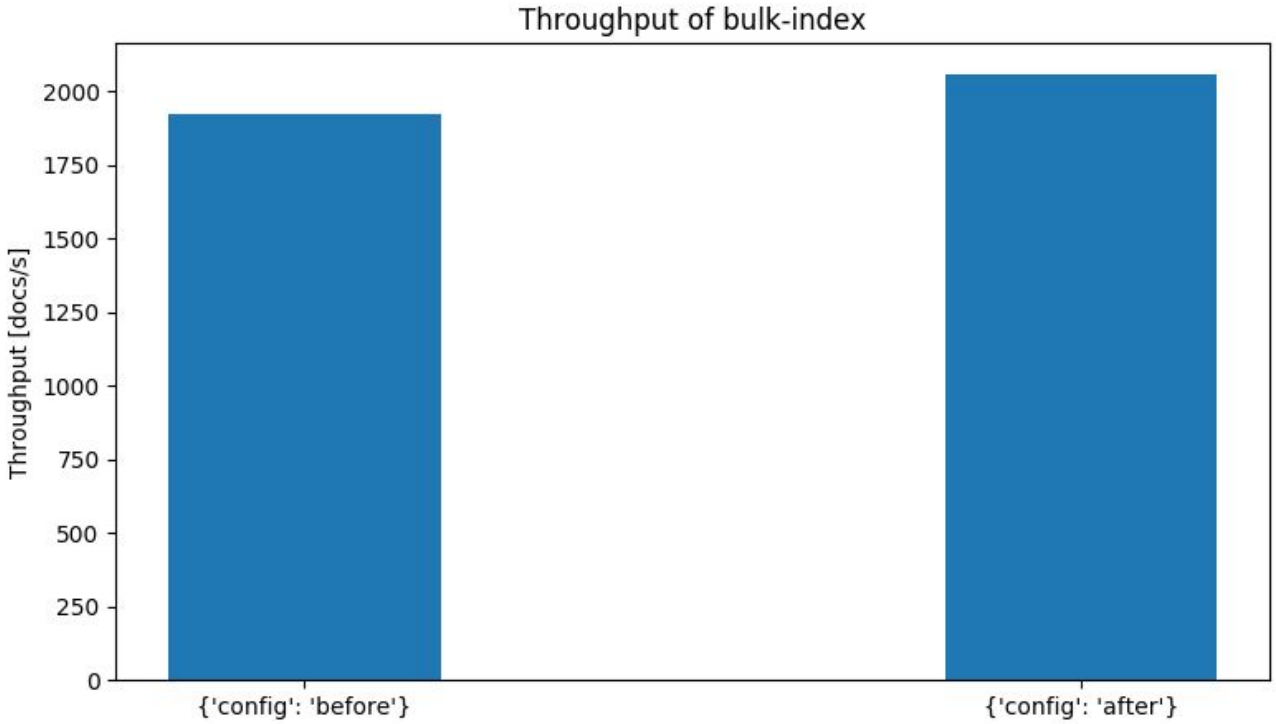

```
{
  "environment": "nightly",
  "trial-timestamp": "20180201T210054Z",
  "@timestamp": 1517544210265,
  "name": "cpu_utilization_1s",
  "value": 799.4,
  "unit": "%",
  "sample-type": "normal",
  "track": "nyc_taxis",
  "car": "4gheap",
  "meta": {
    "distribution_version": "7.0.0-alpha1",
    "source_revision": "df1c696",
    "node_name": "rally-node-0",
    "host_name": "192.168.14.3",
    "cpu_model": "Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz",
    "os_name": "Linux",
    "os_version": "4.10.0-42-generic",
    "jvm_vendor": "Oracle Corporation",
    "jvm_version": "1.8.0_131"
  }
}
```

Rule Seven

Use Statistical Significance Tests

Our Benchmark Results

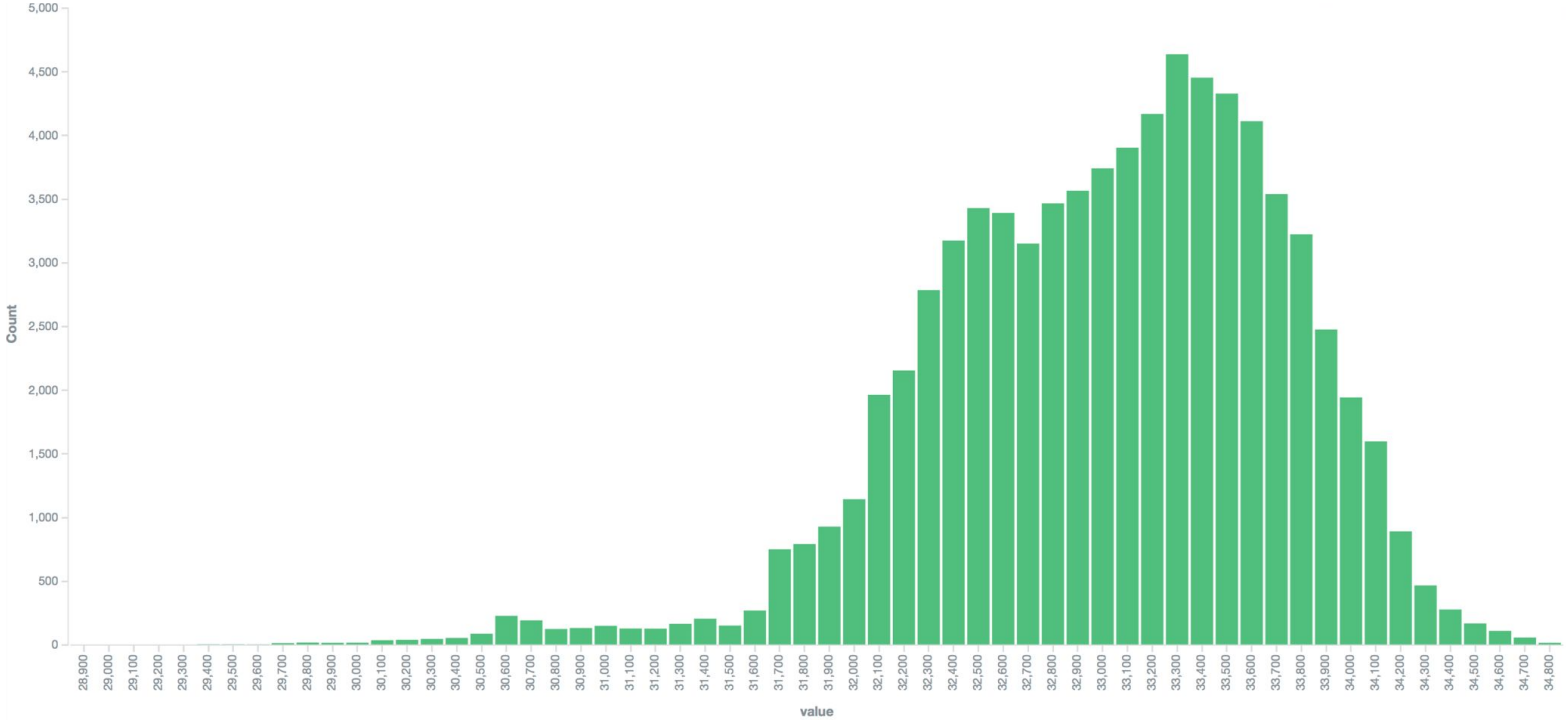
Are we done yet?



elastic

Example: Indexing Throughput Distribution

Lots of trial runs in identical conditions



elastic

Mitigating run-to-run variation

Statistical Significance Tests

- Control every variable that you can (see “reducing noise”)
- Run-to-run variation is a fact
- Multiple trial runs (> 30) and statistical significance tests (e.g. t-test)

Summarizing Results

General Tips

- Median, mean, mode: Median is robust against outliers
- Report also at least minimum and maximum

Summarizing Results

Latency

- No “characteristic” value: Use percentiles
- Don’t assume normal distribution: latency is multi-modal (fast path / slow path)

Summary and Outlook

Ben is happy

1. Benchmarks run in production-like environment
2. Warmup is considered
3. Workload modelled according to production
4. Load test driver checked
5. No accidental bottlenecks
6. Structured benchmarking process
7. Results are checked for statistical significance

How do we benchmark at Elastic?

- Macrobenchmarking tool Rally: https://github.com/elastic/rally
- Implements many best practices
- Everything is open source: Tooling and data
- Everything is public: system configuration and detailed results

Fall Seven Times,
Stand Up Eight.

Japanese Proverb

https://daniel.mitterdorfer.name/talks

Reference Material

Further Reading

- Rule 1: On issuing TRIM: https://www.elastic.co/blog/is-your-elasticsearch-trimmed
- Rule 3: “Relating Service Utilization to Latency” by Rob Harrop: http://robharrop.github.io/maths/performance/2016/02/20/service-latency-and-utilisation.html
- Rule 3: “The Queueing Knee” by Baron Schwartz: https://www.xaprb.com/blog/queueing-knee-tangent/
- Rule 5: USE Method by Brendan Gregg: http://www.brendangregg.com/usemethod.html
- Rule 7: Statistically Rigorous Java Performance Evaluation: https://dri.es/files/oopsla07-georges.pdf
- Rule 7: How not to measure latency by Gil Tene: https://www.youtube.com/watch?v=lJ8ydIuPFeU

Reference Material

Image Credits

- Upgrade by gato-gato-gato (license: CC BY-NC-ND 2.0)
- Oregon Dunes National Recreation Area by Theo Crazzolara (license: CC BY 2.0)
- Paperwork by Erich Ferdinand (license: CC BY 2.0)
- Coffee by Fil.Al (license: CC BY 2.0)
- I miss coffee by Daniel Go (license: CC BY-NC 2.0)
- It's about the coffee by Neil Moralee (license: CC BY-NC-ND 2.0)
- On an adventure by Dirk Dallas (license: CC BY-NC 2.0)
- Traffic Jam by lorenz.markus97 (license: CC BY 2.0)
- Swirl me back home by Nick Fisher (license: CC BY-ND 2.0)