Tips and Tricks to Scale Elasticsearch for 1M RPMs and Beyond

Vinted

>BLN
BZZ/
WRDS

$ whoami


```
{
  "name": "Dainius Jocas",
  "company": {
    "name": "Vinted",
    "mission": "Make second-hand the first choice worldwide"
  },
  "role": "Staff Engineer",
  "website": "https://www.jocas.lt",
  "twitter": "@dainius_jocas",
  "github": "dainiusjocas",
  "author_of_oss": ["lucene-grep"],
  "also_known_for": ["Bernese Mountain dogs"]
}
```

Agenda

1. Intro
2. Scale as of January, 2020
3. IDs as keywords
4. Filtering on dates
5. Feature focused indices
6. Scale as of January, 2021
7. Discussion

Intro: Vinted and Elasticsearch

- Vinted is a second-hand clothes marketplace
- Operates in 10+ countries, 10+ languages
- Elasticsearch is in use since 2014
- Elasticsearch 1.4.1
- Today I'll share tips and tricks learned while scaling Elasticsearch at Vinted

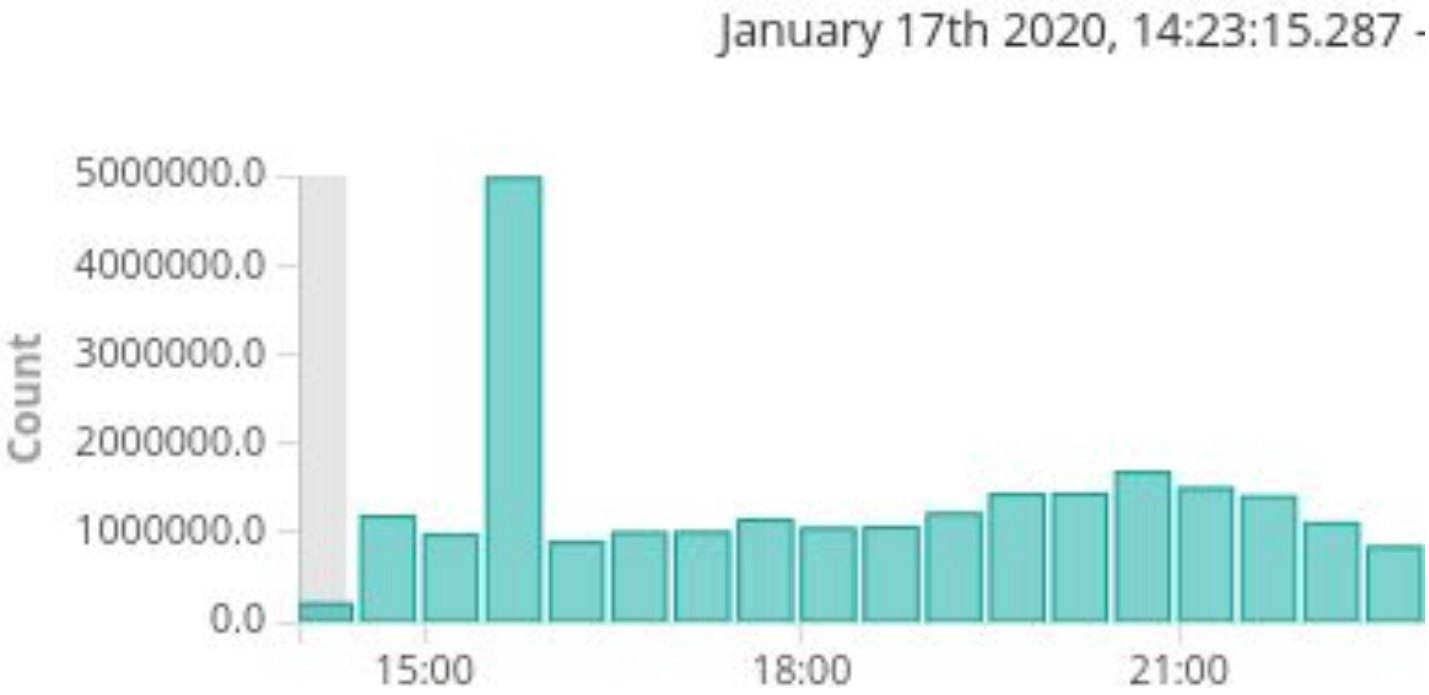
Elasticsearch Scale @Vinted as of January 2020 (1)

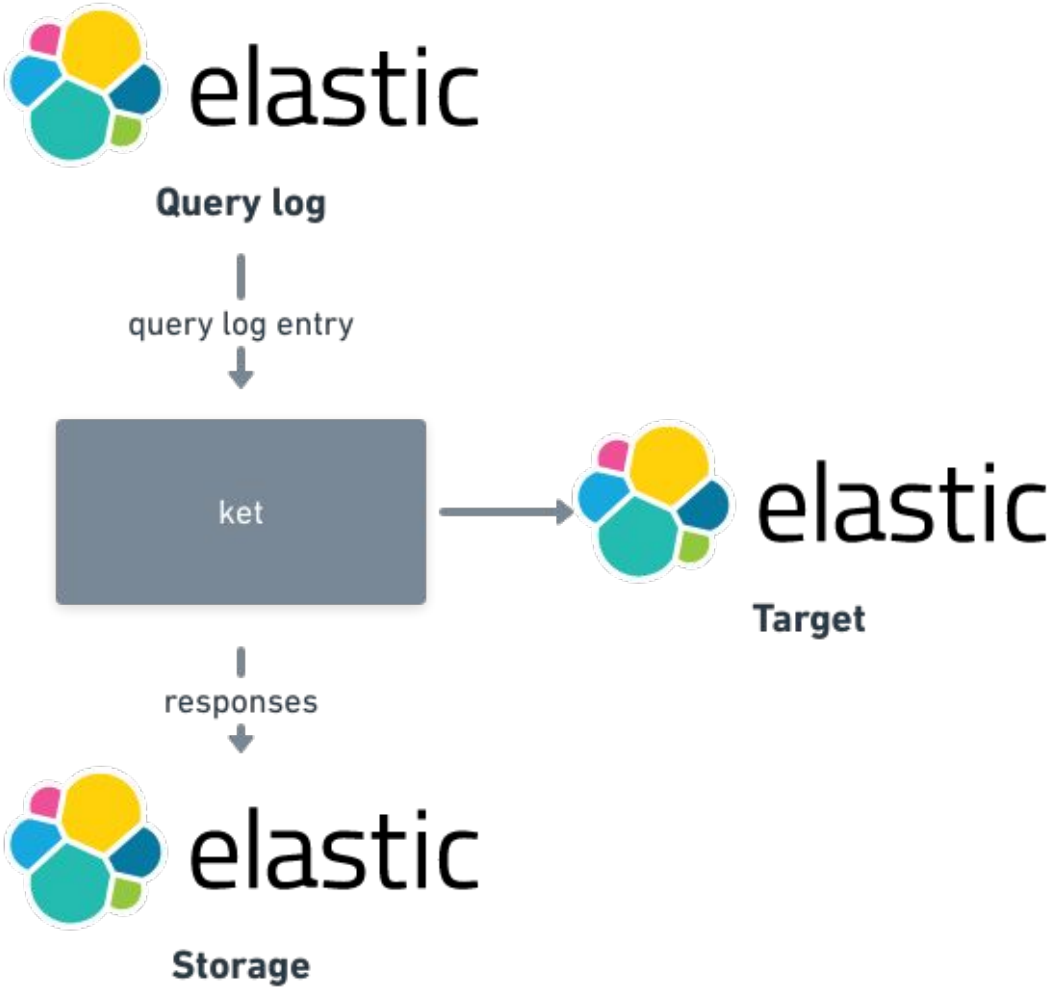
- Elasticsearch v5.8.6 (~end-of-life at that time)
- Upgrade to v6.3.x failed with 2-day downtime back in 2018
- 1 cluster of ~420 data nodes (each 14 CPU with 64GB RAM, bare metal)
- ~300K requests per minute (RPM) during peak hours
- ~160M documents
- 84 shards with 4 replicas
- p99 latency during peak hours was ~250 ms
- Slow Log (i.e latency >100ms) queries skyrockets during peak hours

Elasticsearch Scale @Vinted as of January 2020 (2)

- The company saw Elasticsearch as a business risk(!)
- Back-end developers didn’t want to touch it
- The usage of the Vinted platform was expected to at least double by October (similar increase in usage of Elasticsearch and more servers is a bad idea)
- Functionality on top of Elasticsearch just accumulated over the years, no oversight, no clear ownership
- SRE were on the Elasticsearch duty (hint: server restart doesn’t help all that much when the Elasticsearch is overloaded)

Replayed 10k queries during the “easy Tuesday”




elastic
Query log
query log entry
ket
elastic
Target
responses
elastic
Storage

IDs as keywords

Vinted

IDs as keywords (1)

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "terms": {
           "country_id": [2,4]
          }
        }
      ]
    }
  }
}
```

IDs as keywords (2): context

- Elasticsearch indexed data from MySQL (check vinted.engineering blog on details for that)
- Common practice for Ruby on Rails apps is to create database tables with primary keys as auto-increment **integers**

- **Q:** Which Elasticsearch data type to use?
- **A:** integer, because why not?

IDs as keywords (3)

IDs as keywords (4): TL;DR of the blog post

- Before: integers were indexed as padded **string terms**
- After: integers indexed as **block k-d tree** (BKD)
- Change in Lucene get into Elasticsearch since 5.0
- Numeric data types were optimized for **range queries**
- Numeric data types **continued to support** `terms` queries

IDs as keywords (5): from Vinted point of view

- On IDs we don’t do range queries
- We use IDs for simple filters with **terms** query
- The “optimized” integer data type for our use case degraded query latency
- How much?
- For our workload it was a ~15% instant decrease in p99 latency, **~20ms**
- We use around 10 such fields for filtering in **every search query**
- The required change was as simple as **changing the index mappings** and **reindexing the data**
- No query changes required

IDs as keywords (6): summary

- Remember that Vinted uses Elasticsearch since pre 5.0
- At that time it was OK to index IDs as Elasticsearch integers
- Post 5.0 IDs as integers became a performance issue
- Such a change that brakes nothing can easily slip under the regular developers radar and then could backfire badly
- Outcome is that regular developers think that Elasticsearch performs badly
- Highly recommend to try this optimization on your data

Filtering on Dates

Vinted

Date Math (1)

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "created_at": {
              "gte": "now-7d"
            }
          }
        }
      ]
    }
  }
}
```

Date Math (2)

- From the developer POV, it is simple, you just hardcode `now-7d`
- Note that most queries that use **now** (see Date Math) **cannot be cached**
- If most of your queries are using Date Math then handling **more queries** requires **more CPU**, because no cache hits
- Cached queries clauses -> massive gains in the cluster throughput
- My advice: always use **explicit timestamps** in production (in Kibana the date math is OK)

Date filters (1)

```
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "created_at": {
              "lte": "2021-06-16T09:55:00Z"
            }
          }
        }
      ]
    }
  }
}
```

Date filters (2)

- This query clause asks Elasticsearch to “collect docs that are not newer than X”
- What if the X is meant to be **now** and your documents are accumulated over the last 10 year?
- Then this filter matches ~99% of all docs in your indices
- Not a good “filter”

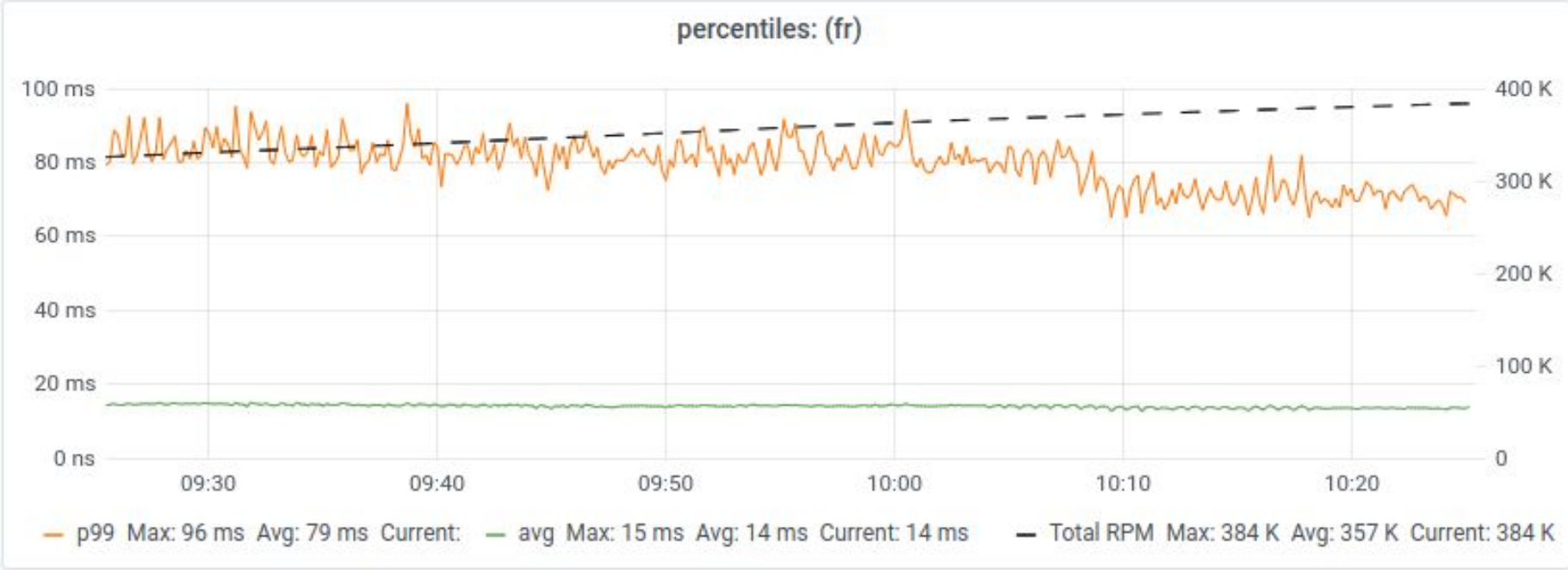
Date filters (3)

```
{
  "query": {
    "bool": {
      "must_not": [
        {
          "range": {
            "created_at": {
              "gt": "2021-06-16T09:55:00Z"
            }
          }
        }
      ]
    }
  }
}
```

Date filters (4)

- This query clause asks Elasticsearch to “collect docs that are not newer than X”
- What if the X is **now** and your documents are accumulated over 10 year?
- Then this filter matches ~1% of all docs in your indices
- A good “filter”, i.e. more specific filter is a good filter
- Bonus tip from docs: “if you must filter by timestamp, use a coarse granularity (e.g. round timestamp to 5 minutes) so the query value changes infrequently”

- For our setup the inversion of the filter **reduced** the p99 latency by ~15%, **~10ms**


percentiles: (fr)
100 ms
80 ms
60 ms
40 ms
20 ms
0 ns
400 K
300 K
200 K
100 K
0
09:30
09:40
09:50
10:00
10:10
10:20
p99 Max: 96 ms Avg: 79 ms Current:
avg Max: 15 ms Avg: 14 ms Current: 14 ms
Total RPM Max: 384 K Avg: 357 K Current: 384 K

Filtering on Dates: summary

- Don’t use Date Math in production
- Write filter clauses on timestamp (or any other data type) in a way that it matches fewer documents

Feature focused indices

Vinted

Feature focused indices (1): context

- Large collection of documents
- Stored in one index (many shards)
- Multiple types of queries are coming to that index, e.g. search and aggs
- When the search traffic increases the p99 latency grows
- Considered splitting the workload into many clusters but decided not to do it because of operational complexity and Elasticsearch cluster should be, well, elastic

Feature focused indices (2): idea

- Same one huge cluster
- Index the same data multiple times
- A different index name with same mappings just to begin experimentation
- Route the traffic to the new index
- Optimize later
- “Divide and conquer” strategy

Feature focused indices: example

- We have a query that searches for **newest items grouped by favourite brands** that were **uploaded over the last week**
- Brands are favorited by the users
- Elasticsearch, `top_hits` aggregation
- The data shape is the same as in the main catalog search
- Only “recent” data is really needed

Feature focused indices: request cache POV

- Request cache is a shard-level **request** cache that caches the local results **on each shard**.
- Useful for frequently used search requests for aggregations
- Request cache hit rate for the specialized index: **0.426**
- Request cache hit rate for the catch-all index: **0.061**
- The two different workloads are not interfering with each other

Feature focused indices: summary

- Given that your Elasticsearch cluster has **enough capacity**
- Split query traffic by use-case and then optimize it

- 👍 Easy to prioritize: by # queries
- 👍 Simpler to measure optimizations
- 👎 When a common attribute changes **all** indices have to be reindexed

Congrats! We are done with lessons

Vinted

Elasticsearch Scale @Vinted as of June 2021 (1)

- ES 7.9.3
- 3 clusters each ~160 data nodes (each 16 CPU with 48GB RAM, bare metal)
- One offline cluster of similar size for testing (upgrades, cluster setup, etc.)
- ~1000K RPM during peak hours
- ~360M documents
- p99 latency during peaks ~150 ms
- Timeouts (>500ms) are 0.0367% of all queries

Elasticsearch Scale @Vinted as of January 2021 (2)

- Team (8 people strong) is responsible for Elasticsearch related features
- Regular capacity testing for 2x load in terms of
  - document count
  - query throughput
- Elasticsearch is seen as the system that can accommodate the growth
- Functionality is tested performance wise before releasing to production
- The team members rotate on duty
  - Keeping clusters operational
  - Maintenance tasks from backlog

Discussion (1)

- Is everything perfect? No.
- Elasticsearch is resource hungry
- Version upgrades still has to be checked before releasing
  - Offline testing cluster helps with that
- Machine Learning engineers insist that Elasticsearch is not up for their tasks
  - Despite the fact that the search ranking data is used for their model training
  - Search re-ranking is done outside of Elasticsearch (e.g. operational complexity)
- Elasticsearch default installation offers very few tools for search relevance work

Discussion (2)

Thank You!