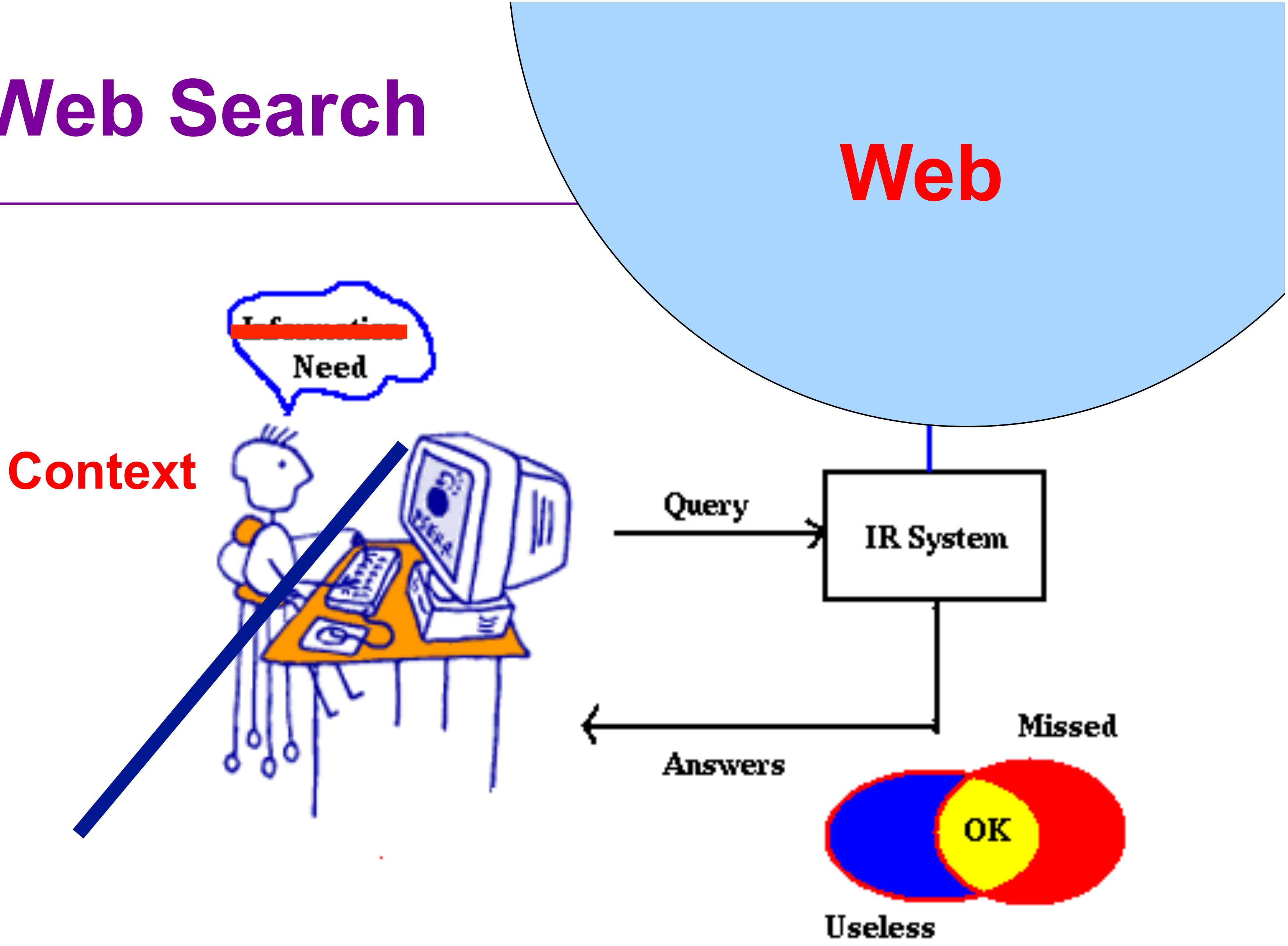
Towards a Distributed Web Search Engine

Ricardo Baeza-Yates
Yahoo! Research
Barcelona, Spain

Joint work with **Barla Cambazoglu, Aristides Gionis, Flavio Junqueira, Mauricio Marín, Vanessa Murdock** (Yahoo! Research)
and many other people

Web Search



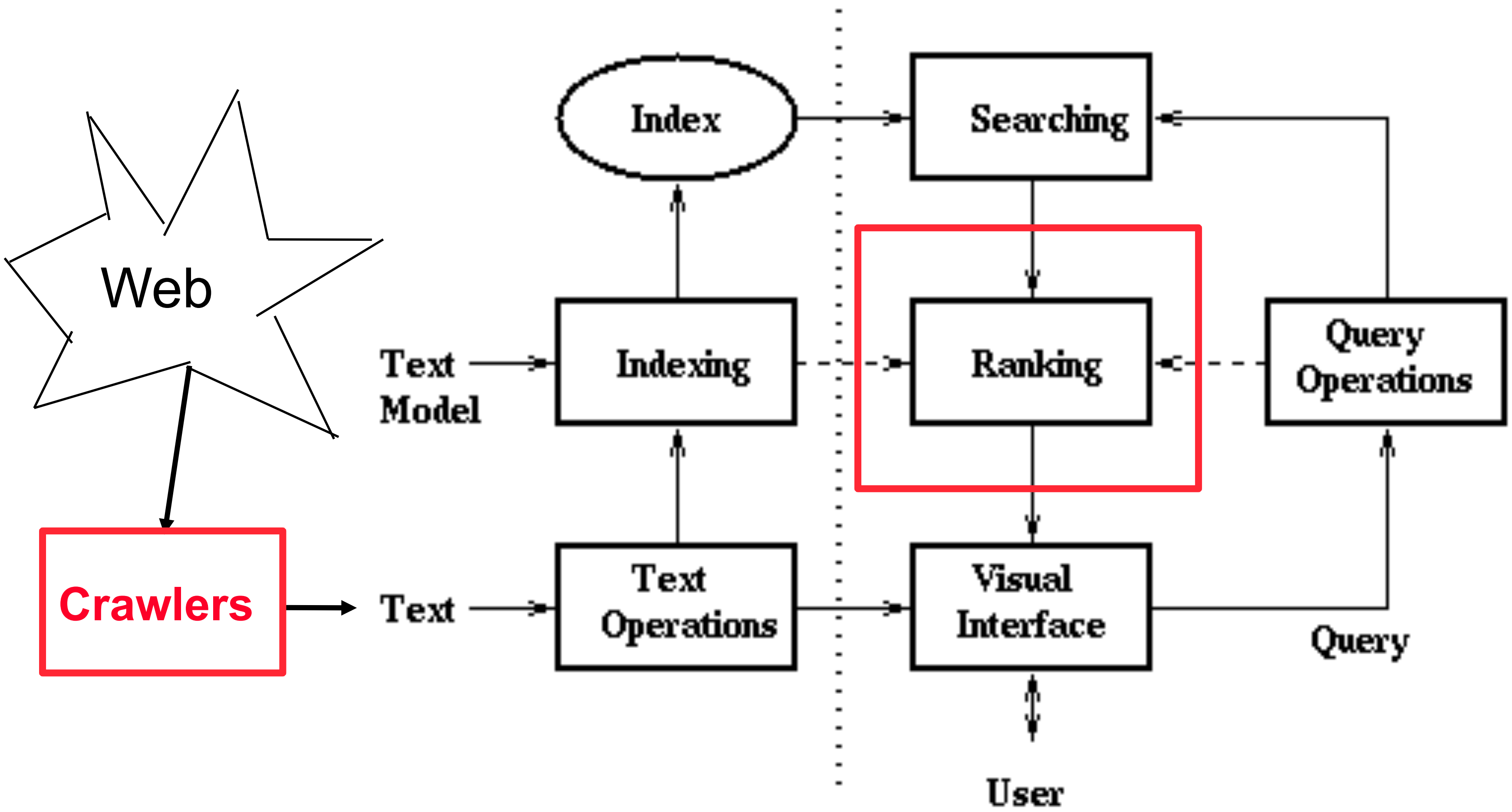
Web
~~Information~~
Need
Context
Query
IR System
Answers
Missed
OK
Useless

Web Search

- This is one of the most complex data engineering challenges today:
  - Distributed in nature
  - Large volume of data
  - Highly concurrent service
  - Users expect very good & fast answers

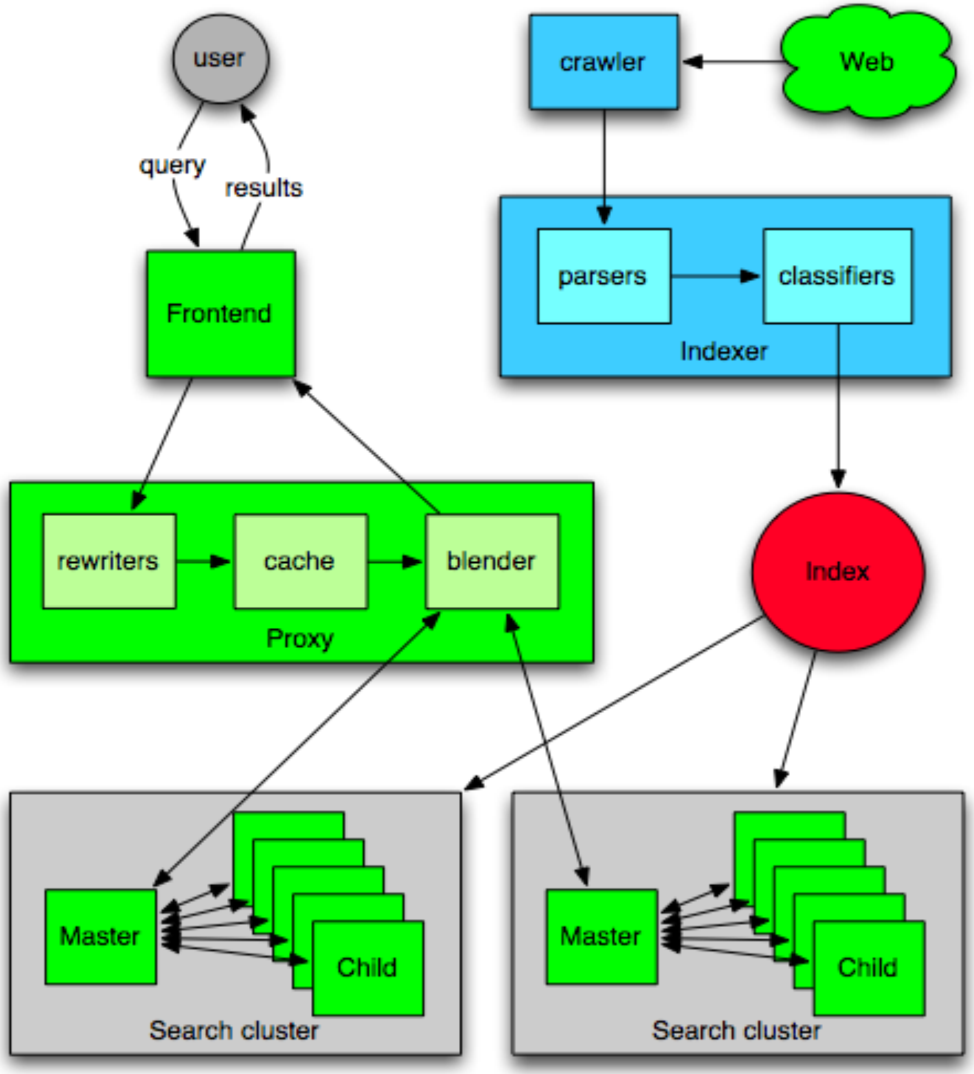
- Current solution: Replicated centralized system

WR Logical Architecture

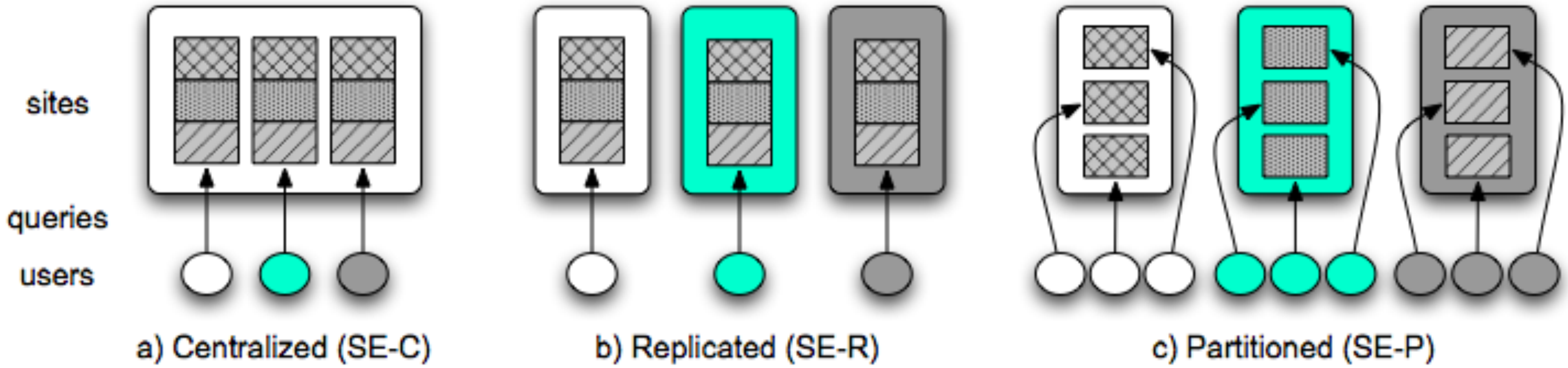


A Typical Web Search Engine

- Caching
  - result cache
  - posting list cache
  - document cache
- Replication
  - multiple clusters
  - improve throughput
- Parallel query processing
  - partitioned index
    - document-based
    - term-based
  - Online query processing



Search Engine Architectures



- **Architectures differ in**
  - **number of data centers**
  - **assignment of users to data centers**
  - **assignment of index to data centers**

System Size

- 20 billion Web pages implies at least 100Tb of text
- The index in RAM implies at least a cluster of 10,000 PCs
- Assume we can answer 1,000 queries/sec
- 350 million queries a day imply 4,000 queries/sec
- Decide that the peak load plus a fault tolerance margin is 3
- This implies a replication factor of 12 giving 120,000 PCs
- Total deployment cost of over 100 million US$ plus maintenance cost
- In 201x, being conservative, we would need over 1 million computers!

Questions

- Should we use a centralized system?
- Can we have a (cheaper) distributed search system in spite of network latency?

- Preliminary answer: **Yes**
- Solutions: caching, new ways of partitioning the index, exploit locality when processing queries, prediction mechanisms, etc.

Advantages

- Distribution decreases replication, crawling, and indexing and hence the cost per query
- We can exploit high concurrency and locality of queries
- We could also exploit the network topology
- Main design problems:
  - Depends upon many external factors that are seldom independent
  - One poor design choice can affect performance or/and costs

Challenges

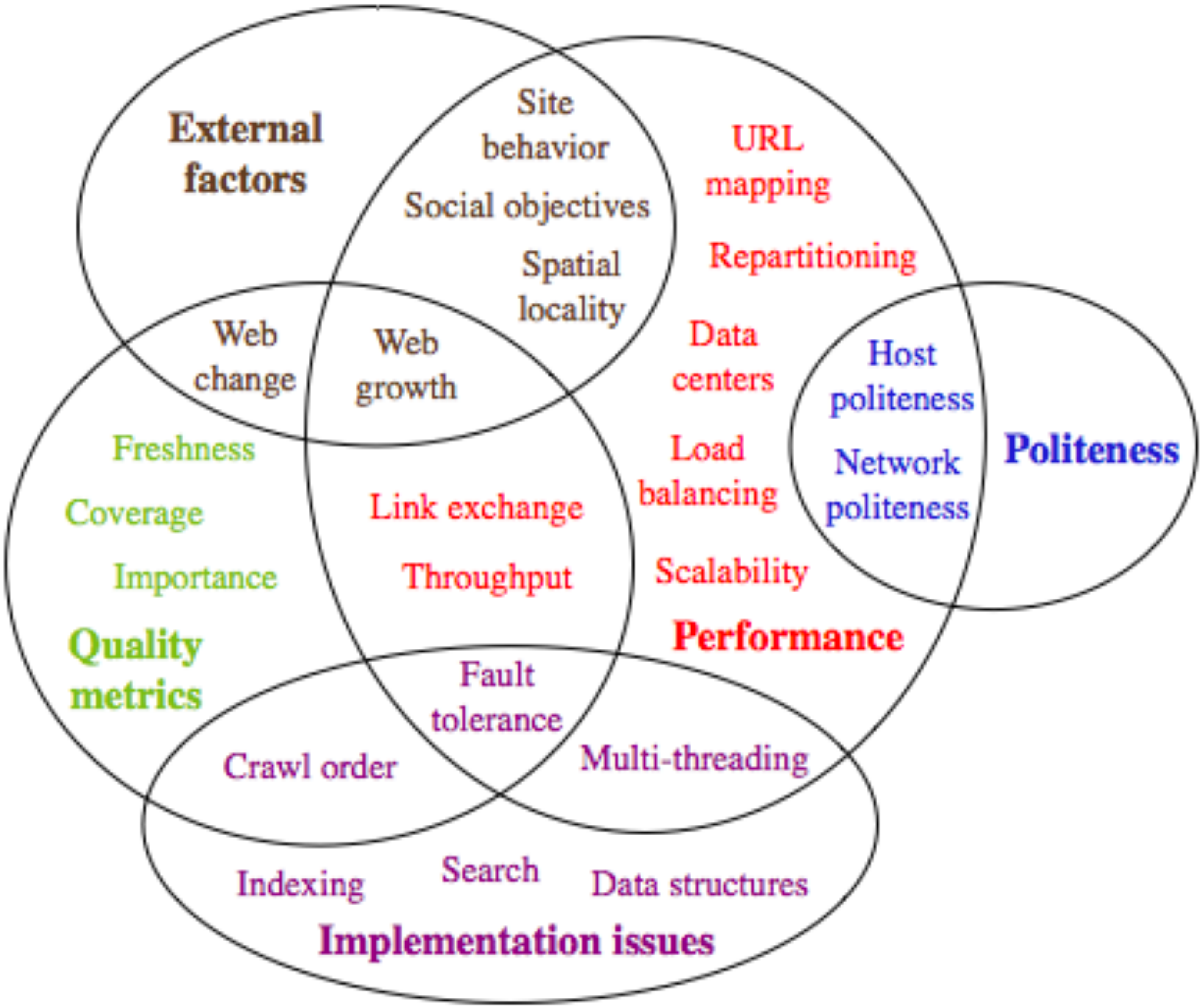
- Must return high quality results (handle quality diversity and fight spam)
- Must be fast (fraction of a second)
- Must have high capacity
- Must be dependable (reliability, availability, safety and security)
- Must be scalable

Crawling

- Index depends on good crawling
  - Quality, quantity, freshness
- Crawling is a scheduling problem
  - NP hard
- Difficult to optimize and to evaluate
- Distributed crawling:
  - Closer to data, less network usage and latency

Too Many Factors

- Quality metrics
- External factors
- Performance
- Implementation issues
- Politeness



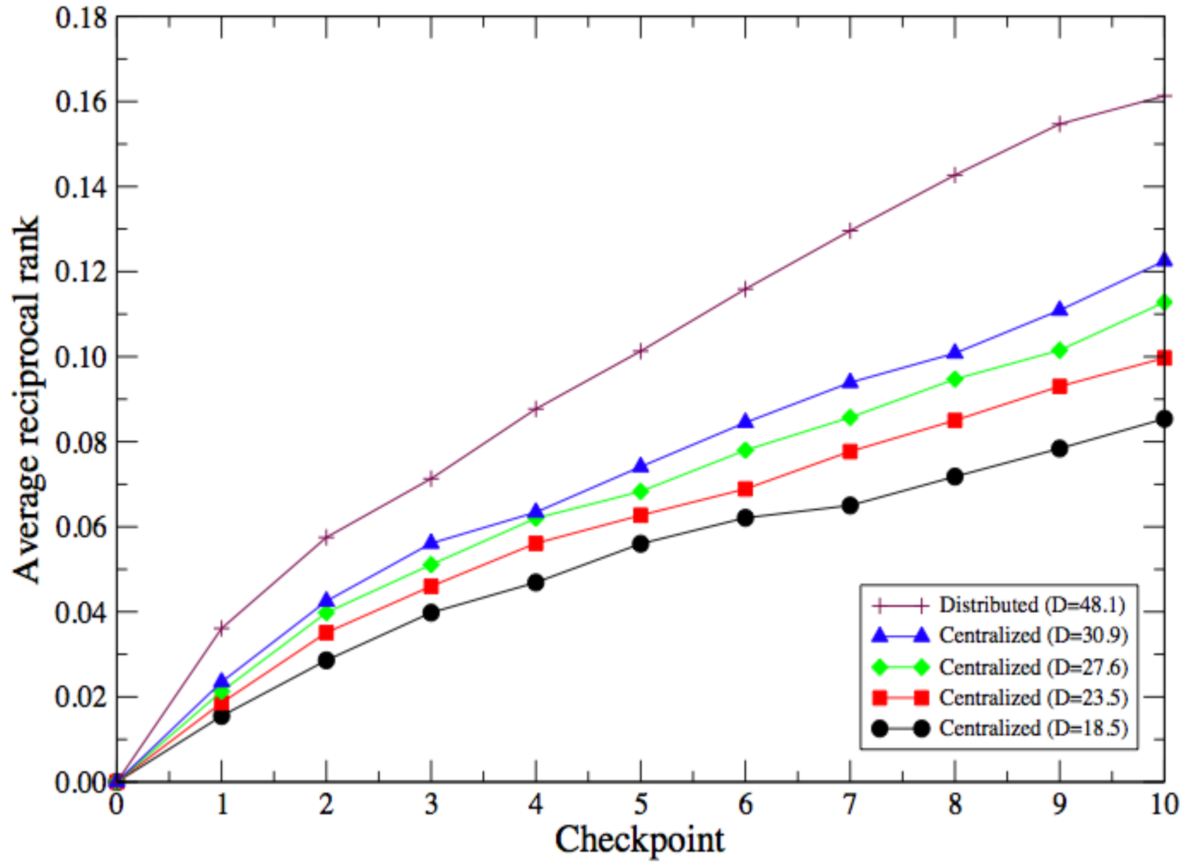
YAHOO!

Impact of Distributed Web Crawling on Relevance [Cambazoglu et al, SIGIR 2009]

- Objective: See the impact of higher page download rates on search quality
- Random sample of 102 million pages partitioned into five different geographical regions
  - location of Web servers
  - page content
- Query sets from the same five regions
- Ground-truth: clicks obtained from a commercial search engine
- Ranking: a linear combination of a BM25 variant and a link analysis metric
- Search relevance: average reciprocal rank

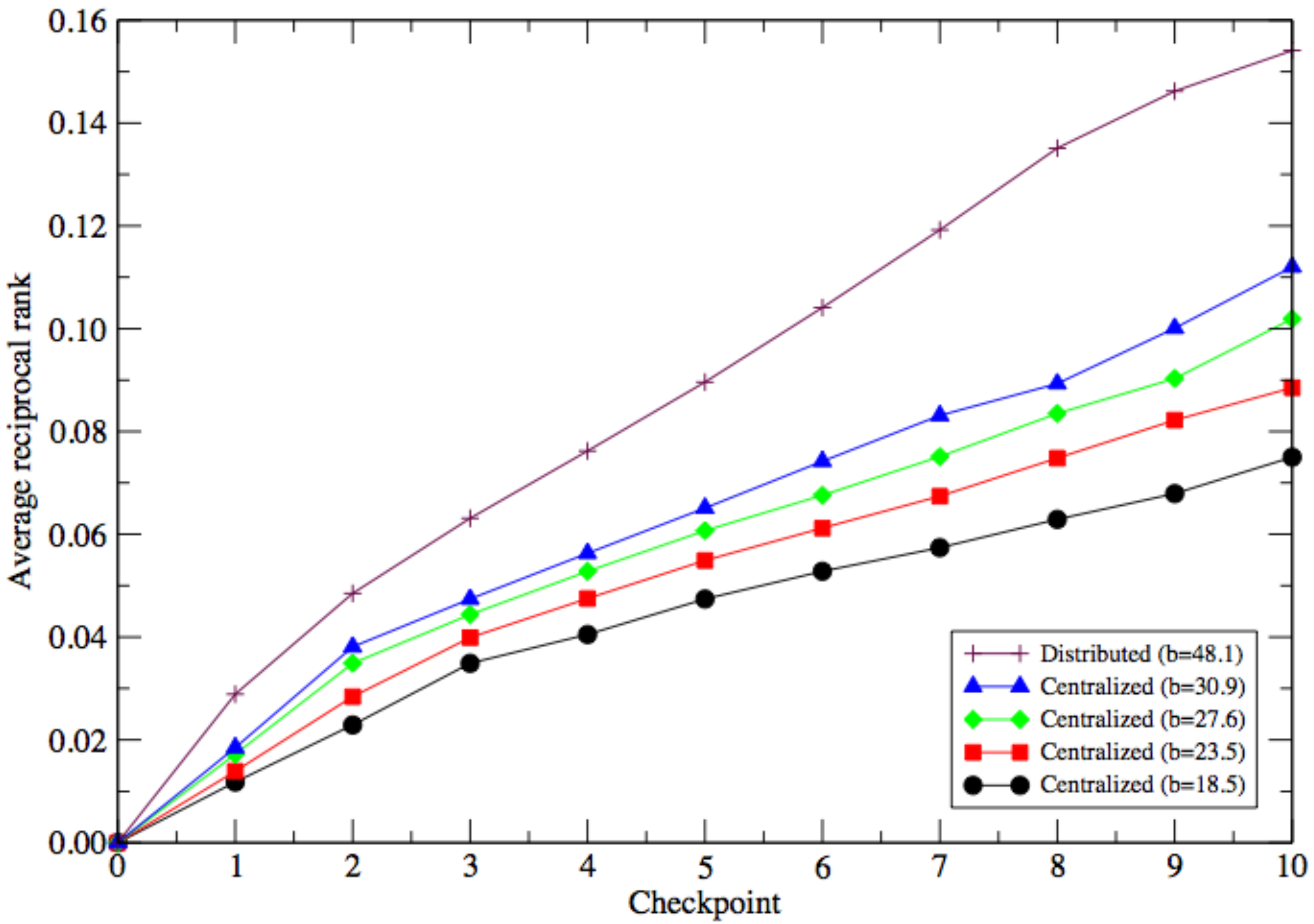
Impact of Download Speed

- **Distributed crawling simulator with varying download rates**
  - **distributed: 48 KB/s**
  - **centralized:**
    - **30.9 KB/s (US)**
    - **27.6 KB/s (Spain)**
    - **23.5 KB/s (Brazil)**
    - **18.5 KB/s (Turkey)**



- **Checkpoint *i*: the point where the fastest crawler in the experiment downloaded 10*i* % of all pages**
- **Crawling order: random**

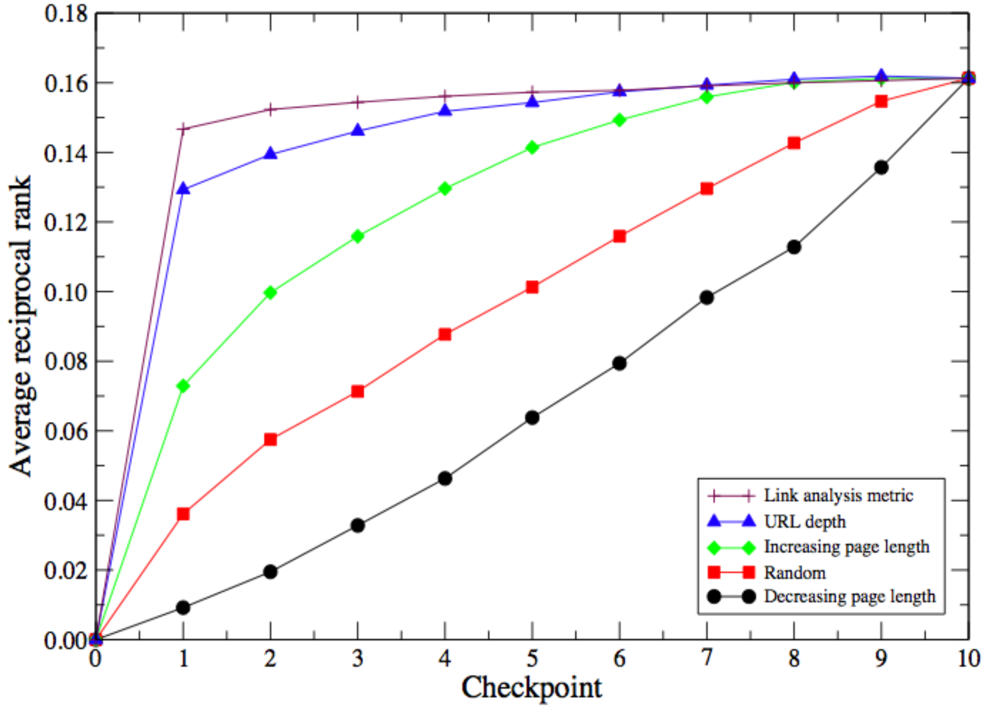
Impact of Download Speed



Impact of Crawling Order

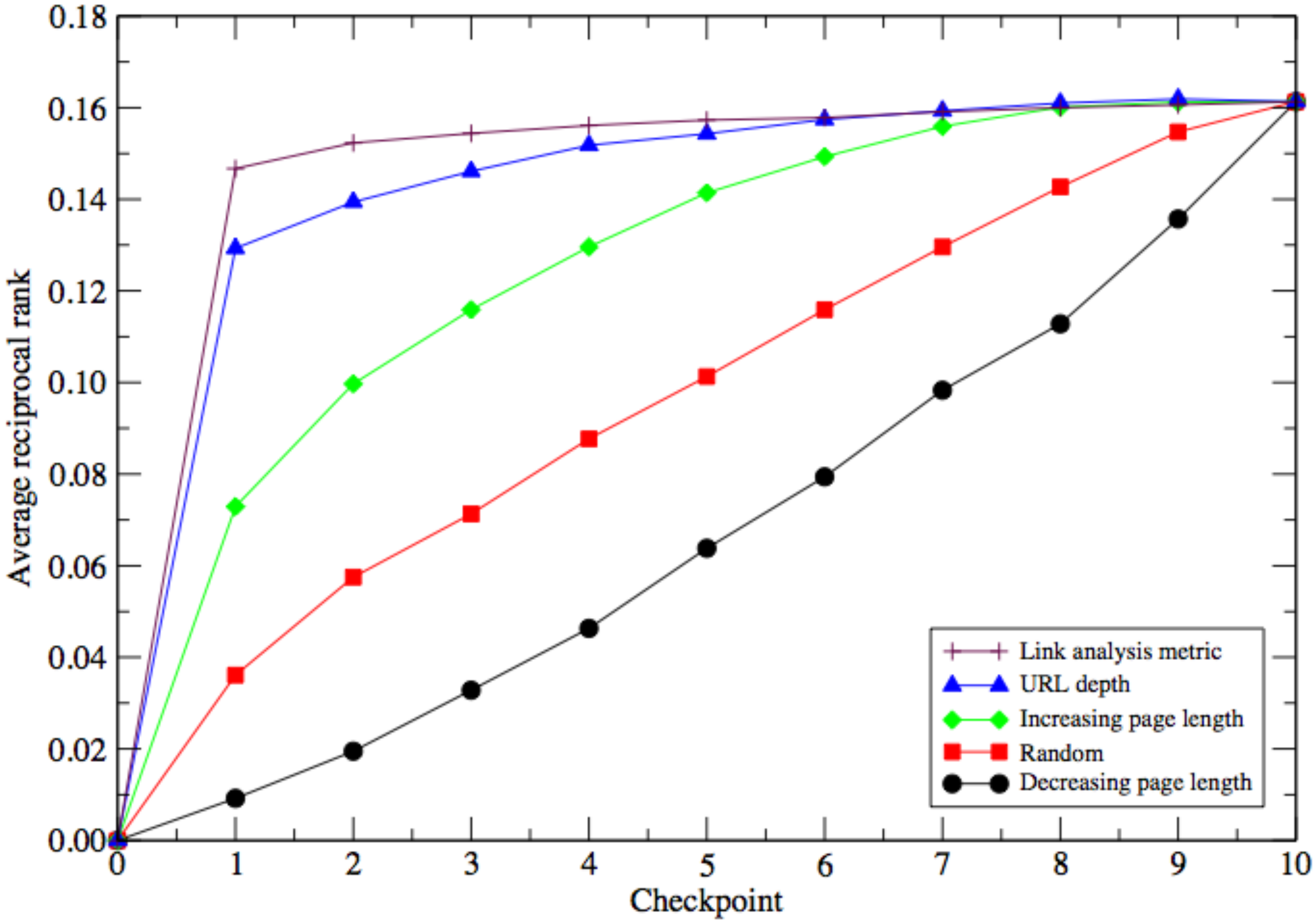
- **Varying crawling orders:**
  - **link analysis metric**
  - **URL depth**
  - **increasing page length**
  - **random**
  - **decreasing page length**

- **Download throughput: 48.1 KB/s**



	D			
Ordering strategy	18.5	23.5	27.6	30.9
Decreasing page length	0.041	0.058	0.072	0.086
Random	0.085	0.100	0.113	0.123
Increasing page length	0.130	0.143	0.150	0.154
URL depth	0.153	0.156	**0.158**	**0.159**
Link analysis metric	**0.156**	**0.158**	0.157	**0.159**

Impact of Crawling Order



Impact of Region Boosting

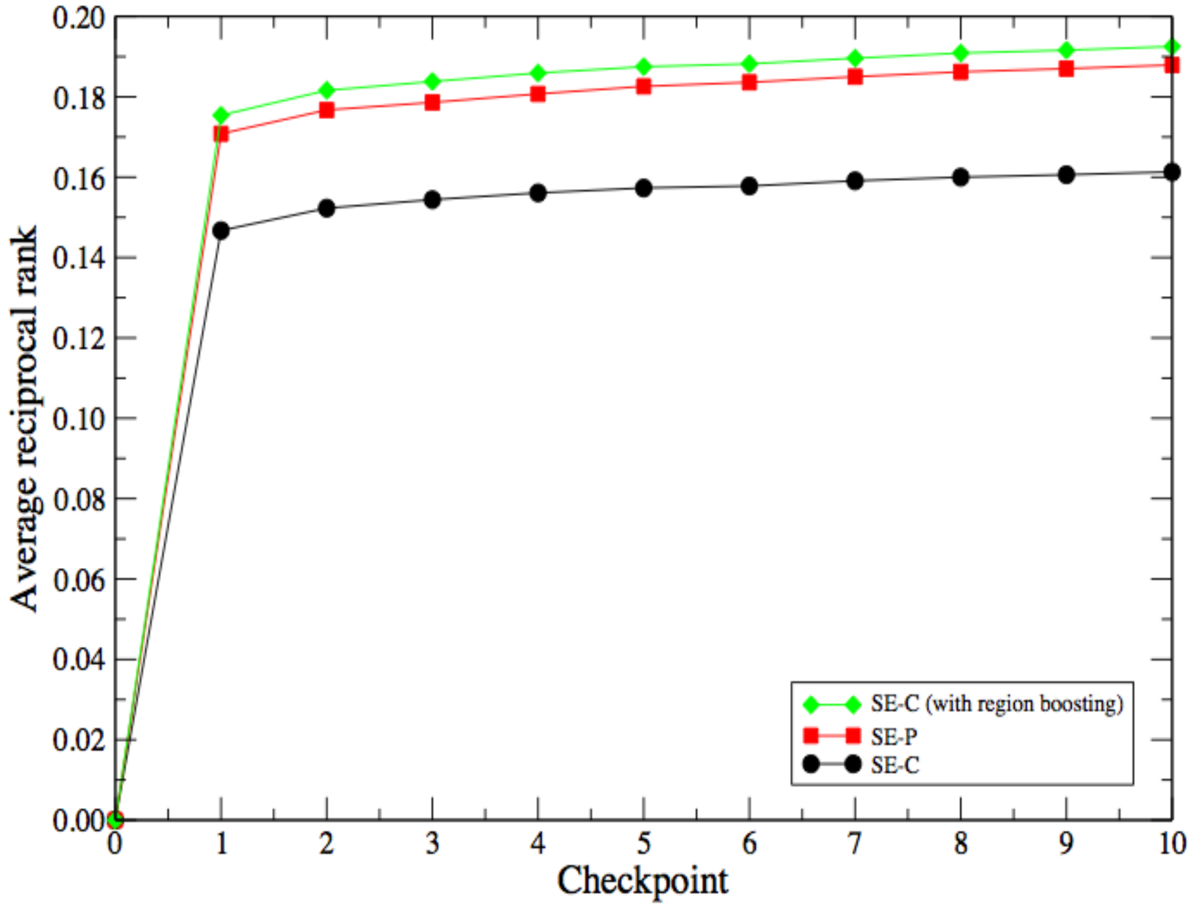
- **Region boosting**
  - **SE-C**

  (with region boosting)
  - **SE-P**

  (natural region boosting)
  - **SE-C**

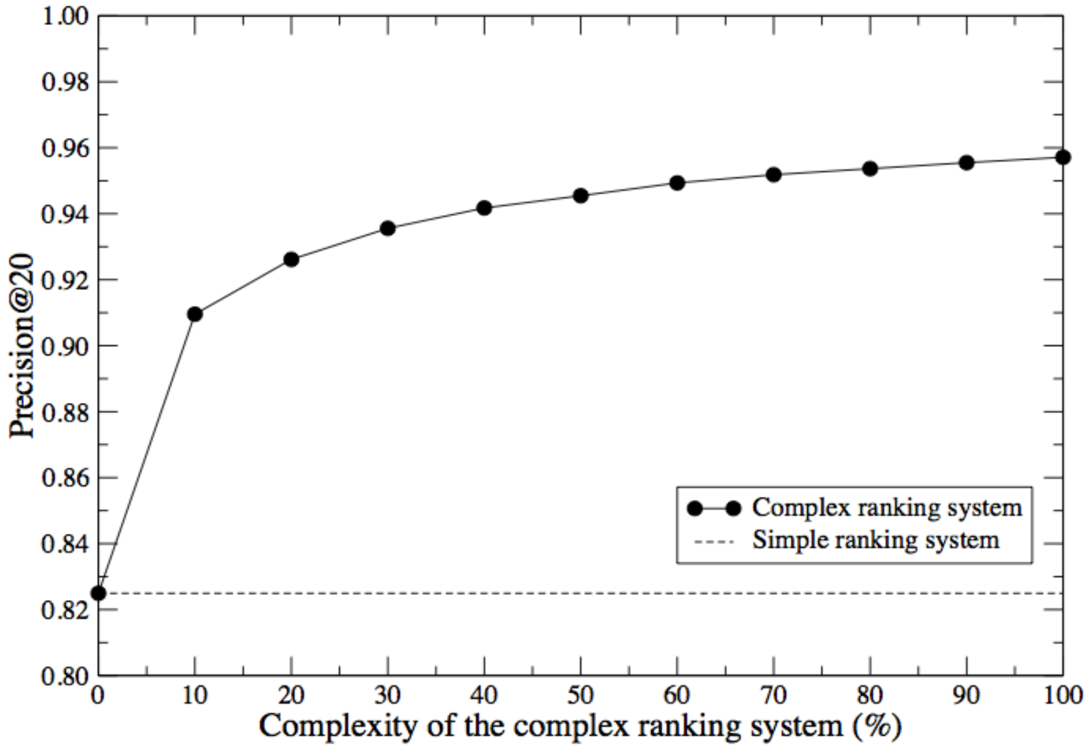
  (without region boosting)

- **Download throughput: 48.1 KB/s**



Search Relevance (Cambazoglu et al, SIGIR 2009)

- **Assuming we have more time for query processing, we can**
  - relax the "AND" requirement
  - score more documents
  - use more complex scoring techniques
    - costly but accurate features
    - costly but accurate functions

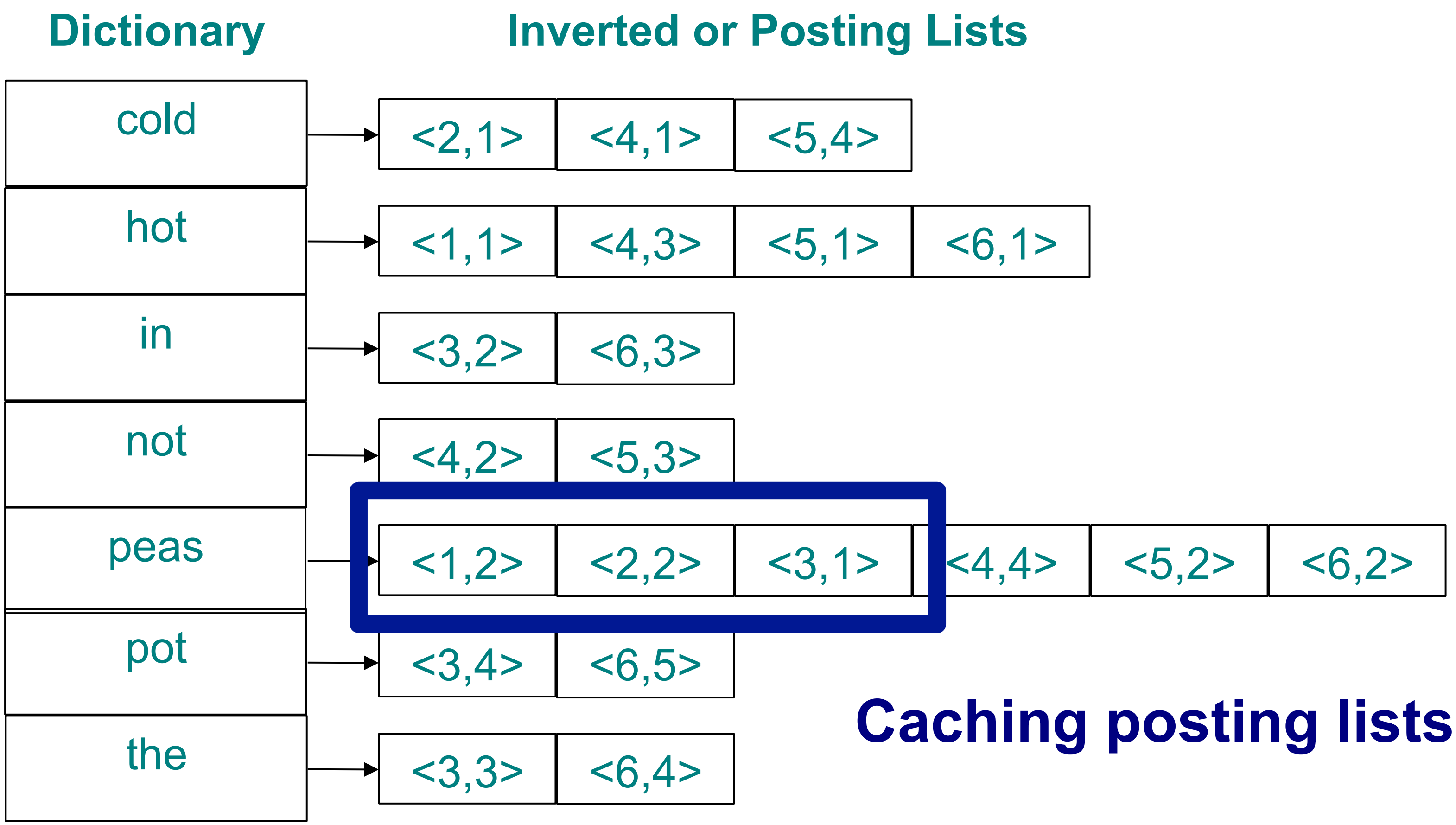


- **Ground-truth: top 20 results**
- **Baseline: linear combination of a BM25 variant with a link analysis metric**
- **A complex ranking function composed of 1000 scorers**

Caching

- Caching can save significant amounts of computational resources
  - Search engine with capacity of 1000 queries/second
  - Cache with 30% hit ratio increases capacity to 1400 queries/second
- Caching helps to make queries “local”
- Caching is similar to replication on demand
- Important sub-problem:
  - Refreshing stale results (Cambazoglu *et al*, WWW 2010)

Inverted Index

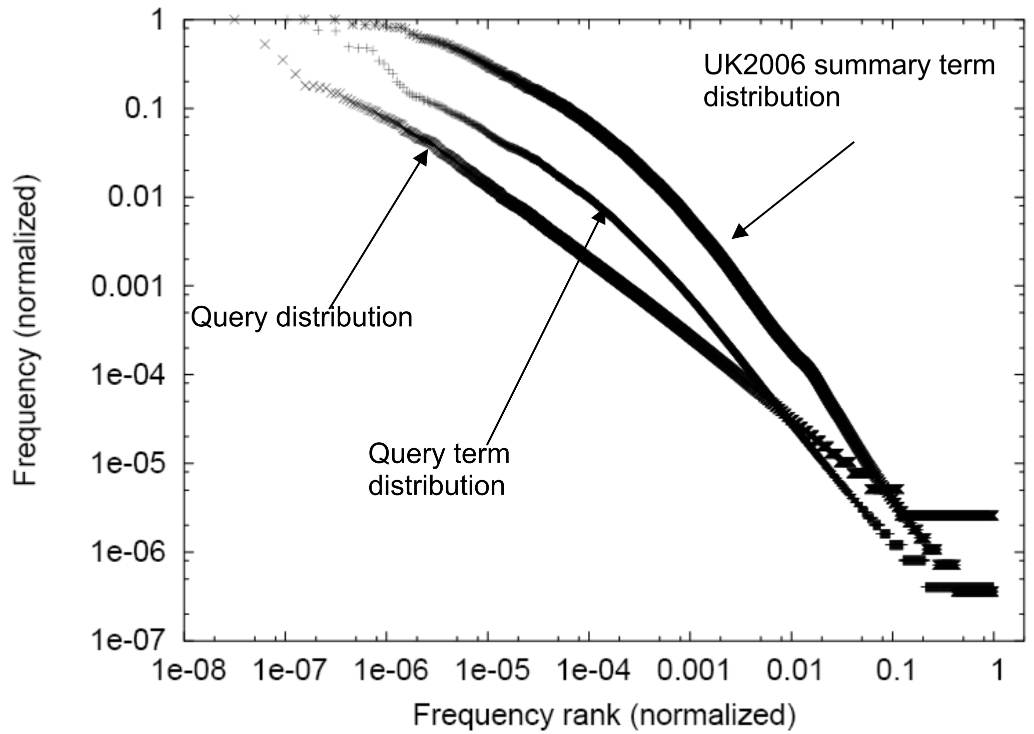


Caching in Web Search Engines

- Caching **query results** *versus* caching **posting lists**
- **Static** *versus* **dynamic** caching policies
- Memory allocation between different caches
- Caching reduce **latency** and **load** on back-end servers
- Baeza-Yates et al, SIGIR 2007

Data Characterization

- 1 year of queries from Yahoo! UK
- UK2006 summary collection
- Pearson correlation between query term frequency and document frequency = 0.424



What you write is NOT what you want

Caching Query Results or Term Postings?

- Queries
  - 50% of queries are unique (vocabulary)
  - 44% of queries are singletons (appear only once)
  - Infinite cache achieves 50% hit-ratio
    - Infinite hit ratio = (#queries – #unique) / #queries

- Query terms
  - 5% of terms are unique
  - 4% of terms are singletons
  - Infinite cache achieves 95% hit ratio

Static Caching of Postings

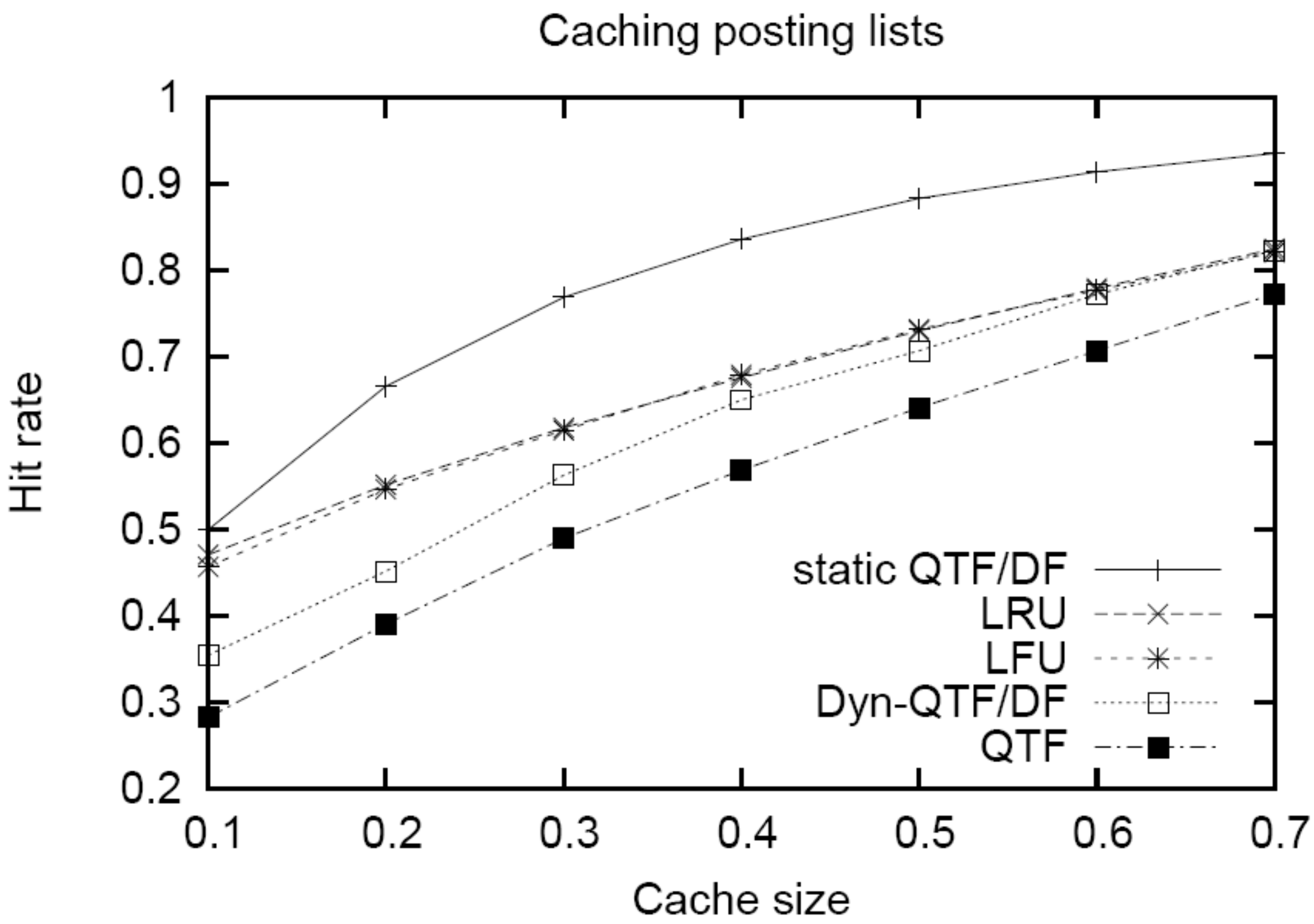
- QTF for static caching of postings (Baeza-Yates & Saint-Jean, 2003):
  - Cache postings of terms with the highest $f_q(t)$
- Trade-off between $f_q(t)$ and $f_d(t)$
  - Terms with high $f_q(t)$ are good to cache
  - Terms with high $f_d(t)$ occupy too much space
- QTFDF: Static caching of postings
  - Knapsack problem:
  - Cache postings of terms with the highest $f_q(t)/f_d(t)$

Evaluating Caching of Postings

- Static caching:
  - QTF : Cache terms with the highest query log frequency $f_q(t)$
  - QTFDF : Cache terms with the highest ratio $f_q(t) / f_d(t)$

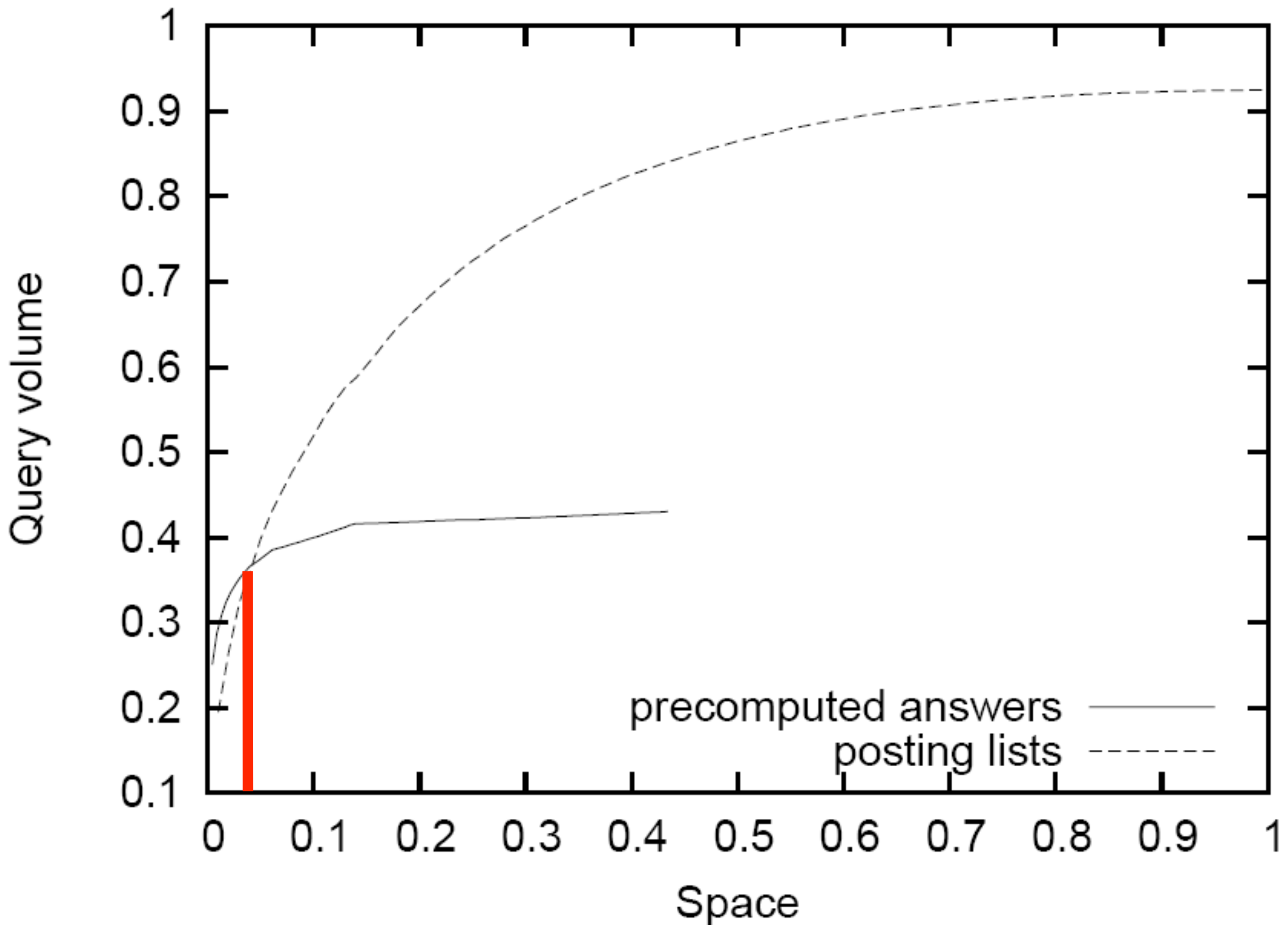
- Dynamic caching:
  - LRU, LFU
  - Dynamic QTFDF : Evict the postings of the term with the lowest ratio $f_q(t) / f_d(t)$

Results



Caching posting lists
Hit rate
Cache size
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
static QTF/DF
LRU
LFU
Dyn-QTF/DF
QTF

Combining caches of query results and term postings

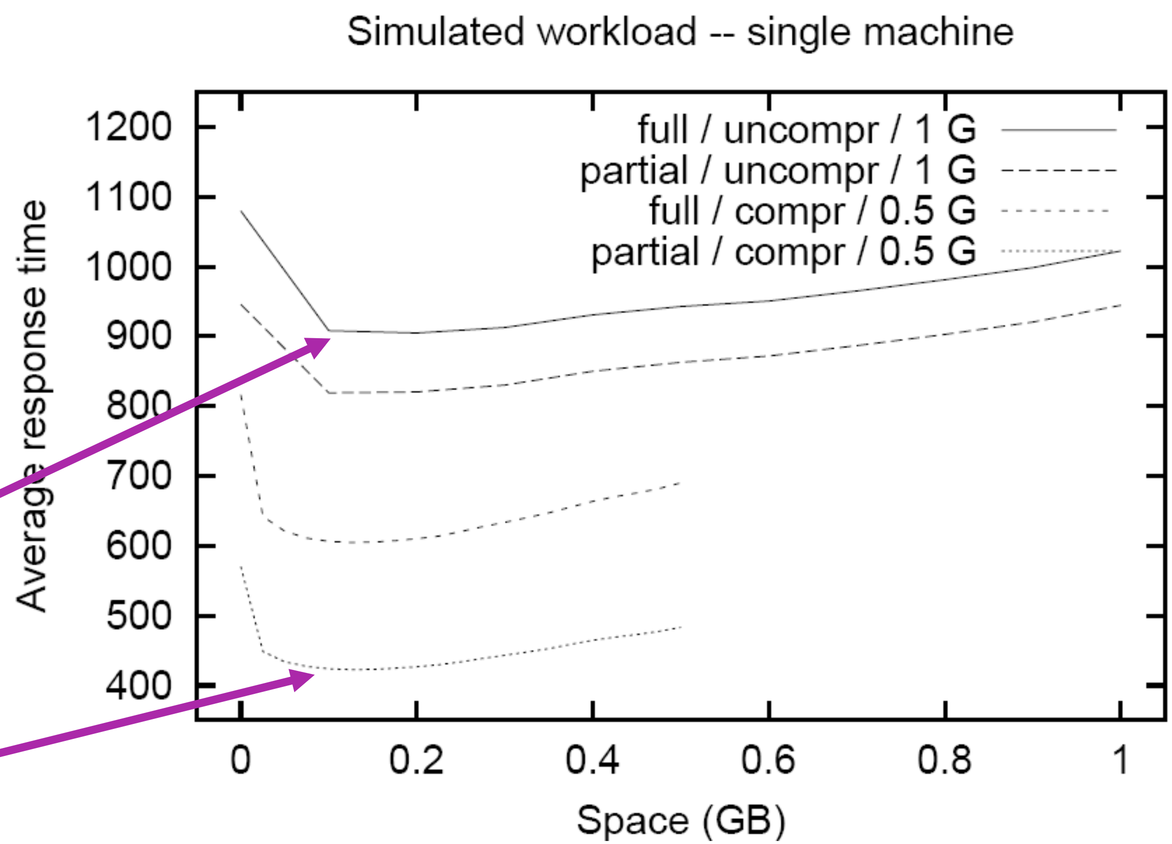


Experimental Setting

- Process 100K queries on the UK2006 summary collection with Terrier
- Centralized IR system
  - Uncompressed/compressed posting lists
  - Full/partial query evaluation
- Model of a distributed retrieval system
  - broker communicates with query servers over LAN or WAN

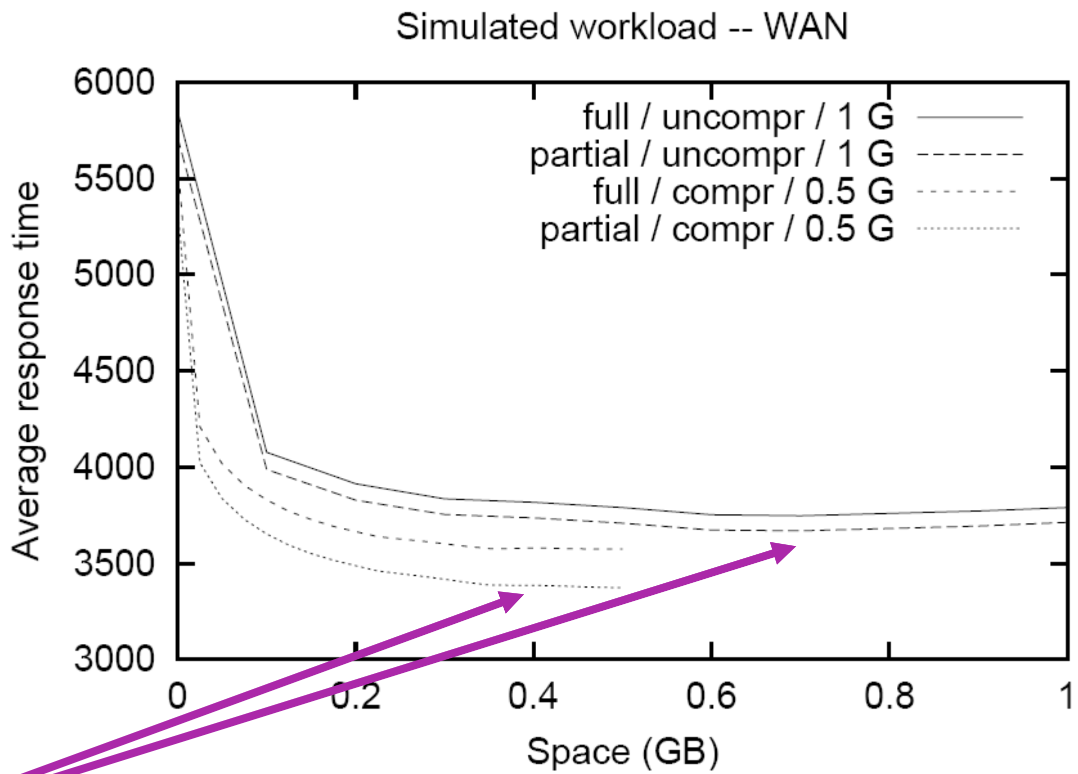
Centralized System Simulation

- Assume M memory units
  - x memory units for static cache of query results
  - M-x memory units for static cache of postings
- Full query evaluation with uncompressed postings
  - 15% of M for caching query results
- Partial query evaluation with compressed postings
  - 30% of M for caching query results



WAN System Simulation

- Distributed search engine
  - Broker holds query results cache
  - Query processors hold posting list cache
- Optimal Response time is achieved when most of the memory is used for caching answers



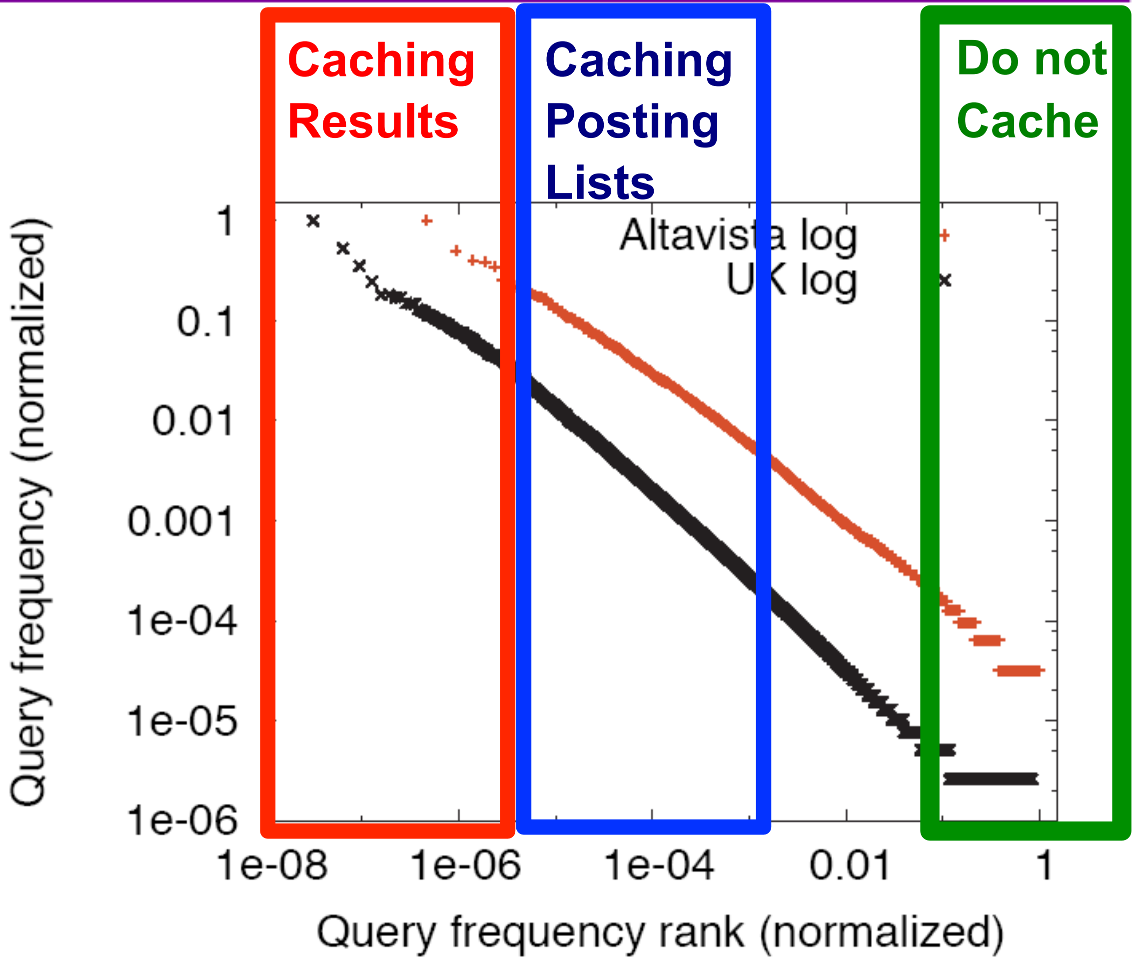
Query Dynamics

- Static caching of query results
  - Distribution of queries change slowly
  - A static cache of query results achieves high hit rate even after a week

- Static caching of posting lists
  - Hit rate decreases by less than 2% when training on 15, 6, or 3 weeks
  - Query term distribution exhibits very high correlation (>99.5%) across periods of 3 weeks

Why caching results can't reach high hit rates

- AltaVista: 1 week from September 2001
- Yahoo! UK: 1 year
  - Similar query length in words and characters
- Power-law frequency distribution
  - Many infrequent queries and even singleton queries
- No hits from singleton queries



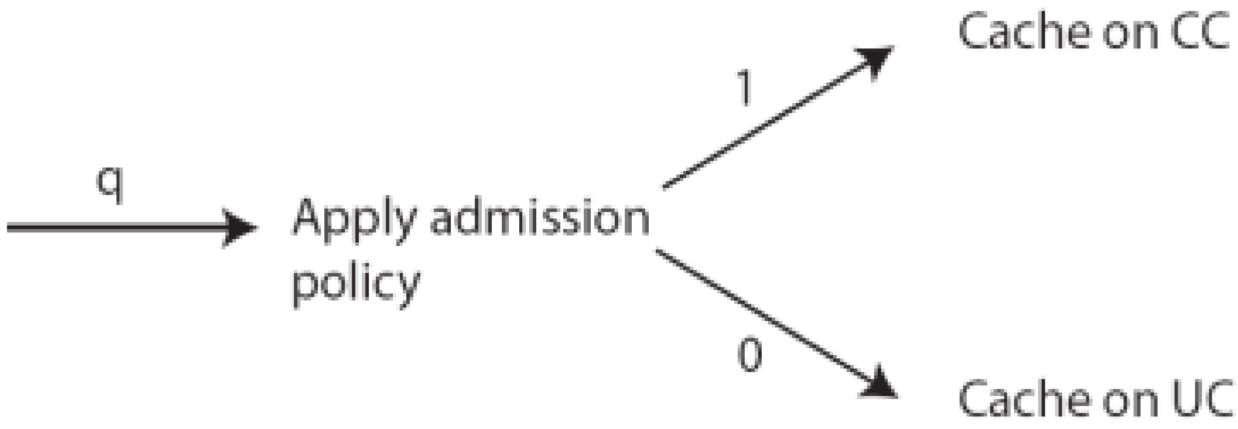
Benefits of filtering out infrequent queries

- Optimal policy does not cache singleton queries
- Important improvements in cache hit ratios

Cache size	Optimal		LRU	
	AV	UK	AV	UK
50k	**67.49**	**32.46**	59.97	17.58
100k	**69.23**	**36.36**	62.24	21.08
250k	**70.21**	**41.34**	65.14	26.65

Admission Controlled Cache (AC)

- General framework for modelling a range of cache policies



- Split cache in two parts
  - Controlled cache (CC)
  - Uncontrolled cache (UC)
- Decide if a query q is frequent enough
  - If yes, cache on CC
  - Otherwise, cache on UC

Baeza-Yates et al, SPIRE 2007

Why an uncontrolled cache?

- Deal with errors in the predictive part
- Burst of new frequent queries
- Open challenge:
  - How the memory is split in both types of cache?

Features for admission policy

- Stateless features
  - Do not require additional memory
  - Based on a function that we evaluate over the query
  - Example: query length in characters/terms
    - Cache on CC if query length < threshold

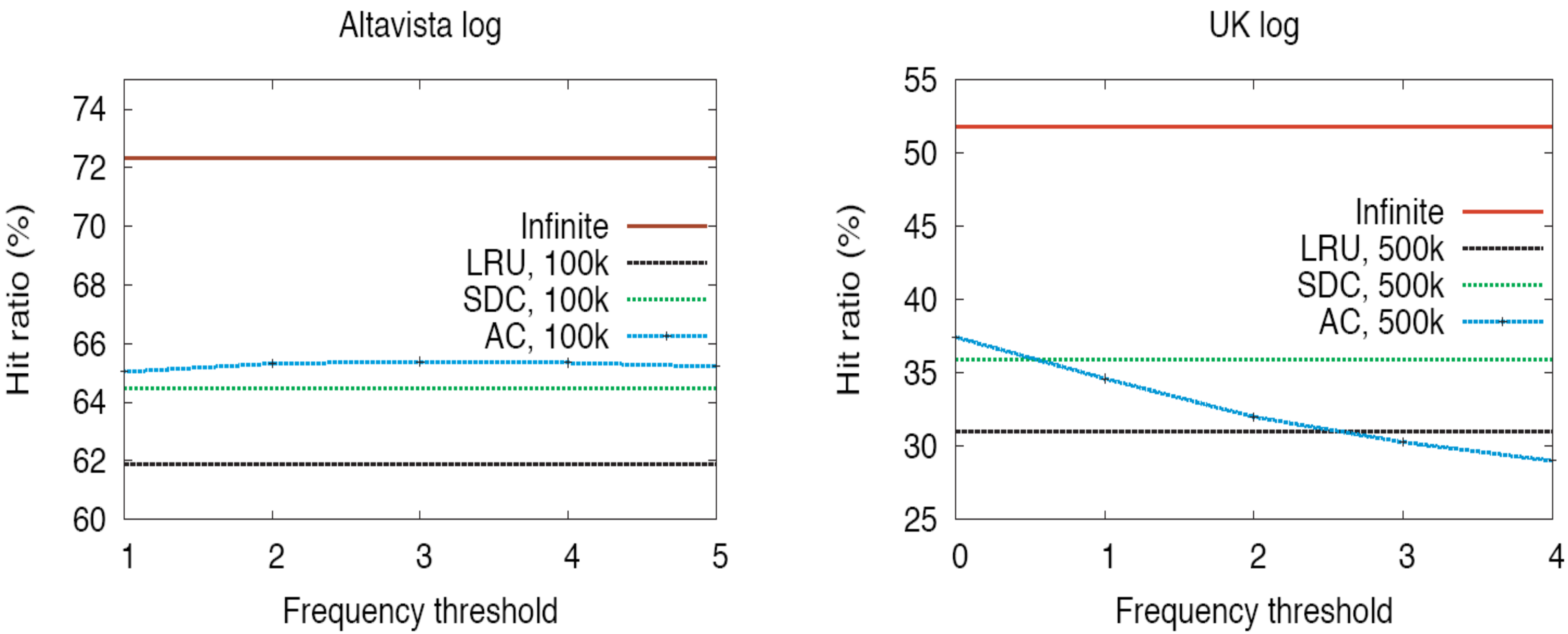
- Stateful features
  - Uses more memory to enable admission control
  - Example: past frequency
    - Cache on CC if its past frequency > threshold
    - Requires only **a fraction** of the memory used by the cache

Evaluation

- AltaVista and Yahoo! UK query logs
  - First 4.8 million queries for training
  - Testing on the rest of the queries

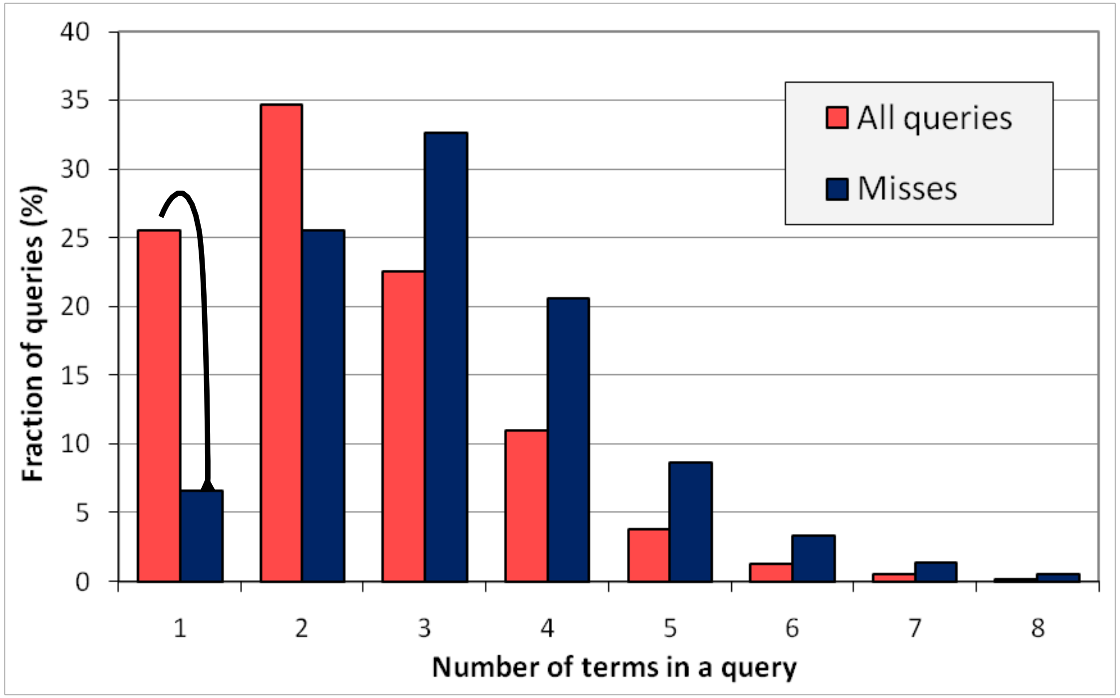
- Compare AC with
  - LRU: Evicts the least recent query results
  - SDC: Splits cache into two parts
    - Static: filled up with most frequent past queries
    - Dynamic: uses LRU

Results for Stateful Features



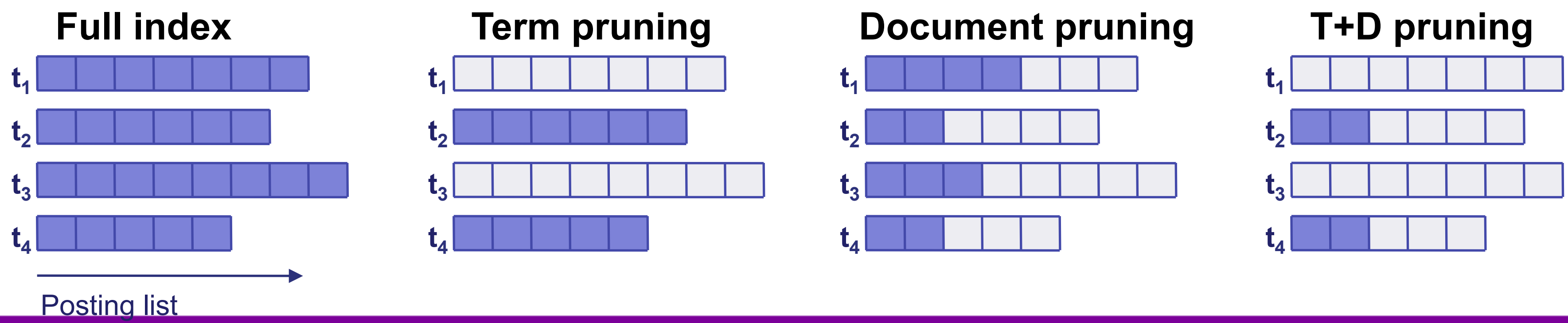
All queries vs. *Misses*: Number of terms in a query

- Average number of terms for *all queries* = **2.4**
  , for *misses* = **3.2**
- Most single term queries are hits in the results cache
- Queries with many terms are unlikely to be hits



Static index pruning (Skobeltsyn et al, SIGIR08)

- Smaller version of the main index after the cache, returns:
  - the top-*k* response that is *the same* to the main index's, or
  - a *miss* otherwise.
- Assumes Boolean query processing
- Types of pruning:
  - **Term pruning** – full posting lists for selected terms
  - **Document pruning** – prefixes of posting lists
  - **Term+Document pruning** – combination of both



Analysis of Results

- **Static index pruning**: addition to results caching, not replacement
  - **Term pruning** performs well for *misses* also

  => can be combined with results cache
  - **Document pruning** performs well for *all queries,* but requires high Pagerank weights with *misses*
  - **Term+Document pruning** improves over document pruning, but has the same disadvantages

- **Pruned index** grows with collection size

- Document **pruning** targets the same queries as **result caching**

- **Lesson learned:** Important to consider the interaction between the components

Locality

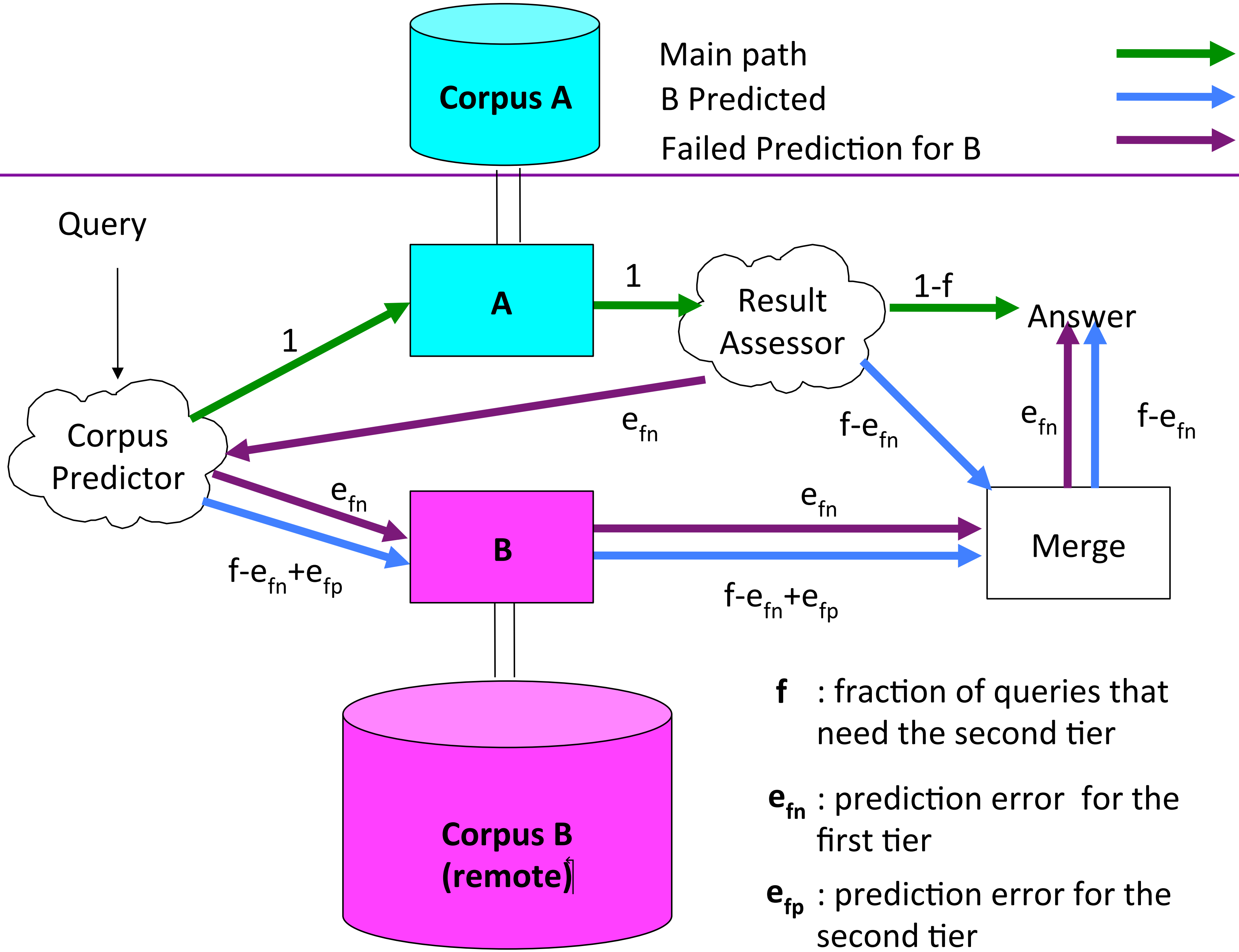
- Many queries are local
  - The answer returns only local documents
  - The user clicks only on local documents
- Locality also helps in:
  - Latency of HTTP requests (queries, crawlers)
  - Personalizing answers and ads
- Can we decrease the cost of the search engine?
- Measure of quality: same answers as centralized SE

Tier Prediction (Baeza-Yates et al, SIGIR 2009)

- Can we predict if the query is local?
  - Without looking at results or
  - increasing the extra load in the next level
- This is also useful in centralized search engines
  - Multiple tiers divided by quality
- Experimental results for
  - WT10G and UK/Chile collections

Motivation: Centralized Systems

- Traditionally partitioned corpora searched in serial, say two tiers
  - Second tier searched when first tier results are unsatisfactory
  - First tier faster and often sufficient
  - If second tier required, system is less efficient
- Better: search both corpora in parallel
- Best: predict which corpora to search



f : fraction of queries that need the second tier

$\mathbf{e_{fn}}$: prediction error for the first tier

$\mathbf{e_{fp}}$: prediction error for the second tier

Experimental Results

- Centralized case:

	Random	Centralized
Classifier Accuracy	0.714 $\pm$0.008	0.789$\pm$0.009
Precision	n/a	0.983$\pm$0.006
Recall	na	0.265$\pm$0.022

- Distributed case:

	Random	Distributed
Classifier Accuracy	0.539 $\pm$0.006	0.776$\pm$0.006
Precision	n/a	0.675$\pm$0.006
Recall	n/a	0.991$\pm$0.003

Trade-off Analysis (Baeza-Yates et al., 2008)

$$\begin{aligned} T_P &= T_S - (f - e_{FN}) t_A \\ &= T_{min} + e_{FN}\ t_A \end{aligned}$$

$$\Delta T = \frac{f - e_{FN}}{1 + f\ t_B / t_A} \quad \Delta C = \frac{e_{FP}}{f(1 + C_A / C_B)}$$

Is it worth it?

$$\frac{T_S}{T_P} > \frac{C_P}{C_S}$$

$$R_C = \frac{C_A}{C_B} \propto \frac{Size(A)}{Size(B)} \frac{t_B}{t_A} = \beta\ R_T$$

$$\beta > \frac{e_{FP}}{f - e_{FN}} \qquad e_{FN} < f - \frac{e_{FP}}{f + e_{FP}}$$

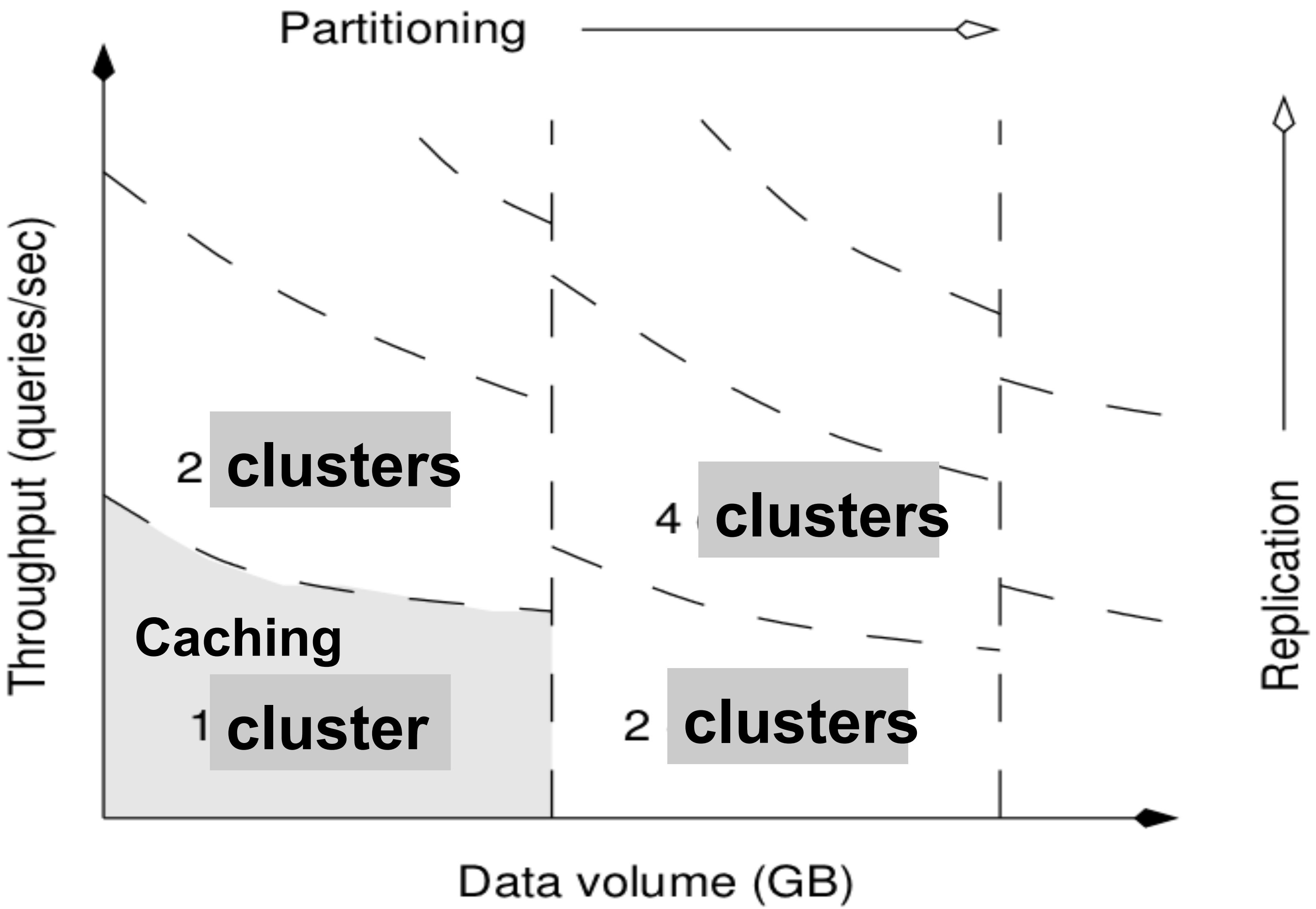
YAHOO!

Tier Prediction Example

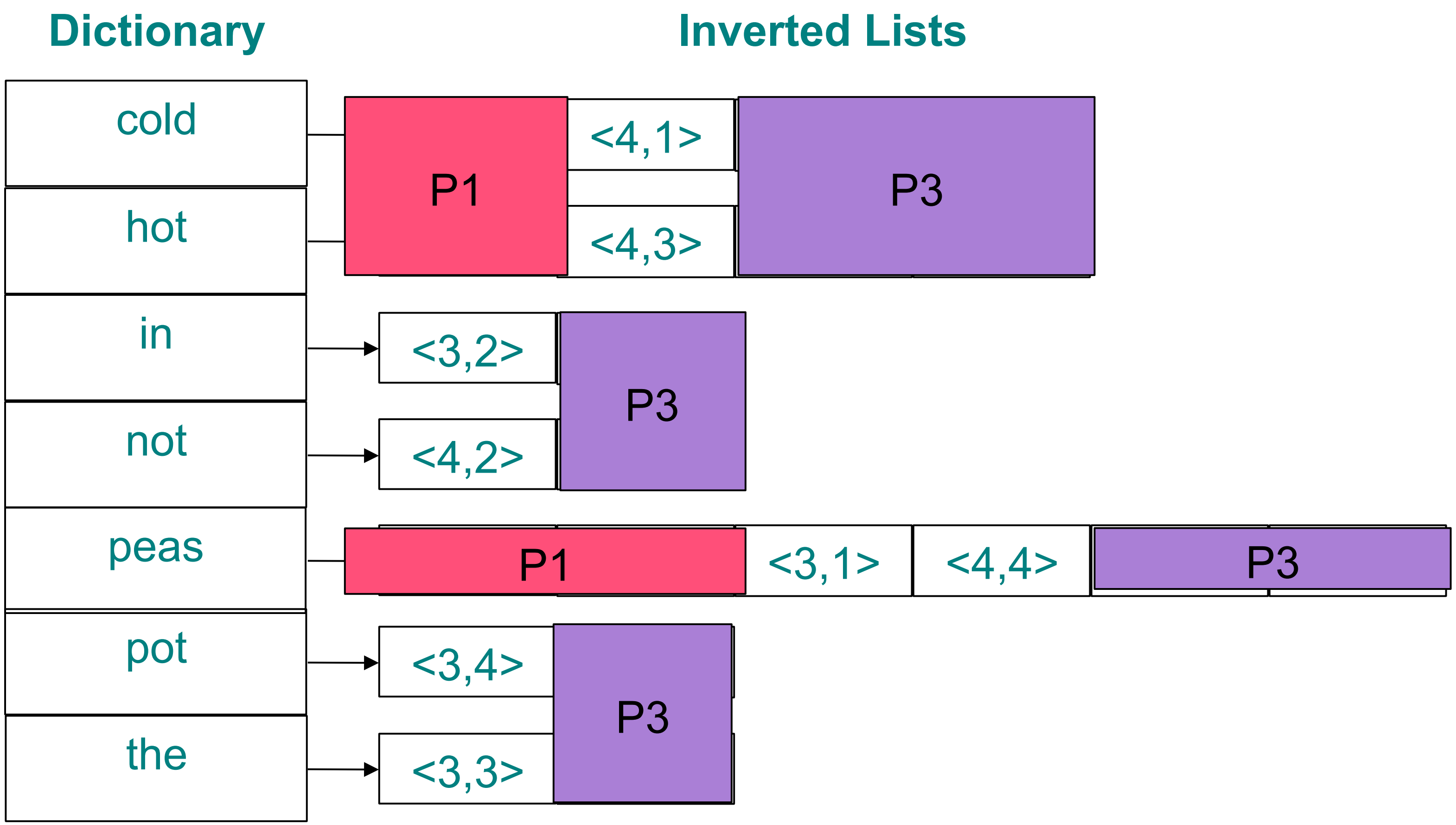
- Example:
  - System A is twice faster than System B
  - System B costs twice the costs of System A
- Centralized case:
  - 29% faster answer time at 20% extra cost
- Distributed case:
  - 15% faster answer time at 0.5% extra cost
- In both cases the trade-off is worth it

Scaling Up

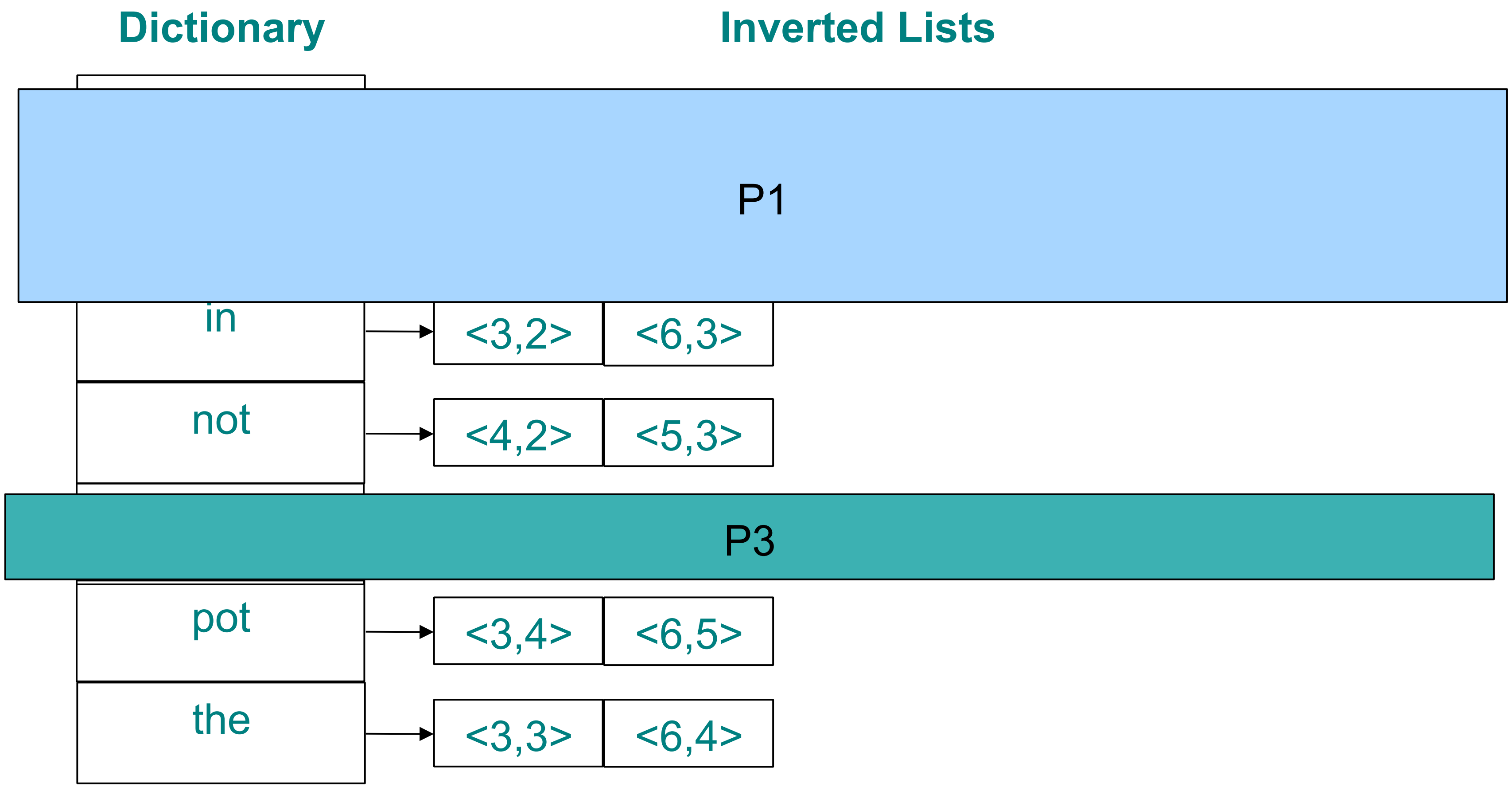
Adapted from Moffat and Zobel, 2004.



Document Partitioning



Term Partitioning

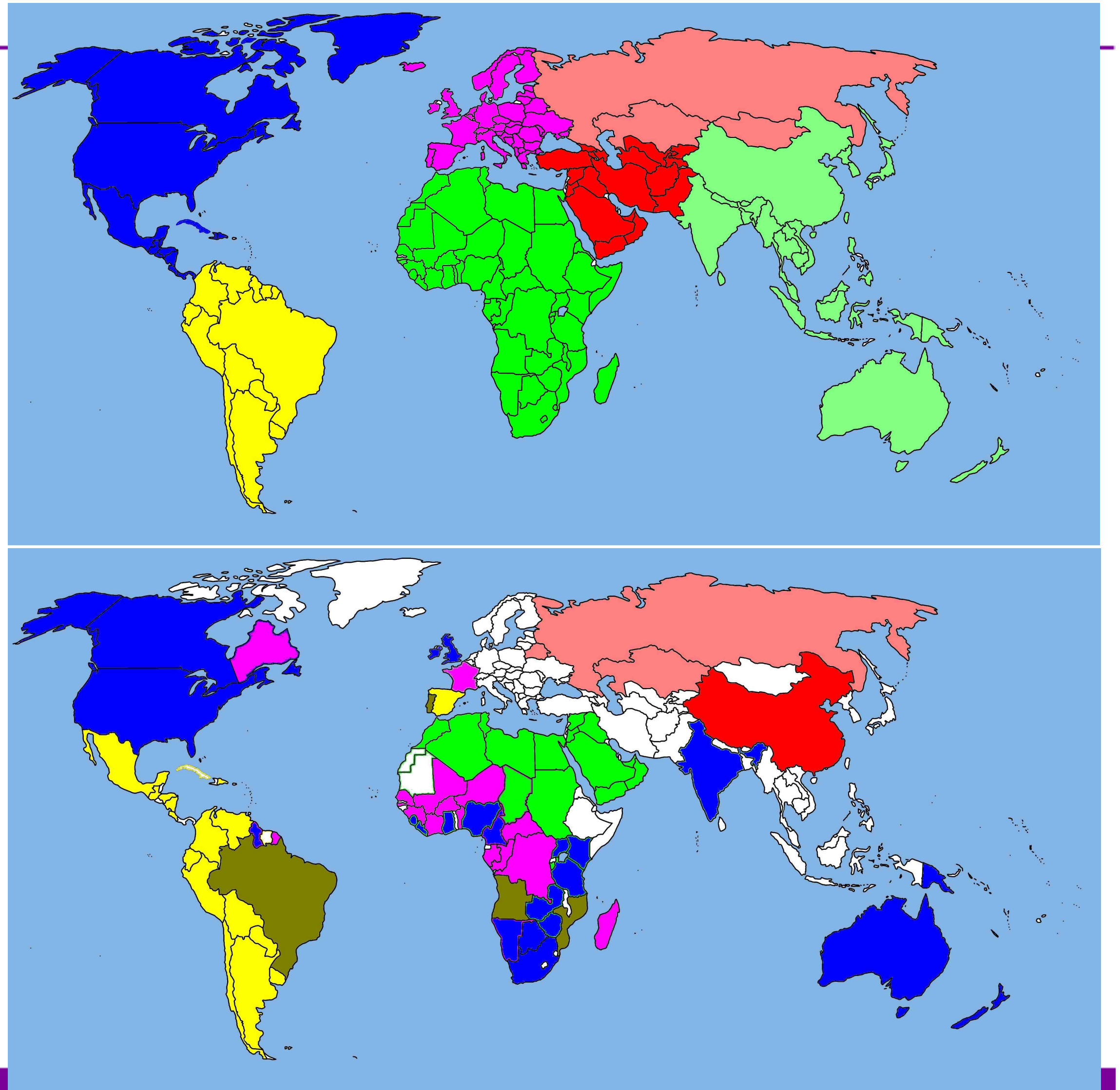


Index Partitioning: Comparison

• By documents	By terms
• Easy to partition	Random partition
• **Easier to build**	Hard to build
• No concurrency	Concurrent
• Perfect balance	Less balanced
• Less variance	Higher variance
• **Easier to maintain**	Harder to maintain

Index Partitioning: Practice

- **Within a cluster**
  - **term-based**
    - **performance**
  - **document-based**
    - **fault tolerance**
    - **load balance**
- **Across data centers**
  - **geographical**
  - **language-based**



Indexing

- The main **open** problem?
- Document partitioning is natural
- Mixing partitionings:
  - Improves search
  - Does not improve indexing
- More on collection selection?
  - Puppin *at al,* 2010

Master Site Selection

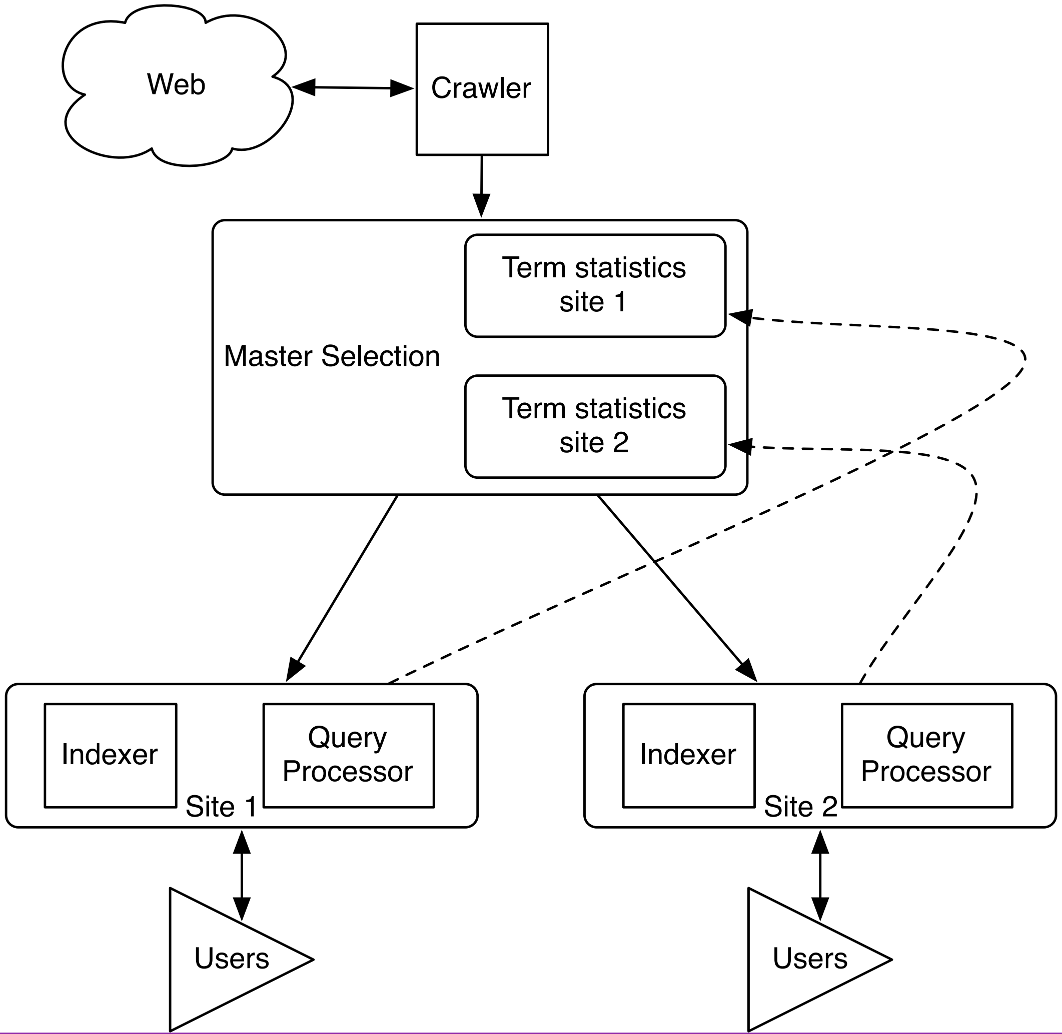
(Brefeld, Cambazoglu & Junqueira, WSDM 2011;
R. Blanco et al, CIKM 2011

- New documents
  - No search log yet
  - Assign master site
- Predict where document will be requested
  - Use evidence of user interest of each site
    - Language
    - Query terms distribution
    - Results cache invalidation

Terms Distribution

- Fine grain language/interest
- Compare terms in document with terms at each site
  - KL divergence
  - Dirichlet priors smoothing
- Sources of terms distribution
  - User queries
  - Documents in user results

Master Selection: Terms Distribution



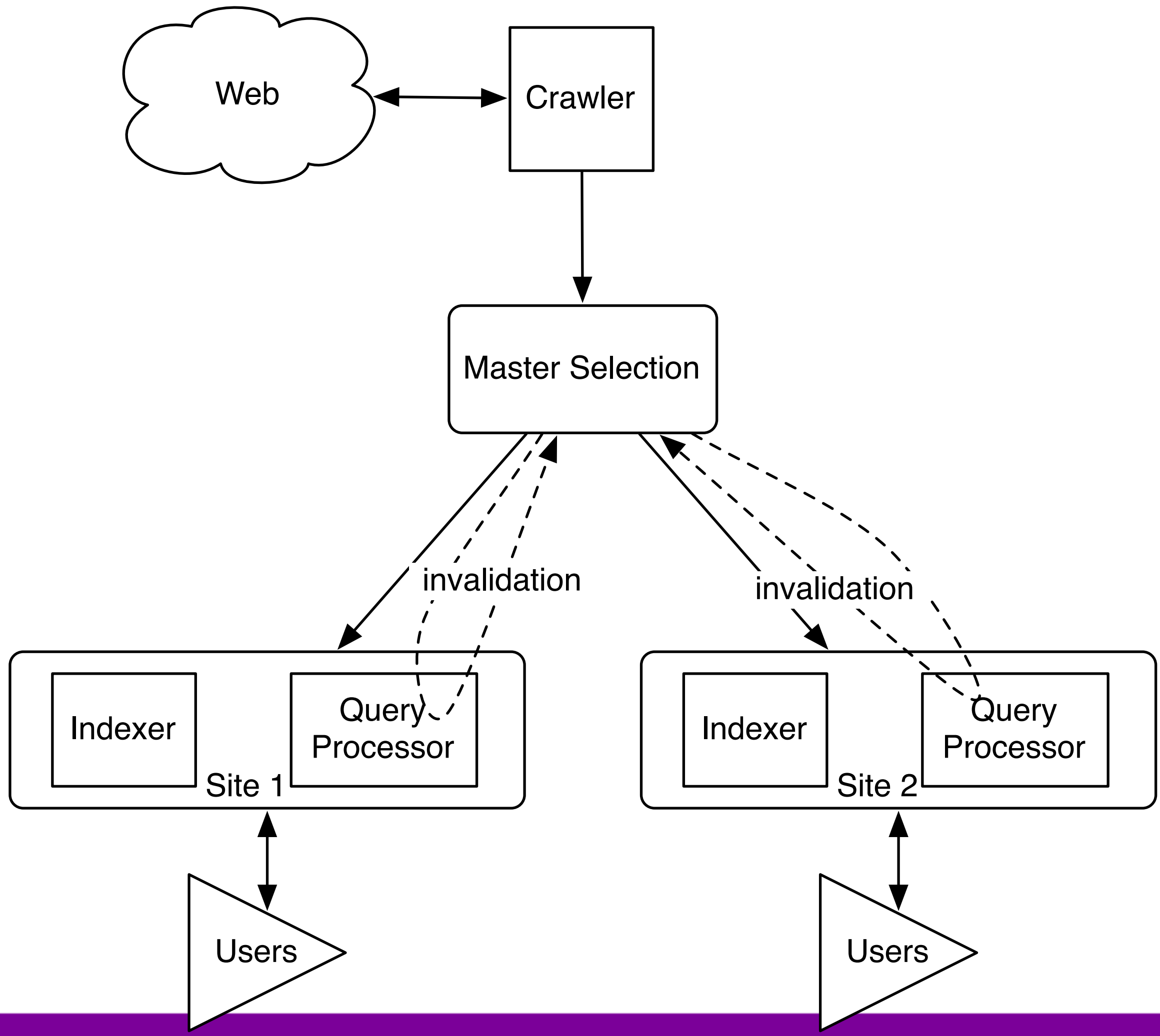
Cache Invalidation

- Search engine cache results
  - Less processing
  - Protect from activity spikes
- Incremental indexing
  - Better reactivity
  - Cache may serve stale results
- Cache invalidation algorithms

Cache Invalidation

- Number of invalidations to evaluate potential impact
  - Target top-k results directly
  - Preserve co-occurrences of terms
- Cost
  - Requires all document
  - Approximations available
  - Free if already in place
- Not all documents cause invalidation

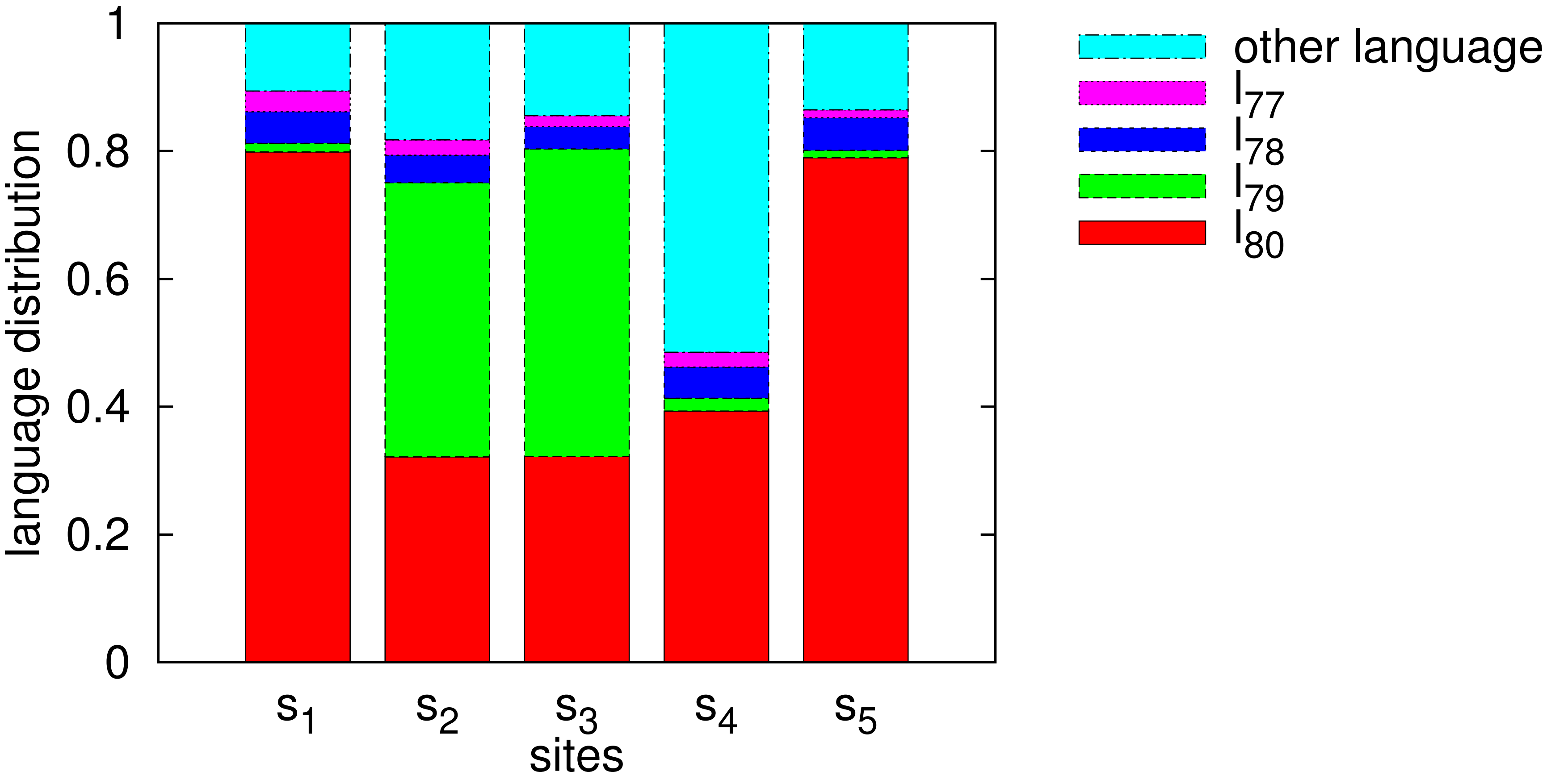
Master Selection: Invalidation



Experimental Setup

- 5 search sites
  - Non-trivial
- 32 millions Web pages
- 7 millions queries
- 2 sets
  - Training
  - Testing
- 3 algorithms

Language Based Assignment



Evaluation

Goal = provide local results

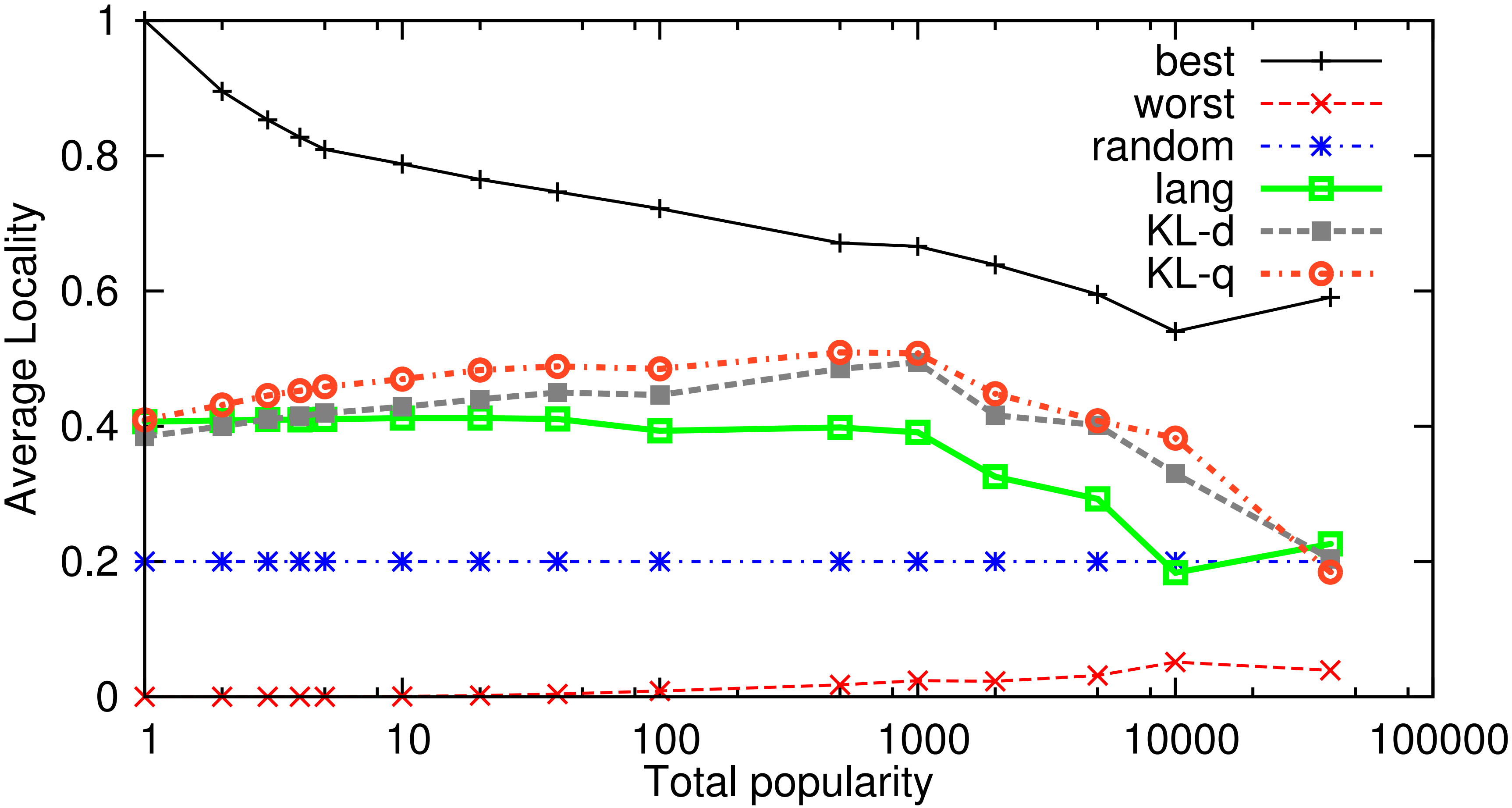
Metric = proportion of local results

1. Build knowledge using training documents/queries on centralized search engine
2. Label documents with search site using algorithm
3. Run test queries

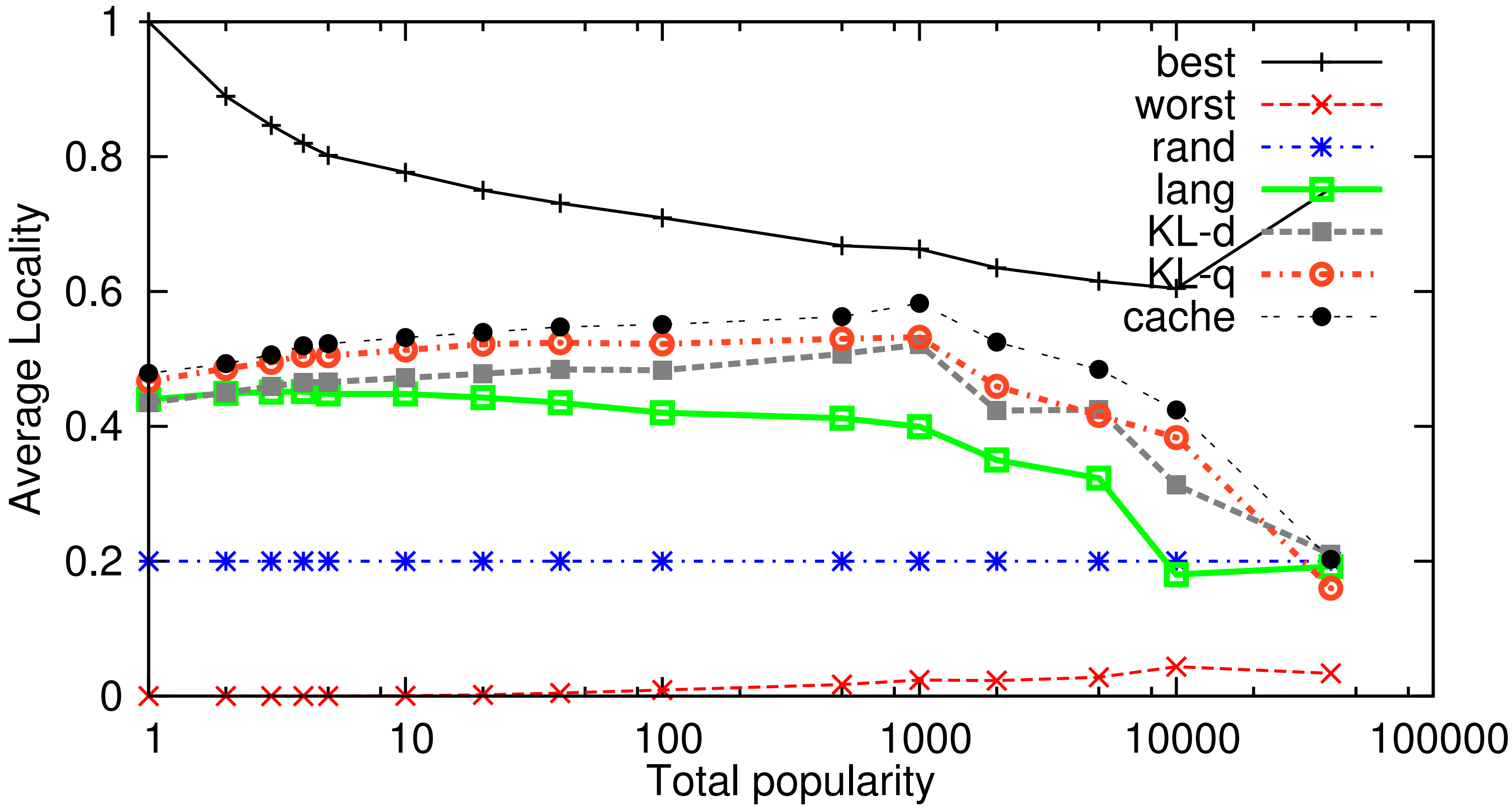
Evaluation

- 100% locality is unlikely
  - Some documents are accessed from different locations
  - Random gives 20% locality (5 search sites)
- Very popular documents are difficult
  - “Universal” success
- Unpopular documents are difficult
  - Low quality
  - Noise

Performance on all Documents



Performance with Invalidation



New Document Assignment

Significant improvement over language baseline

Stable enough to rely on master selection

- May add master migration in the future

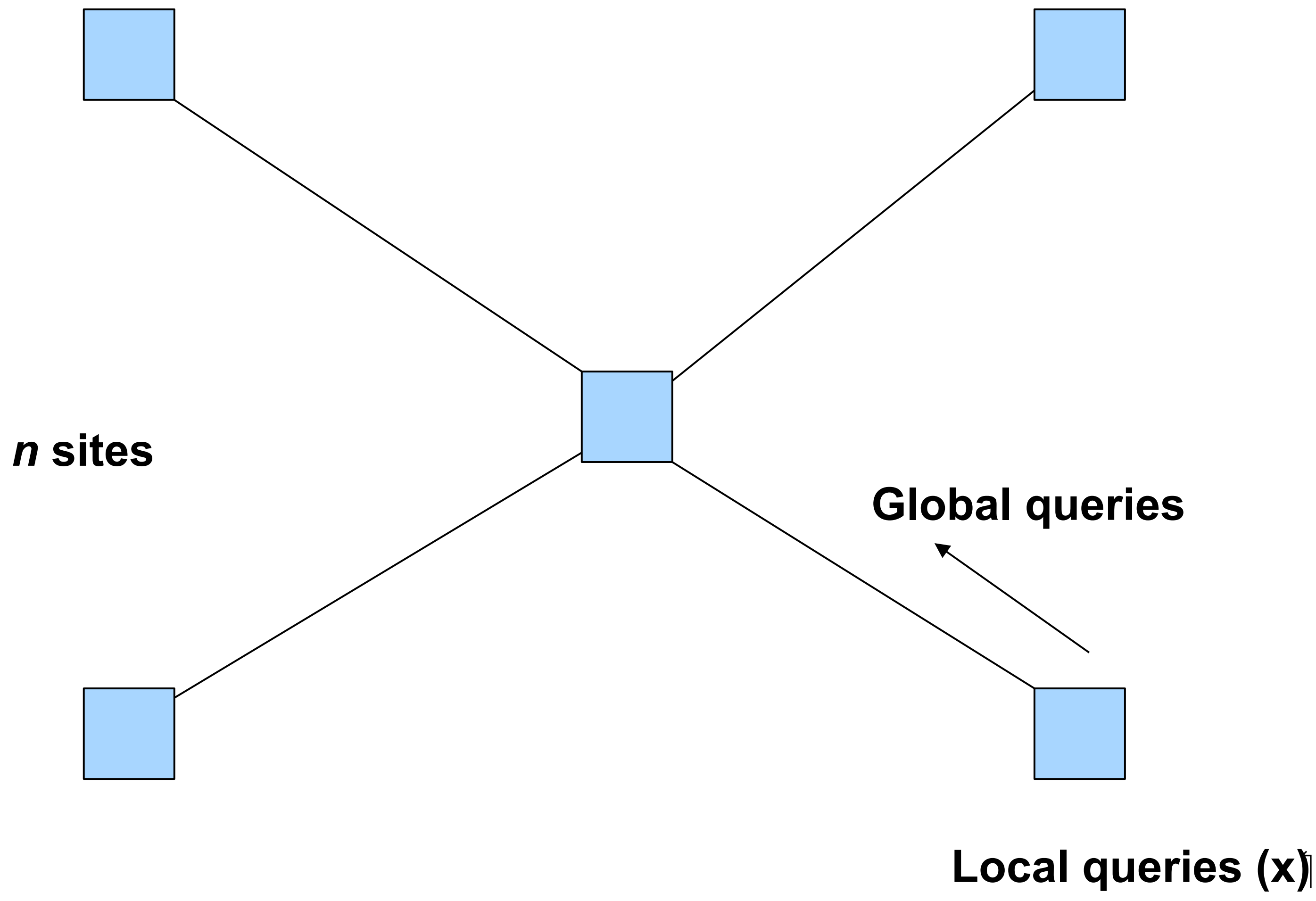
Master selection

- Ensures recall
- Avoid waste of indexing capacity

Need for a replication algorithm for popular documents

- Less forwarding
- Slightly increase processing of **ALL** queries

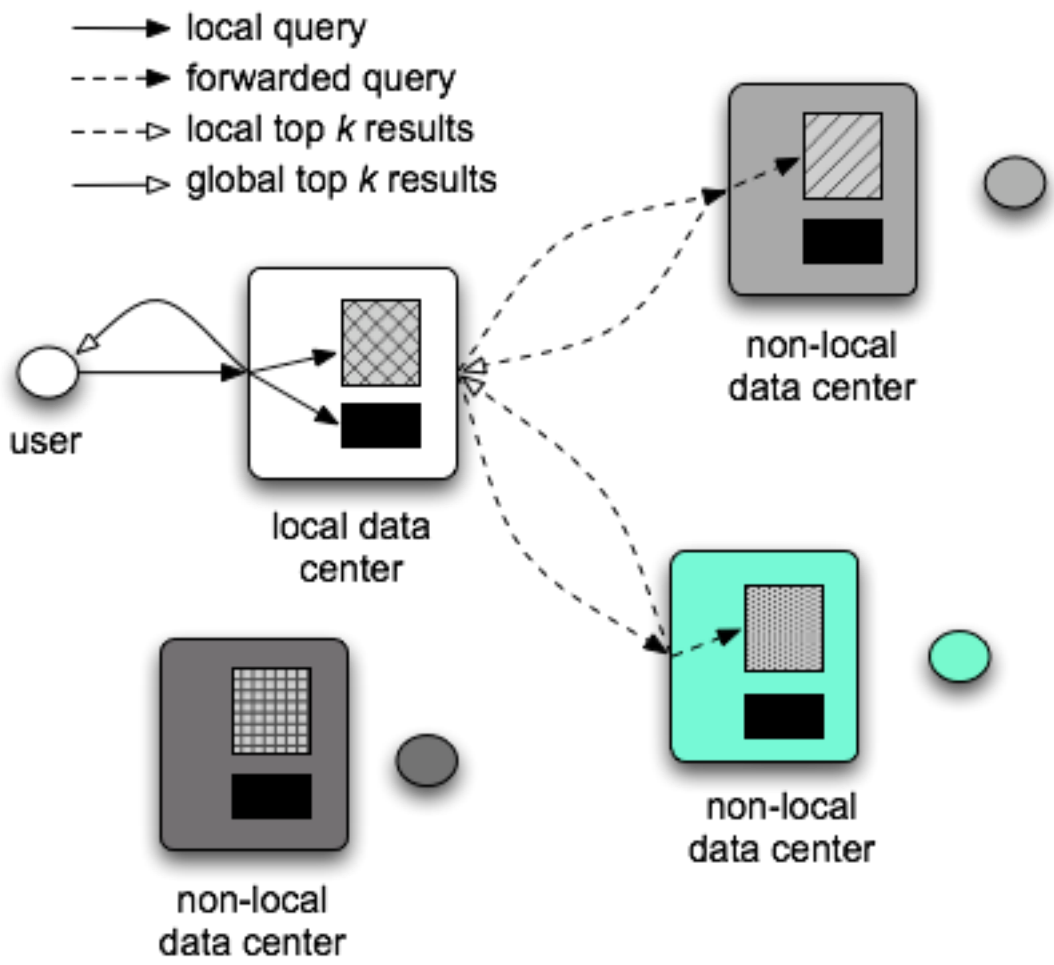
Star Topology (Baeza-Yates et al, CIKM 2009 Best paper award)



Multi-site Web Search Architecture

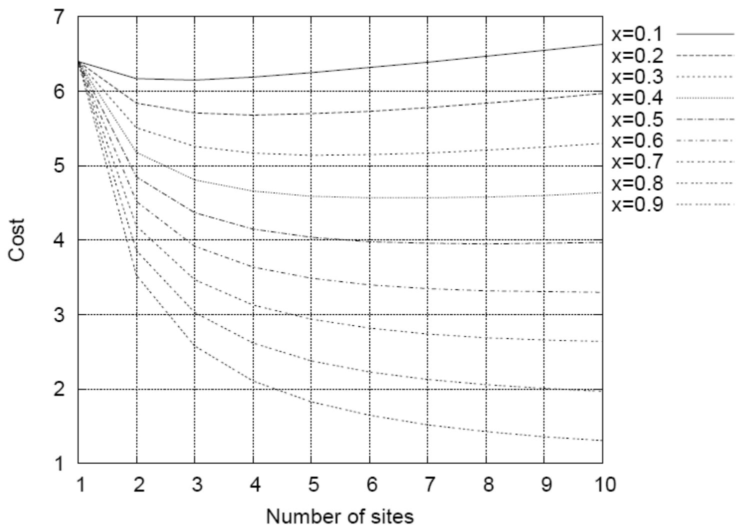
Key points

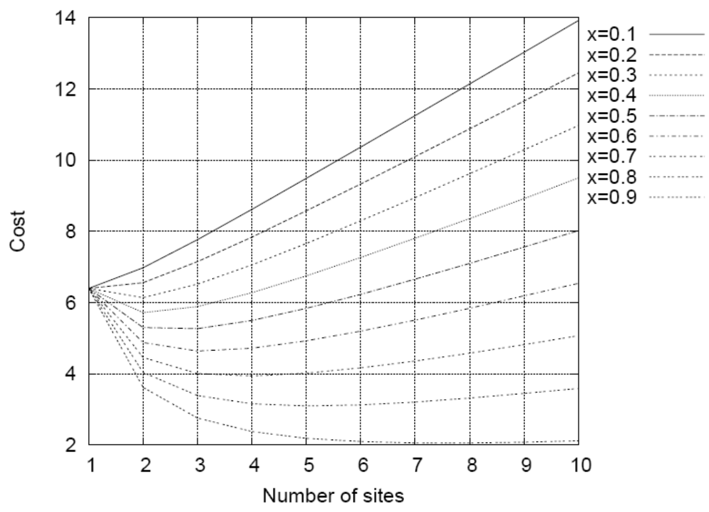
- multiple, regional data centers (sites)
- user-to-center assignment
- local web crawling
- partitioned web index
- partial document replication
- query processing with selective forwarding



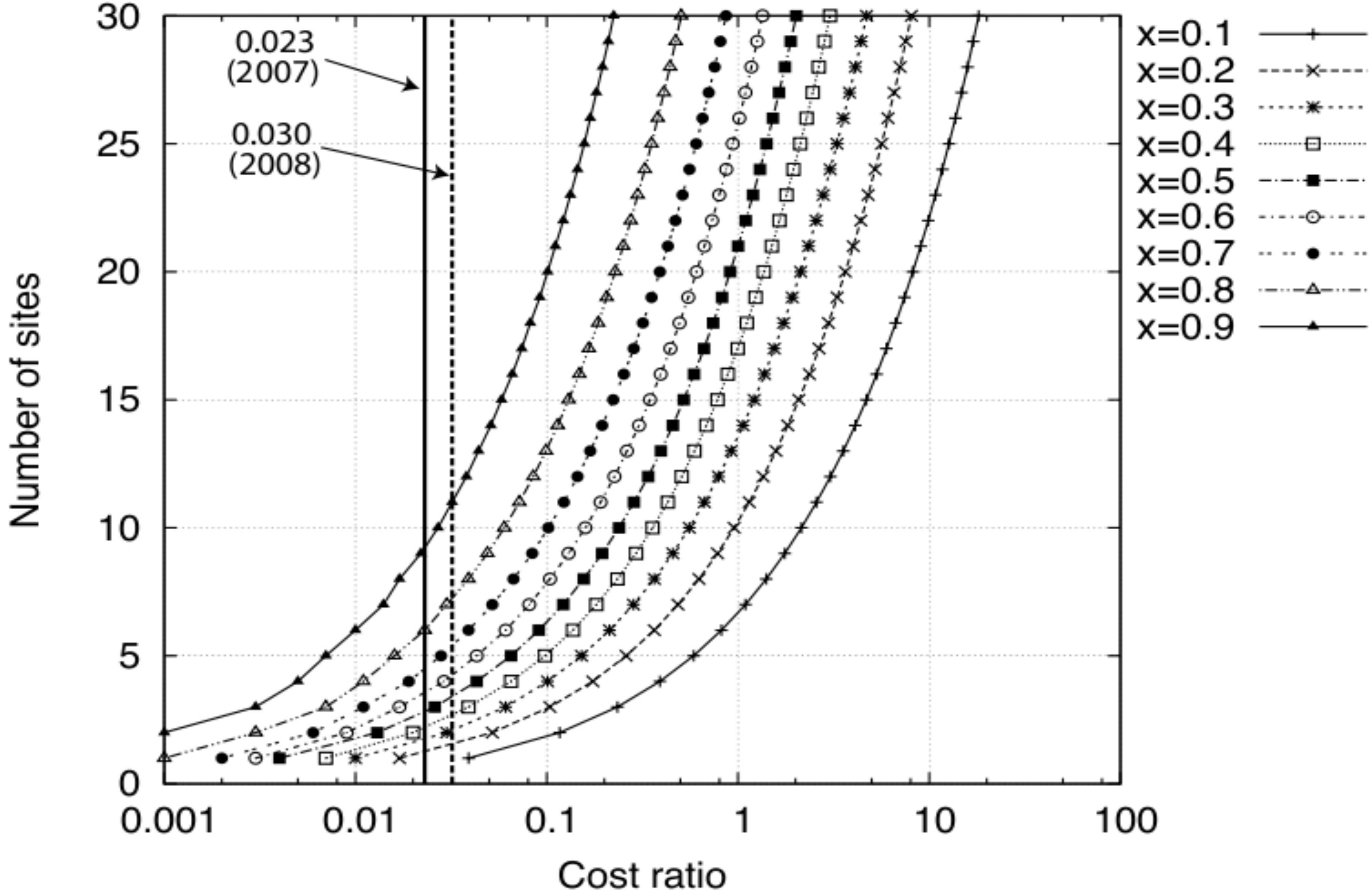
Cost Model

- Cost depends on **Initial cost**, **Cost of Ownership over time**, and **Bandwidth over time**.
- Cost of one QPS
  - *n* sites, *x* percentage of queries resolved locally, and relative cost of power and bandwidth 0.1 (left) and 1 (right)



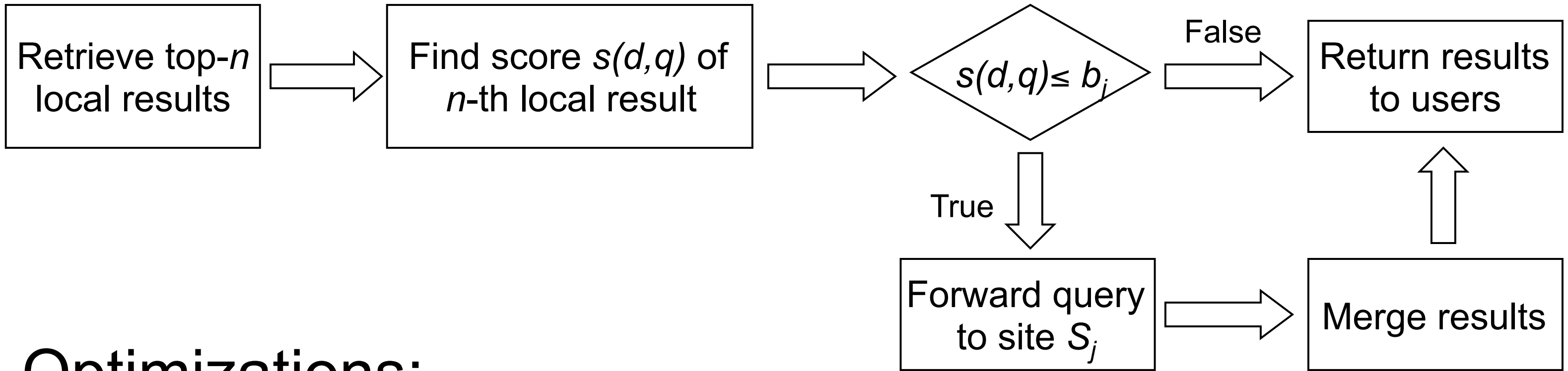


Optimal Number of Sites



Query Processing

- Site S_i knows the highest possible score b_j that site S_j can return for a query
  - Assume independent query terms
- Site S_i processes query q:

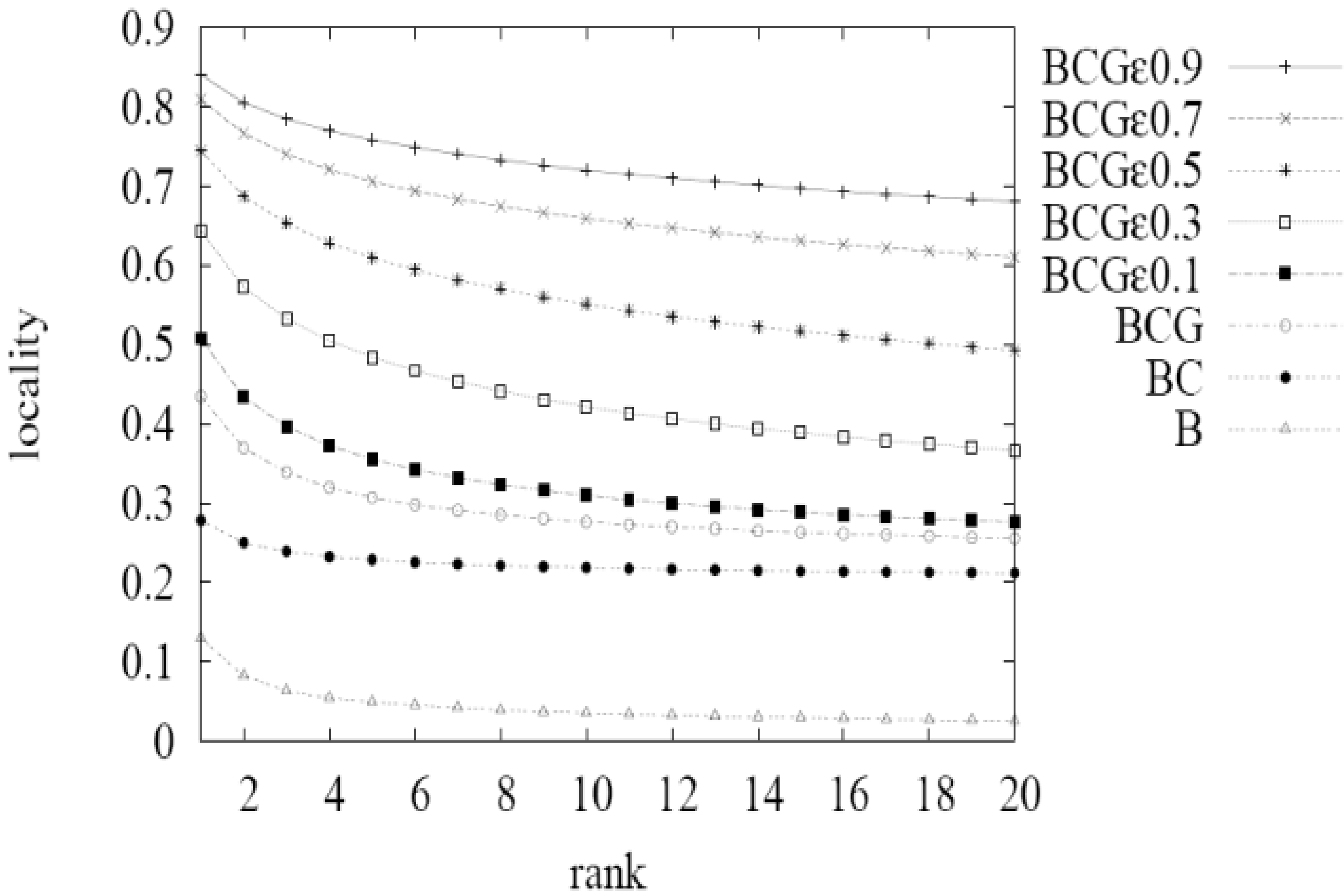


- Optimizations:
  - Caching
  - Replication of set *G* of most frequently retrieved documents
  - Slackness factor ε replacing b_j with $(1-\varepsilon)b_j$

Query Processing Results

- Locality at rank *n* for a search engine with 5 sites

- For what percentage of query volume, we can return top-*n* results locally



Cost Model Instantiation

- Assume a **5-site** distributed Web search engine in a **star topology**
- Optimal choice of central site S_x : site with **highest traffic** in our experiments
- Cost of distributed search engine relative to cost of centralized one

Query Processing	*Power Cost*	*Bandwidth Cost*	$\frac{\textit{Cost of distributed}}{\textit{Cost of centralized}}$
B	1.483	0.019	1.502
BC	1.278	0.016	1.294
BCG	1.156	0.013	1.169
BCG$\epsilon_{0.1}$	1.103	0.012	1.115
BCG$\epsilon_{0.3}$	0.970	0.010	**0.980**
BCG$\epsilon_{0.5}$	0.835	0.008	**0.843**
BCG$\epsilon_{0.7}$	0.719	0.006	**0.725**
BCG$\epsilon_{0.9}$	0.652	0.005	**0.657**

Improved Query Forwarding

(Cambazoglu et al, SIGIR 2010)

- Ranking algorithm
  - AND mode of query processing
  - the document score is computed simply summing query term weights (e.g., BM25)

- Query forwarding algorithm
  - a query should be forwarded to any site with potential to contribute at least one result to the global top *k*
  - we have the top scores for a set of off-line queries on all non-local sites

- Idea
  - set an upper bound on the possible top score of a query on non-local sites using the scores computed for off-line queries
  - decide whether a query should be forwarded to a site based on the comparison between the locally computed *k*-th score and the site's upper bound for the query

Thresholding Algorithm

- Notation
  - q: query
  - $\hat{S}$: local site
  - $\tilde{\mathcal{S}}$: set of non-local sites
  - $\tilde{S}$: a non-local site $\tilde{S} \in \tilde{\mathcal{S}}$
  - $s(q, k, S)$: score at rank k as computed by site S for query q
  - $m(q, S)$: an upper-bound for the score of q on site S
  - $f(q, \hat{S}, \tilde{\mathcal{S}}) \rightarrow \{0, 1\}$

Thresholding Algorithm

- Offline phase
  - obtain an offline query set $Q' = \{q'_1, \ldots, q'_m\}$ of m queries with each query $q'_i = \{t^i_1, \ldots, t^i_{n_i}\}$ composed of n_i distinct terms
  - precompute top score $s(q'_i, 1, \tilde{S})$ for every $q'_i \in Q'$ and $\tilde{S} \in \tilde{\mathcal{S}}$
  - replicate precomputed scores on all sites

- Online phase
  - given an online query $q = \{t_1, \ldots, t_n\}$ on local site $\hat{S}$
  - compute kth local score $s(q, k, \hat{S})$ on $\hat{S}$
  - compute $m(q, \tilde{S})$ values for all $\tilde{S} \in \tilde{\mathcal{S}}$ (as tight as possible)
  - decide on forwarding q to $\tilde{S}$ by comparing $s(q, k, \hat{S})$ against $m(q, \tilde{S})$

LP Formulation

- We introduce constraints

$x_j \geq 0, \ \forall t_j$ s.t. $t_j \in q$
$\sum_{t_j \in q'} x_j \leq s(q', 1, \tilde{S}), \ \forall q'$ s.t. $q' \in Q'$ and $q' \subset q$

- Given these constraints, the problem is to optimize

$m(q, \tilde{S}) = \max \sum_{t_j \in q} x_j$

- Query forwarding decision
  - if $\exists t_j, \forall q'$ s.t. $t_j \in q$, $t_j \notin q', q' \in Q'$, then $f(q, \hat{S}, \tilde{S}) = 1, \forall \tilde{S}$ s.t. $\tilde{S} \in \tilde{\mathcal{S}}$
  - $\forall \tilde{S} \in \tilde{\mathcal{S}}$:
    - if $\exists q'$ s.t. $q' \subset q, m(q, \tilde{S}) = 0$, then $f(q, \hat{S}, \tilde{S}) = 0$
    - if $m(q, \tilde{S}) \leq s(q, k, \hat{S})$, then $f(q, \hat{S}, \tilde{S}) = 0$
    - otherwise, $f(q, \hat{S}, \tilde{S}) = 1$

Offline Query Generation

- Offline query sets
  - D1: the vocabulary of the document collection
  - D2: all possible term pair combinations in the collection vocabulary
  - Q1: vocabulary of a train query log
  - Q2: term pairs in train queries
- Tested combinations
  - Q1
  - D1 (baseline: B-Y et al., CIKM’09) – 10% improvement
  - $Q1 \cup Q2$
  - $D1 \cup Q2$
  - $D1 \cup D2$
  - Oracle

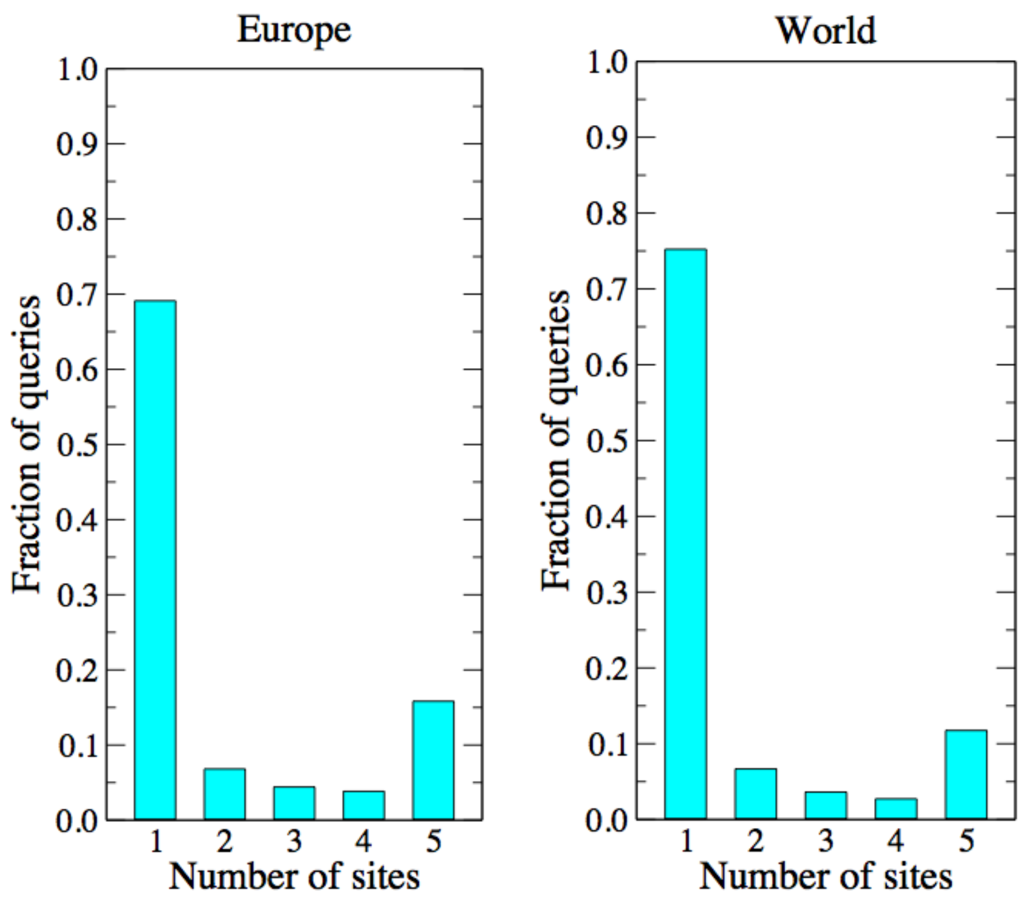
Experimental Setup

- Simulations via a very detailed simulator
- Data center locations
  - scenarios:
    - low latency (Europe): UK, Germany, France, Italy, Spain
    - high latency (World): Australia, Canada, Mexico, Germany, Brazil
  - assumed the data centers are located on capital cities
  - assumed that the queries are issued from the five largest city in the country
- Document collection
  - randomly sampled 200 million documents from a large Web crawl
  - a subset of them are assigned to a set of sites using a proprietary classifier
- Query log
  - consecutively sampled about 50 million queries from Yahoo! query logs
  - queries are assigned to sites according to the front-ends they are submitted to
  - first 3/4 of the queries is used for computing the thresholds; remaining 1/4 is used for evaluating performance

Locality of Queries

- Regional queries
  - most queries are regional
  - Europe: about 70% of queries appear on a single search site
  - World: about 75% of queries appear on a single search site

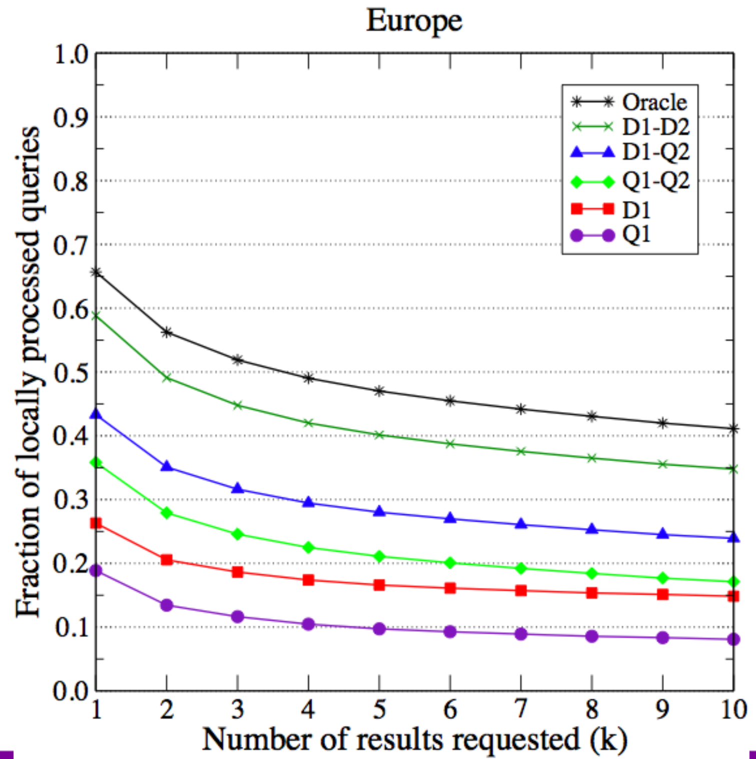
- Global queries
  - Europe: about 15% of queries appear on all five search sites
  - World: about 10% of queries appear on all five search sites

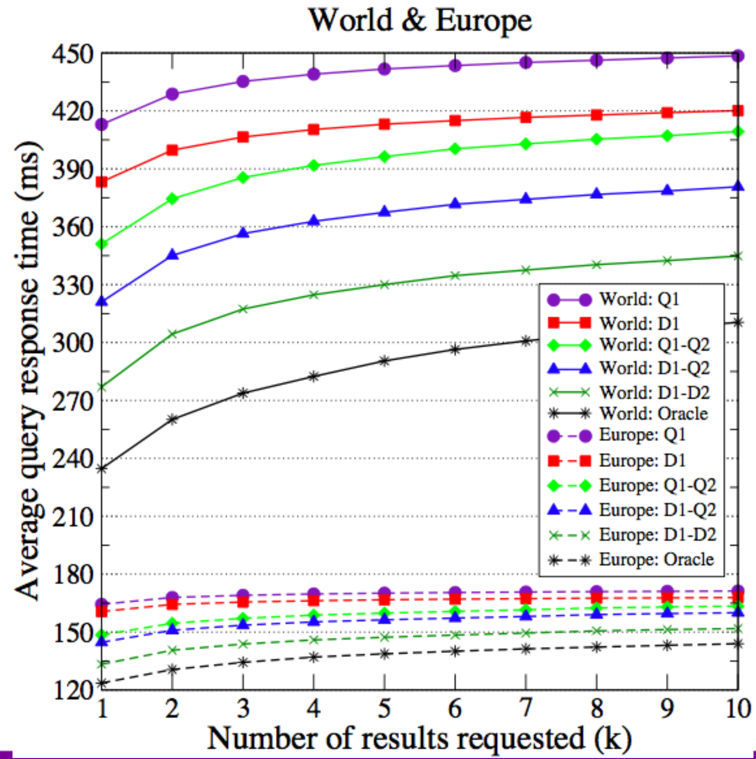


Performance of the Algorithm

- Local queries
  - about a quarter of queries can be processed locally (D1-Q2)
  - 10% increase over the baseline
  - oracle algorithm can achieve 40%

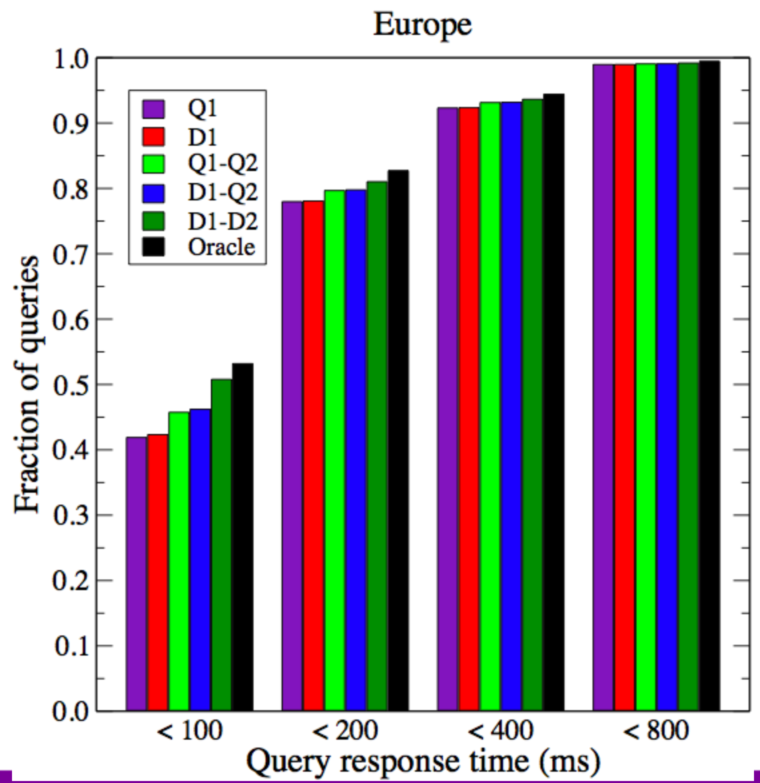
- Average query response times
  - Europe: between 120ms–180ms
  - World: between 240ms–450ms

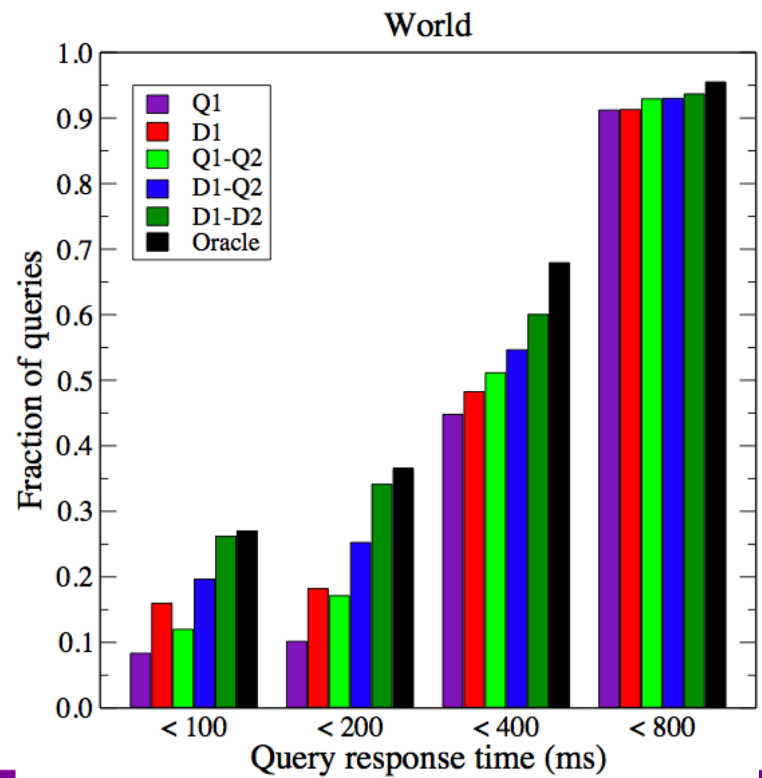




Performance of the Algorithm

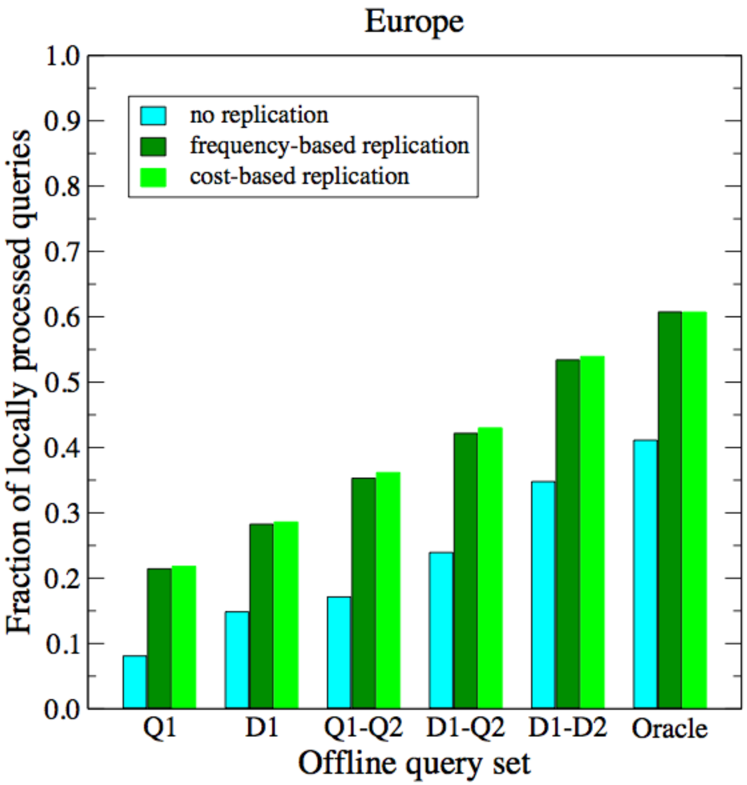
- Fraction of queries that are answered under a certain response time
  - Europe: around 95% under 400ms
  - World: between 45%–65% under 400ms



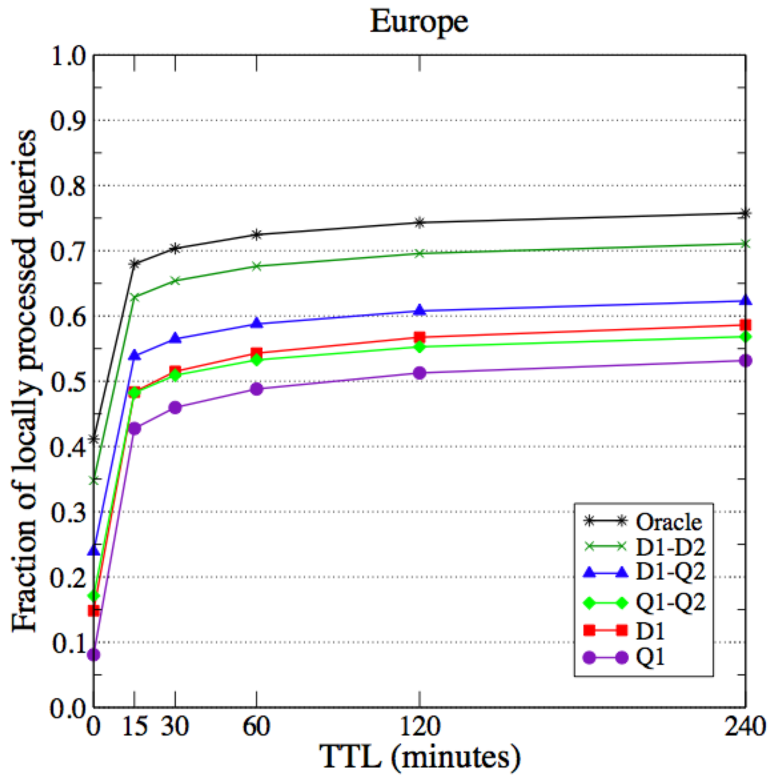


Partial Replication and Result Caching

- Replicate a small fraction of docs
  - prioritize by past access frequencies
  - prioritize by frequency/cost ratios



- Result cache
  - increase in local query rates: ~35%–45%
  - hit rates saturate quickly with increasing TTL



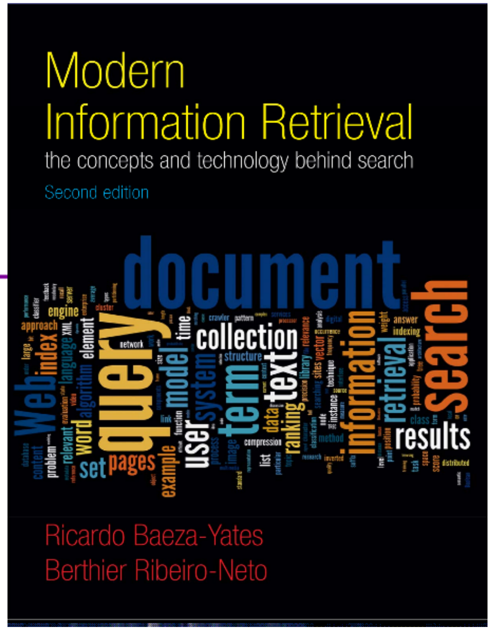
Further Optimizations

- Non-local top k optimization
  - request $k - r$ results, where r is the lowest rank such that $m(q, \tilde{S}) < s(q, r, \hat{S})$
  - 5%–17% saving in remote snippet computations
- Early query forwarding
  - q can be forwarded immediately if $m(q, \tilde{S}) > \sum_{t \in q} s(t, \lfloor (k-1)/|q| \rfloor + 1, \hat{S})$
  - precompute $s(t, \lfloor (k-1)/|q| \rfloor + 1, \hat{S})$
  - for single term queries, 20ms response time saving
- Early result presentation
  - show the user local results w/o waiting for remote results
  - top 10 from local site: 25%
  - top 1 from local site: 55%–60%
- Remote result presentation
  - result can be prepared remotely if only one non-local site is contacted
  - 10ms–15ms response time saving

Conclusions

- By using caching (mainly static) we can increase locality and we can predict when not to cache
- With enough locality we may have a cheaper search engine without penalizing the quality of the results or the response time
- We can predict when the next distributed level will be used to improve the response time without increasing too much the cost of the search engine
- We are currently exploring all these trade-off's

Thank you! Merci!



Second edition appeared in 2010

Questions?

rbaeza@acm.org

ACM WSDM 2011, February, Hong Kong
ACM SIGIR 2011, July, Beijing, China
SPIRE 2011, September, Pisa, Italy