Performance optimizations for e-commerce search

with Apache Solr

Tomasz Sobczak, MICES 2017

FINDWISE

About me

Work at FINDWISE
SEARCH DRIVEN SOLUTIONS

Lucene, Solr and Elasticsearch

Everything related to search tech & business

Focus on search relevancy

Enterprise Search Warsaw Meetup

@sobczakt

Introduction

Document 1

Powerful you have become, the Dark Side I sense in you.

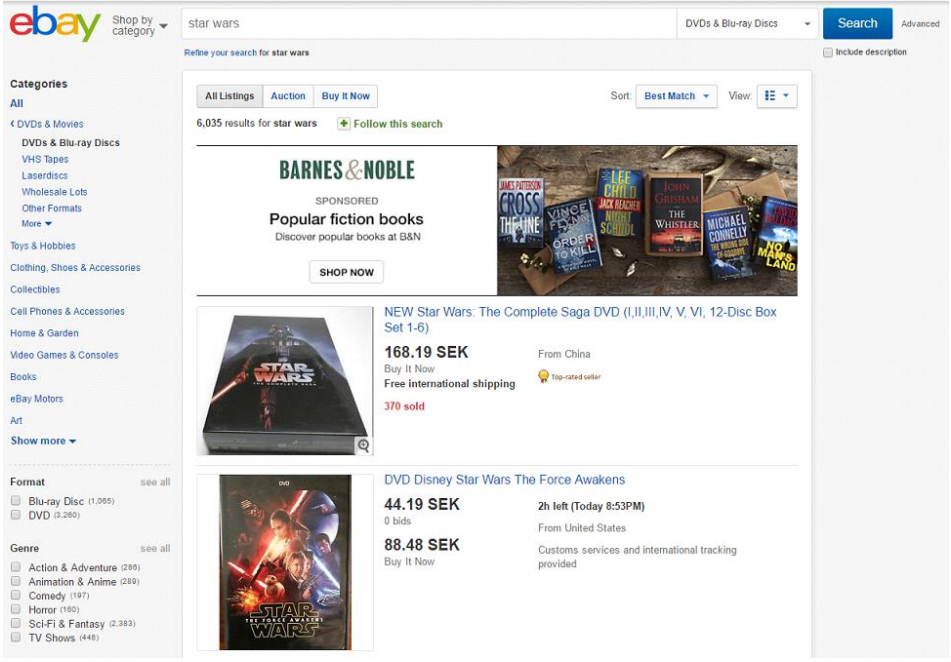
Document 2

You don't know the Power of the Dark Side.

ID	Term	Doc ID
1	become	1
2	dark	1,2
3	don't	2
4	have	1
5	I	1
6	in	1
7	know	2
8	of	2
9	power	2
10	powerful	1
11	sense	1
12	side	1,2
13	the	1,2
14	you	1,2

FINDWISE

Introduction



FINDWISE

Typical challenges

Size of the cluster

Hardware

How many shards

How many replicas

Scaling strategy

Index design

Data archiving

Searching / indexing performance

Optimizing queries

Relevancy in big assets

Speed of reindexing all data

Monitoring & managing

JVM, caches etc.

FINDWISE

Immutability

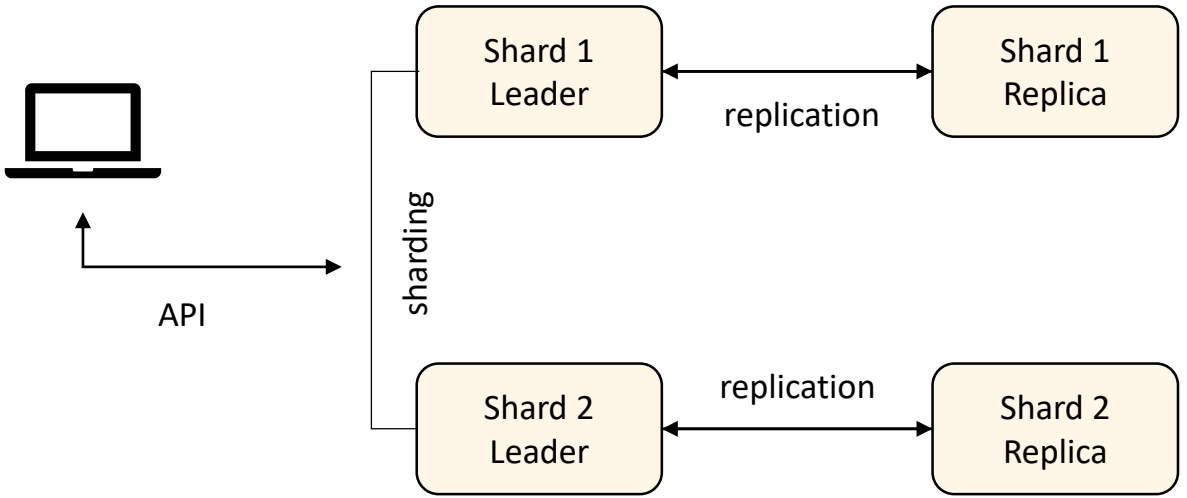
Inverted index written to disk is immutable

- ✓ no need for locking – no worries about updating and parallel processes
- ✓ index stays in the kernel's filesystem cache, because it never changes
- ✓ data doesn't change – caches stay valid for the life of index
- ✓ single inverted index allows for compression, reduce I/O operations and amount of RAM memory

- ⛔ New data, rebuild entire index
- ⛔ Updating or deleting is impossible, so firstly mark as deleted then remove from the results and clean up when time comes

FINDWISE

General architecture



FINDWISE

Shards and replicas

Shard your data

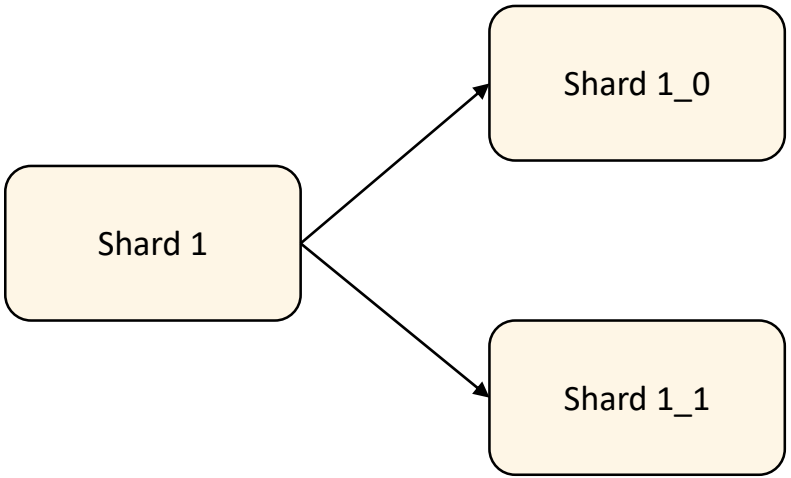
- Splits your content volume horizontally
- Increases performance / throughput because of distributed, parallelized operations

Add replicas

- Provides high availability when node fails
- Scales search volume / throughput because of parallel searches on all replicas

FINDWISE

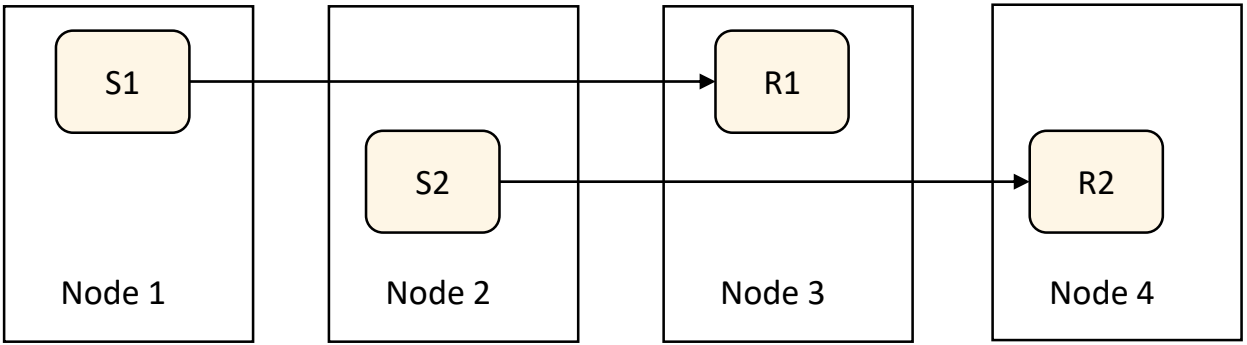
Splitting shards



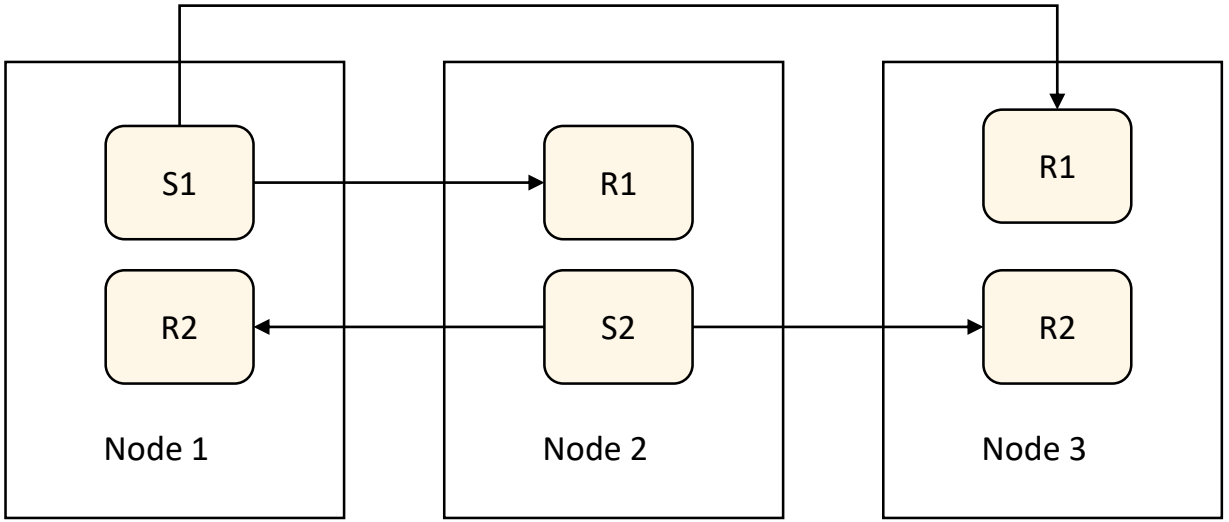
FINDWISE

Replicate your data

- When a shard learder has a downtime, replica takes new role
- Replicas increase search performance, but only when you add more hardware



Replicate your data



How many shards?

- No precise answer
  - What kind of hardware? How do your documents look like? How are you going to use them? How to analyze? Aggregations?
  - Start with X shards and test if you can have less without hurting performance
- Shard is a full right Lucene, so it costs resources
- Every query must look into every shard in the index
  - Fine, but things start to get complicated when shards compete for the same node's resources
- Small data assets in many shards can hurt relevance
  - Not necessary (*distributed IDF*), but can

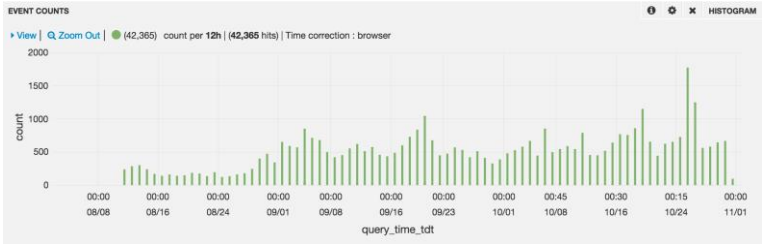
FINDWISE

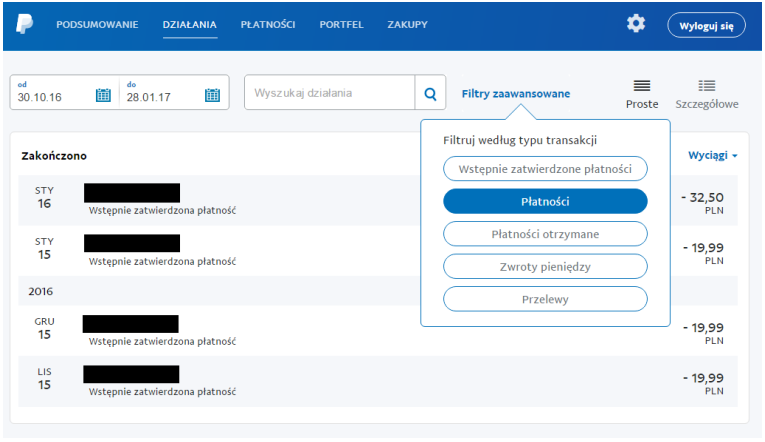
How many shards?

1. Start single server node with target hardware
2. Create collection / index with target data model but only one shard and no replicas
3. Index as much documents as you can to approach production state
4. Run your queries and simulate real traffic
5. Try to reach the limit when your single node cluster won't meet expectations
6. With the result for single shard, estimate your target multishard and repliacted enviroment

FINDWISE

Designing your cluster





FINDWISE

Design: per user

- Index per user
- When your users search only own data
- Can be not very effective if users have small data assets
  - In fact filters are fast
  - Lucene internals can be better used when less number of indexes
  - Remember about clusterstate
- Separeted index for user who own much more data than average

FINDWISE

Design: routing

- Multi-tenancy and co-location
- Most of the time you work with defult routing based on doc's ID
  - Data is partitioned quite equally
  - Query needs to look into all shards
- You can specify routing parameter and direct documents into the same shard
  - Then need to rememebr about this parameter in query time
  - Something between single big data asset and indexes per user

Design: time-stamped data

- An endless stream of logs
- You need to remove old data (or archived) to not run out of the space
- Delete, even bulk is inefficient (remember, immutable)
- Create collections / indexes per time frame
  - Yearly
  - Monthly
  - Daily
- Close / delete / move unused data sets

FINDWISE

Design: hot & cold

- Approach to archive data easily and efficiently
- Hot nodes
  - Better hardware
  - Heavy indexing and searching
  - No optimization
- Cold nodes
  - No indexing, rare queries
  - Optimize index

FINDWISE

Assign replicas based on rules

- Don't assign more than 1 replica of this collection to a host
- Assign all replicas to nodes with more than 100GB of free disk space or, assign replicas where disk space is more
- Do not assign any replica on a given host because I want to run an overseer there
- Assign replica in nodes hosting less than 5 cores or assign replicas in nodes hosting least number of cores

- https://cwiki.apache.org/confluence/display/solr/Rule-based+Replica+Placement

- Rule = shard + replica + tag (attribute of a node like freedisk or rack)
- Example:
  shard:shard1,replica:*,rack:730
- Rules are specified per collection during creating collection (REST API)

FINDWISE

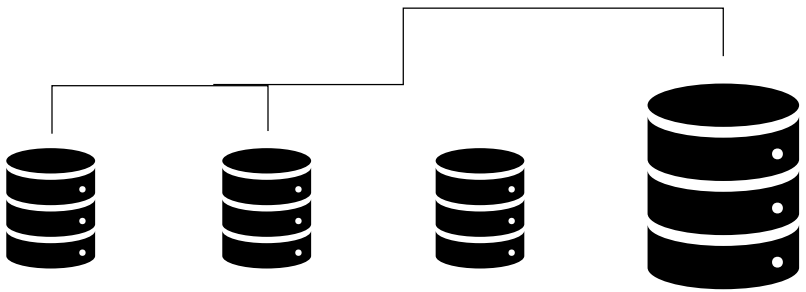
Your own commit policy

1. New documents indexed = added to the buffer and transaction log
2. Docs from memory buffer go to the new segment
3. New segment is searchable (it's opened)
4. Buffer is cleared (transaction log not, it collects docs)
5. Full commit makes 2 - 4 and creates new tlog, data is persisted

FINDWISE

Merging policy

- Number of segments is a trade off between search and indexing performance
  - Too many segments – worse for searching
  - Too few segments – too much work for merge process
- Segments are merged in the background, it doesn't affect NRT search
- Small segments are merged into bigger ones (and so on) in accordance to some policy
  - Couple similar (size) segments are selected and merged into a bigger
- Don't optimize (to single segment) your live, hot collections!

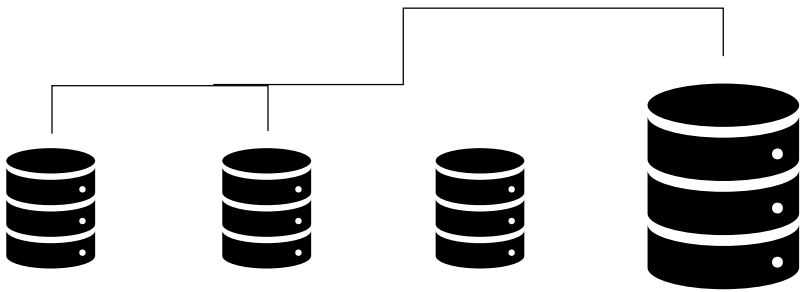


FINDWISE

Merging policy

- EarlyTerminatingSortingCollector

```
<mergePolicyFactory class="org.apache.solr.index.SortingMergePolicyFactory">
  <str name="sort">timestamp desc</str>
  <str name="wrapped.prefix">inner</str>
  <str name="inner.class">org.apache.solr.index.TieredMergePolicyFactory</str>
  <int name="inner.maxMergeAtOnce">10</int>
  <int name="inner.segmentsPerTier">10</int>
</mergePolicyFactory>
```



FINDWISE

More performance

- Performance testing
  - start with single node, no shards / replicas
  - start with default settings and target data / queryes - as far as possible
  - run tests for a long time, at least 30 minutes
- Batch indexing
  - find your right packet size
  - try to start with 5 – 15 MB per batch
  - then start increasing concurrency of your batch operations
- The more throughput your disks can handle, the more stable your cluster will be

FINDWISE

Doc values

- Use DocValues for sorting & faceting
- Column-oriented fields with a document-to-value mapping

```
{
        'document 1': {
                'field1':A,
                'field2':B
        },
        'document 2': {
                'field1':C,
                'field2':D
        }
}
```

```
{
        'field1':{
                'document 1':A,
                'document 2':C
        },
        'field2':{
                'document 1':B,
                'document 2':D
        }
}
```


FINDWISE

External File Field

- Values from an external file instead of the index
- Not searchable, can be used for function queries or display
- Example: boost most visited pages in search result. Statistics are changing daily and you don't want to re-index all pages every day
  - doc33=1.414
  - doc34=3.14159
  - doc40=42

Filtering

1. &fq= is your friend for faster queries
2. No score calculations for filter queries
3. Conceptually, non-scoring queries are executed *before* the scoring queries. Non-scoring queries reduce the number of documents and then run (costly) scoring.
4. Don’t cache unique filter queries for better caching (cache=false)
5. Control order of not cached filter queries with costs

FINDWISE

Caches

- filterCache, queryResultCache, documentCache
  - And others
- Generally: just cache… but sometimes it’s better to not cache ;-)
- Monitor stats like evictions, hitratio, warmup
- Understand cache invalidation and warming up
- *useFilterForSortedQuery* allows to use filterCache if request contains sorting and doesn’t have score. Filter will be used to get document ids and then sorting will be applied

How much RAM memory?

- It’s a huge topic!
- Avoid long GC
- Remember OS also needs memory
- Don’t set heap too large
- Max 50% of RAM for Solr
- No more than 32 GB as a heap
- Keep an eye on available and used heap space, caches size and stats

FINDWISE

Performance optimizations for e-commerce search

with Apache Solr

Tomasz Sobczak, MICES 2017

FINDWISE