Distributed search

of heterogeneous collections

with Apache Solr

lucid IMAGINATION

Andrzej Białecki
ab@lucidimagination.com

About the speaker

- Started using Lucene in 2003 (1.2-dev…)
- Created Luke – the Lucene Index Toolbox
- Apache Nutch, Hadoop, Solr committer, Lucene PMC member
- Apache Nutch PMC Chair
- Developer at Lucid Imagination

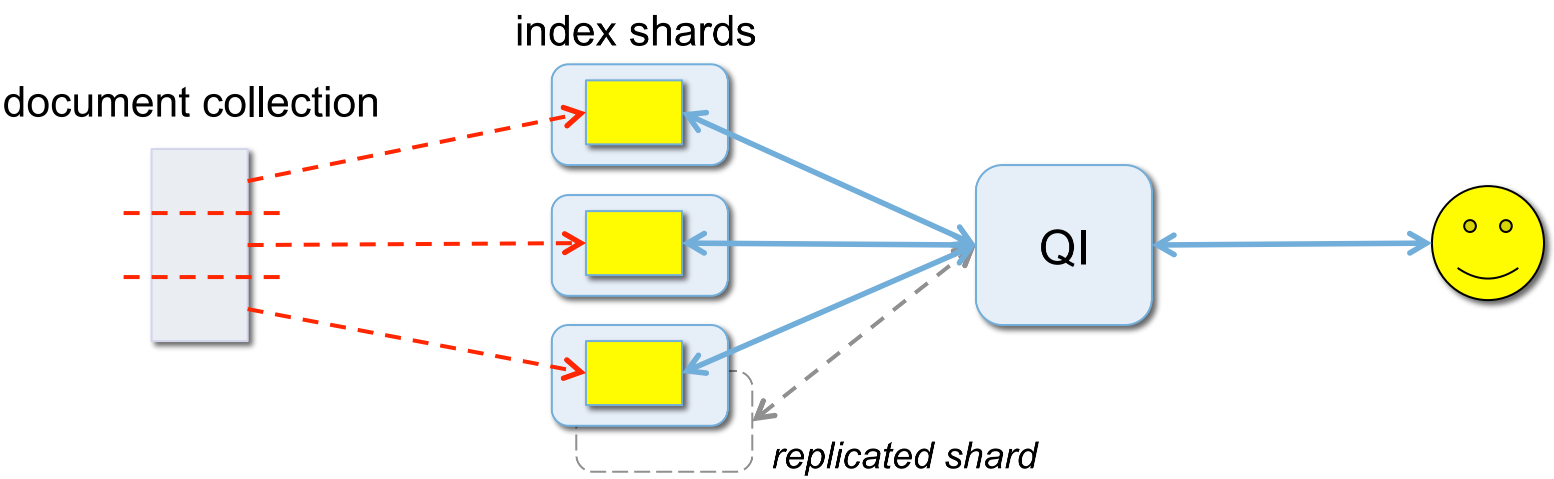
Agenda

- Distributed search concepts
- Apache Solr distributed search
- Experiments
- Challenges: analysis and solutions

Distributed search concepts

Overview

- Distributed versus “local” (non-distributed)
- Document collection is split among search servers
- Non-overlapping index parts are called “shards”
    - Distributed search with overlaps more complex!
- Query Integrator
    - Dispatches queries to shard servers
    - Merges partial result lists to return the whole result
- Distribution != replication



Why and when to distribute?

- Local search is obviously much simpler!
  - Use local search as long as you can
  - Optimize the index structure and memory use
- Distribute only when you hit the index size limits
  - Index too large -> RAM limits -> OS RAM paging
  - Cost of paging can be substantial and random
  - Worst case scenario: swapping 
    - Cost of swapping >> cost of paging
- Distribute when the cost of local search limitations outweighs the cost of distributed search

Costs of distributing

- Increased maintenance & ops cost
- Increased complexity – lower resilience to failures
  - But partial failures are usually not catastrophic
- Increased latency
  - *Dispatch* + *search* + *merge* time versus just *search*
- Many other “interesting” issues

Other reasons to distribute

- Heterogeneous collection
  - Distinct parts with different update regimes
- Outsourced collection parts
  - Parts are maintained by third-parties
  - “Federated” search
- Security aspects
  - E.g. per-user indexes, but global search needed too
  - Adversarial IR – prevent attackers from obtaining global index metrics

How to distribute

- Distribution by document
    - Distribution by term rarely used – complex queries difficult to execute
- Distributed indexing
    - Document is submitted to a front-end server
    - Front-end assigns a shard number
        - Round-robin, hash(id), consistent hashing, etc …
    - Document is sent to a shard server for indexing
- Distributed search
    - Query is submitted to a front-end server
    - Query is passed to all shard servers in parallel
    - Partial result lists are merged at the front-end
        - …with the assumption that scores and rankings are comparable across the partial lists!

Apache Solr distributed search

Distributed indexing & search in Solr

- Built-in distributed search across predefined shards, out of the box
- No built-in mechanism to manage shard servers
  - “Cluster awareness” in SolrCloud via Zookeeper
  - But shard management left to the admin
- No distributed indexing out of the box
  - But easy to implement via UpdateProcessor chain
  - Partitioning schema needed
    - Simple: by hash(docId), fixed number of shards
    - Less simple: consistent hash of docId, flexible number of shards
    - Custom: e.g. by document creation time

Distributed search in Solr

- SearchHandler handles the request dispatch
  - 3.x: “shards” parameter defines the targets
  - 4.x: shard sets can be managed in Zookeeper
- Any search node can perform as a Query Integrator
  - The cost of dispatch & merge can be load-balanced
- QueryComponent handles the search and the merging of query results

- Example request:

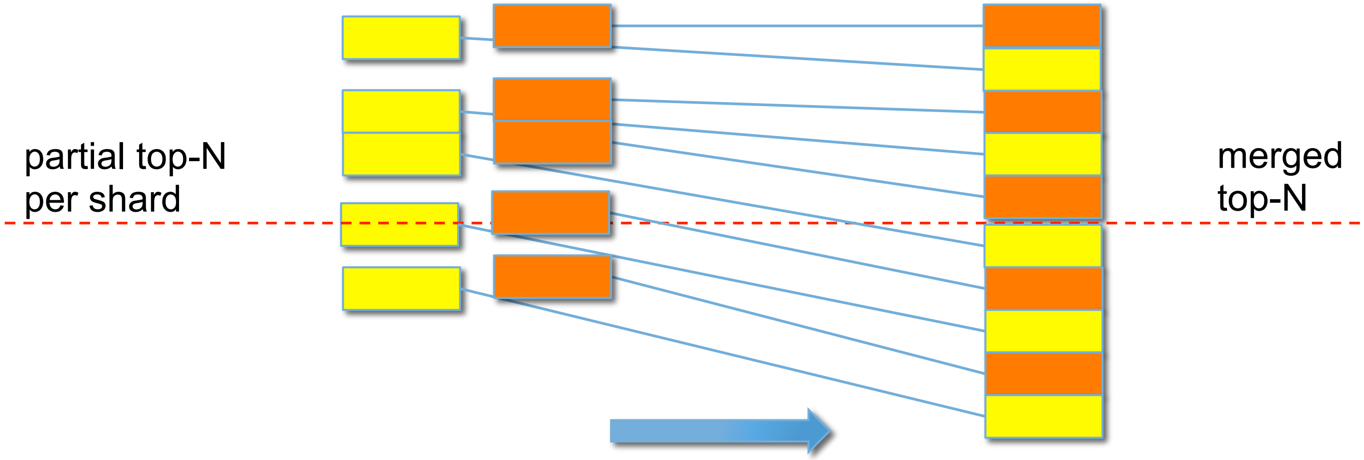
```
http://hostOne:8983/solr/shard1/select?q=test&shards=
hostOne:8983/solr/shard1,hostTwo:8983/solr/shard2
```

SearchHandler: dispatch

- Parallel dispatch to all live shard servers
  - 3.x: dead servers will cause exceptions
  - 4.x: dead servers are detected and avoided
- Wait for asynchronous retrieval of shard responses
  - Communication errors will cause exception -> 0 results
  - No graceful fallback to partial results (yet?)

QueryComponent: merge

- Two-phase process:
  - Retrieve & merge document id-s
  - Retrieve document fields for a merged list of id-s
- Simple duplicate removal (by id)
- Priority queue sorted by multiple sort criteria
  - Queue size: [0, 1, …, start + rows]



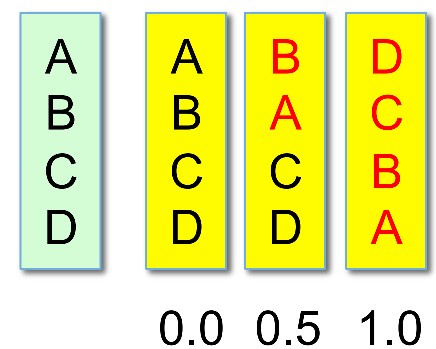
Experiments

Test corpus

- OHSUMED collection
  - 55000 medical abstracts
  - 5000 queries and relevance judgments
- Query types
  - MeSH – usually short, abstract concepts
  - OHSU – usually long, descriptive
  - Default OR operator favors recall over precision
- Obtained via Apache Open Relevance Project
  - Prepared to work with the Lucene TREC benchmark

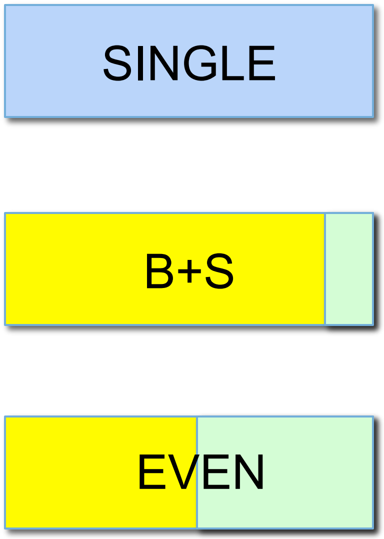
Metrics of quality – Spearman's footrule

- Precision, recall, mean reciprocal rank
  - Classic metrics, implemented in Lucene benchmark
  - Poorly reflect differences in ranking order
- Web-like search strongly favors top-N (N=10 ? or N=3 !)
- Kendall's tau and Spearman's footrule
  - Measures of disagreement in ranking
  - Spearman's footrule ≈ tau, but easier to compute
  - Spearman's ρ – quadratic distance metric
- Normalized <0..1>, 1 – total disagreement
- Spearman @ N – considers only top-N results
- Weighted Spearman footrule @ N
  - Highly positioned disagreements cost more



Corpus setup – three cases

- SINGLE – full corpus in a single index
  - Baseline for quality metrics and rankings
- B+S – split corpus into big and small part
  - Using a low-frequency term
- EVEN – split corpus into roughly even parts
  - Using a medium-frequency term



- Each shard (or SINGLE) as a Solr core
- Tests use either all queries, or MeSH, or OHSU

Test runs: baseline precision/recall

- OR-type queries -> high recall, medium precision
- On average Top-40 considered, 5000 queries

```
SUMMARY
  Search Seconds:          0.041
  Average Precision:       0.765
  Recall:                  0.992
```

- Not bad at all!
- What about the Top-10 ?

Test runs compared (Top-N=10)

SINGLE	Precision	Recall	Time [ms]
All	0.335	0.364	4
MeSH	0.338	0.367	4
OHSU	0.050	0.113	9

EVEN	Precision	Recall	Time [ms]
All	0.333	0.363	11
MeSH	0.337	0.366	10
OHSU	0.045	0.102	14

B+S	Precision	Recall	Time [ms]
All	0.334	0.364	10
MeSH	0.338	0.367	10
OHSU	0.050	0.110	14

Spearman's footrule tests

- **Expectation**: rankings from SINGLE should be close to the rankings from EVEN or B+S
  - EVEN should produce better results than B+S
- Baseline results from SINGLE define the ordering
- The same queries are run on EVEN and B+S using Solr distributed search
- Top-10 and Top-3 are then compared pairwise
  - Un-weighted (all positions equally important)
  - Weighted : (10 9 7) (4 3 2) (1 1 1) (2)
    - Top-3 most important, 1st is the winner
    - Next 3 sometimes checked
    - Bottom result often checked

Spearman's footrule test results (%)

Average deviation in rankings between result lists across query sets

SINGLE / B+S	SF @ 10	SFW @ 10	SF @ 3	SFW @ 3
All (4967)	3.96	3.58	3.57	3.64
MeSH (4904)	3.92	3.55	3.55	3.61
OHSU (63)	7.16	6.22	5.73	5.87
OHSU 1-term (4)	2.75	2.93	8.33	8.33

SINGLE / EVEN	SF @ 10	SFW @ 10	SF @ 3	SFW @ 3
All	9.21	8.07	7.42	7.55
MeSH	8.98	7.88	7.29	7.41
OHSU	26.57	22.75	17.90	18.40
OHSU 1-term	13.99	14.12	19.44	19.87

Challenges: analysis and solutions

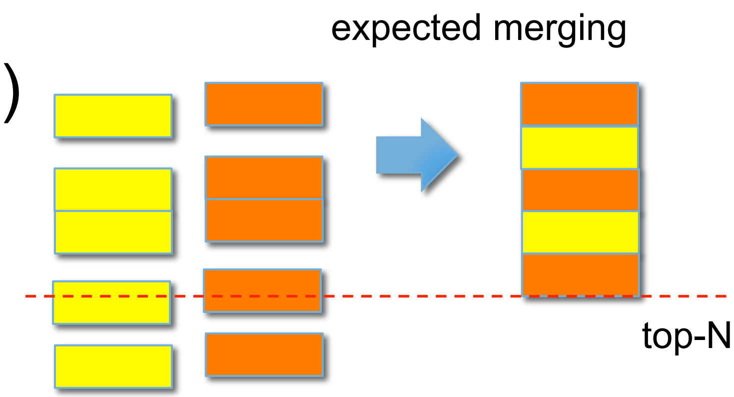
Investigation of outliers

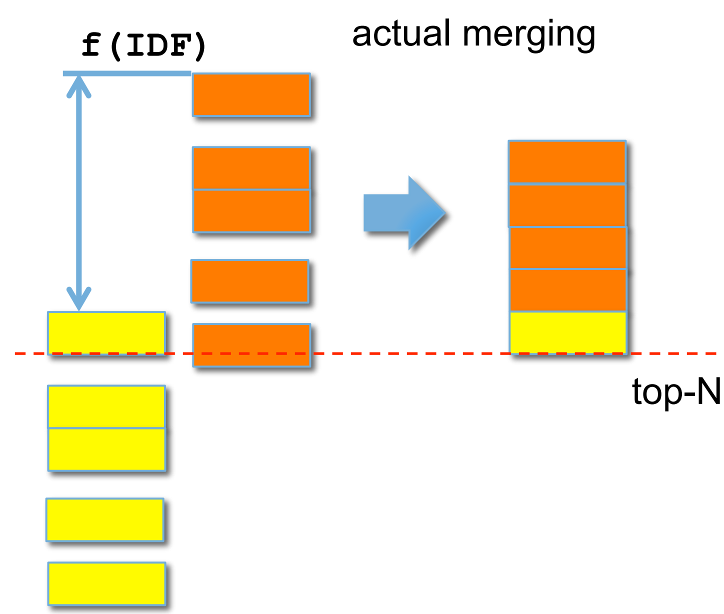
- OHSU results have similar scores (OR)
- IDF differences per shard
  - Affect all scores from a shard
  - Scores of partial result lists become shifted by a variable factor `f(IDF)`
- Evenly divided shards – half of results from each shard
- HOWEVER!

Variable score diff + closely spaced scores =

Top-N dominated by one shard only

- Paradox: uneven shards –> results may merge with a smaller loss in top-N
  - BUT results from a smaller shard may be totally lost from Top-N !





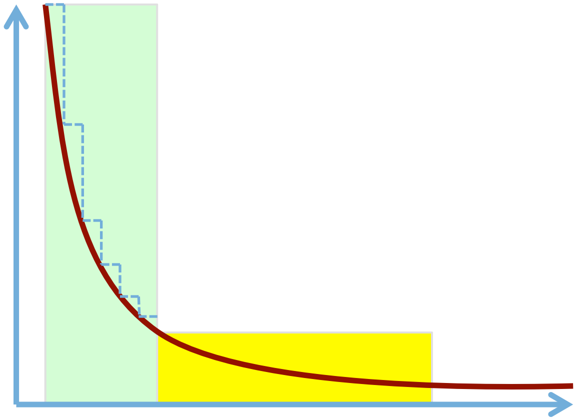
Global ranking

- Merging partial lists needs to be smarter
  - Are scores comparable across shards?
    - Apparently not always (maybe not usually?)
    - **Even with equally-sized shards!**
  - Is top-N from one shard “worse” because it uses lower scores than top-N from another shard?
    - Apparently not always
- Proposed improvement (in absence of global IDF)
  - Top-N from different shards could be normalized to start from the same initial score for top-1
    - Shift down results from a smaller shard by x positions ?
- QueryComponent patches are welcome ☺

Global IDF

- Main source of different scores in the experiment
- Lucene IndexSearcher can be modified to use custom IDF values
  - Query -> Weight includes IDF weights
- Exact IDF – two round-trips
  1. Submit the query to shards to obtain terms and per-shard IDF for each term
  2. Collect and aggregate IDFs from shards into global IDFs
  3. Submit the query + global IDFs
  - Modified IndexSearcher can use the aggregated IDFs to produce Weight-s (and scores) that reflect global IDF
  - Result: absolute values of scores become comparable
  - Problem: two round-trips per query
- SOLR-1632 (still needs work)

Estimated global IDF



- Zipf-ian distribution of term frequencies
  - ~ 50% of terms have frequency lower than 10
  - IDF doesn't have to be exact, just consistent across shards
- Compact representation of the distribution
  - Example 1:
    - List of the top 1000 terms+freqs, maybe in buckets
    - Bloom filter of any other term with freq > 5
    - Skip all other terms (assume freq = 1, or freq = O(shard size))
  - Example 2: Count-Min sketches (exercise for the reader ☺)
- Periodically broadcast this compact structure to all other shard servers
  - E.g. after large updates when IDF changes significantly (> 40% diff)
- Cons: not exact, takes memory, broadcast traffic
- Pros: one round-trip per query! "Goodenuf" quality

Latency and comm errors

- Latency determined by the slowest shard (straggler)
  - The more shards the larger the max latency, unbounded
  - Limit the max latency at the cost of losing some results?
  - Replicate most loaded shards and load-balance requests?
- Communication errors – they will happen…
  - Solr gives up too easily
  - Quick handling needed
    - Sufficient quorum within a time limit
  - Accept partial results by default
- SearchHandler improvements are welcome ☺

Conclusions

- Distributed search is a must as your index grows
  - …but until then a single index is strongly preferred!
- Distributed search in Solr works … with caveats
  - Cumbersome shard management / distributed indexing
  - Quality of search affected by different scoring per shard
    - Too simplistic method of merging partial lists
    - Lack of global IDF, either exact or estimated, makes scores incomparable
  - Fragile – better handling of comm errors needed
  - Unbounded latency – better handling of straggler shards needed
- Let's fix it!

References

- *Comparing top-k lists*, R. Fagin, R. Kumar, D. Sivakumar, SIAM 2003
- *Overlap-Aware Global df Estimation in Distributed Information Retrieval Systems*, Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, MPI–I–2006–5-001
- *Estimation of global term weights for distributed and ubiquitous IR*, Hans Friedrich Witschel, Elsevier 2007
- *Exploring the Stability of IDF Term Weighting*, Xin Fu, Miao Chen, Springer 2008
- *A Comparison of Techniques for Estimating IDF Values to Generate Lexical Signatures for the Web*, M. Klein, M.L. Nilson, WIDM'08
- *Robust Result Merging Using Sample-Based Score Estimates*, M. Shoukouhi, J. Zobel, ACM TOIS 2009

Summary & QA

- Distributed search concepts
- Apache Solr distributed search
- Experiments
- Challenges: analysis and solutions

- More questions?

ab@lucidimagination.com

APACHE
LUCENE
EUROCON
Barcelona 2011

17-18 October 2011 | Training
19-20 October 2011 | Conference

CALL FOR PARTICIPATION NOW OPEN:
http://2011.lucene-eurocon.org

PRESENTED BY: lucid IMAGINATION

ALL PROCEEDS BENEFIT THE APACHE SOFTWARE FOUNDATION