(4) Crawling
Crawling

- NP-Hard Scheduling Problem
- Different goals
- Many Restrictions
- Difficult to define optimality
- No standard benchmark
Crawling Goals

- Quality
  - General Search Engine Crawlers
  - Focused and Personal Crawlers
  - Mirroring Systems
- Freshness
- Quantity
  - Research and Archive Crawlers
Bandwidth [bytes/second]

\[ P_1 = T^* \times B_1 \]

\[ P_2 = T^* \times B_2 \]

\[ P_3 = T^* \times B_3 \]

\[ P_4 = T^* \times B_4 \]

\[ P_5 = T^* \times B_5 \]
Software Architecture

- Single threaded Scheduler
- Database of URLs
- Collection of Text
- Multi threaded Crawler or Spider

World Wide Web
Basic Crawl Architecture

WWW

DNS

Fetch

Parse

Doc FP’s

Content seen?

robots filters

URL filter

Dup URL elim

URL Frontier

URL set
Queue of Web sites
(long-term scheduling)

Queue of Web pages for each site
(short-term scheduling)
Formal Problem

- Find a sequence of page requests \((p,t)\) that:
  - Optimizes a function of the volume, quality and freshness of the pages
  - Has a bounded crawling time
  - Fulfils politeness
  - Maximizes the use of local bandwidth

- Must be on-line: how much knowledge?
Crawling Heuristics

• Breadth-first
• Ranking-ordering
  – PageRank
• Largest Site-first
• Use of:
  – Partial information
  – Historical information
• No Benchmark for Evaluation
  – Use simulation
Comparing crawling algorithms

![Graph comparing crawling algorithms](image)

- Fraction of Pagerank collected vs Fraction of pages downloaded
- Very good (green line)
- Random (grey line)
- Very bad (red line)
No Historical Information

Baeza-Yates, Castillo, Marin & Rodriguez, WWW2005
Validation in the Greek domain

![Graph showing the validation in the Greek domain with different crawling strategies and their corresponding curves. The x-axis represents the day of crawling, while the y-axis represents the PR and number of documents retrieved.](image)
(5) Final Remarks
Young research field

- The Web is scientifically young
- The Web is intellectually diverse
- The technology mirrors the economic, legal and sociological reality
- Search is evolving to “task completion” and implicit search
- Plenty of open problems
Web Design and Search

Expected Needs

Design

Information Architecture

Findability

Ubiquity

Search Engine Person

Use

Fidelity

Usability

Visibility

Web usage mining
## Main Open Problems

**Usage data at a very large scale**

- over larger and larger populations
- over longer and longer periods of time

### Personalization

- More data via larger communities, makes data less personalized

  *wisdom of crowds does not work well on small corpora*

### Privacy

- Over personalization endangers privacy
  - Long-term logs endanger privacy

Large scale usage data is key BUT
• **Front-end and user experience**
  – The most probable reason for users to switch between quasi-equivalent engines is a better user experience

• **Depart from the rectangle/ranked list paradigm**
  – Get rid of queries? **Implicit search**
    • Content delivery is one flavor
    • But in general, why should we even have to formulate a query?
What’s next? Fourth generation:
From Information Retrieval to Information Supply

Explicit demand for information driven by a user query

Increase use of context

Active information supply driven by user activity and context
• Modern Information Retrieval
• Introduction to Information Retrieval
• Web Search: The Role of the Users.
  Tutorial at SIGIR 2011 and other conferences.
• Websites:
  – http://www.searchenginewatch.com/
  – http://www.searchengineshowdown.com/
• Main upcoming conferences:
  SPIRE 2011, October, Pisa, Italy
  WSDM 2012, February, Seattle, USA
  ECIR 2012, April, Barcelona, Spain
  ACM SIGIR 2012, July, Portland, USA
Distributed Web Search

Ricardo Baeza-Yates
Yahoo! Research
Barcelona, Spain & Santiago, Chile

ESSIR 2011, Koblenz, Germany
Agenda

• Challenges
• Crawling
• Indexing
• Caching
• Query Processing
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
Related Distributed Search Architectures

• Federated search
  – autonomous search sites
  – no explicit data partitioning
  – heterogeneous algorithms and resources
  – no dedicated network

• P2P search
  – high number of peers
  – dynamic and volatile peers
  – low cost systems
  – completely autonomous
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
Experimental Setup

• Network access statistics over the .edu domains
  – using a customized echoping version
  – over one week

• Eight crawled countries
  – US, Canada
  – Brazil, Chile
  – Spain, Portugal
  – Turkey, Greece

• Four crawling countries
  – US
  – Brazil
  – Spain
  – Turkey
Experimental Results
Experimental Results
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 $10i$% of all pages

- Crawling order: random
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
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
Indexing

- Distributed: the main open problem?
- Document partitioning is natural
- Mixing partitionings:
  - Improves search
  - Does not improve indexing
- More on collection selection?
  - Puppin et al, 2010
Query Processing: Pipelining

Term partitioning case, Moffat et al, 2007
Query Processing: Round Robin

Works for both partitionings
Marin et al, 2008
Caching basics

• A cache is characterized by its size and its eviction policy
  • *Hit*: requested item is already in the cache
  • *Miss*: requested item is not in the cache

• Caches speed up access to frequently or recently used data
  – Memory pages, disk, resources in LAN / WAN
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)
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
Caching at work

Query processing:

- Caching reduce **latency** and **load** on back-end servers
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

- \(Q_{TF}\) 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

- \(Q_{TFDF}\): 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:
  – $Q_{TF}$ : Cache terms with the highest query log frequency $f_q(t)$
  – $Q_{TFDF}$ : Cache terms with the highest ratio $f_q(t) / f_d(t)$

• Dynamic caching:
  – LRU, LFU
  – Dynamic $Q_{TFDF}$ : Evict the postings of the term with the lowest ratio $f_q(t) / f_d(t)$
Results
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
Parameter Estimation

- The average ratio between the time to return an answer computed from posting lists and from the query result cache is:
  - $TR_1$: when postings are in memory
  - $TR_2$: when postings are on disk

- $M$ is the cache size in answer units
  - A cache of query results stores $N_c = M$ queries

- $L$ is the average posting list size
  - A cache of postings stores $N_p = M/L = N_c/L$ posting lists
Parameter Values

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed Postings (L=0.75)</th>
<th>Compressed Postings (L’=0.26)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Centralized system</strong></td>
<td>$TR_1$</td>
<td>$TR_2$</td>
</tr>
<tr>
<td>Full evaluation</td>
<td>233</td>
<td>1760</td>
</tr>
<tr>
<td>Partial evaluation</td>
<td>99</td>
<td>1626</td>
</tr>
<tr>
<td><strong>WAN system</strong></td>
<td>$TR_1$</td>
<td>$TR_2$</td>
</tr>
<tr>
<td>Full evaluation</td>
<td>5001</td>
<td>6528</td>
</tr>
<tr>
<td>Partial evaluation</td>
<td>4867</td>
<td>6394</td>
</tr>
</tbody>
</table>
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
  - Similar query length in words and characters

- Yahoo! UK: 1 year
  - 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

<table>
<thead>
<tr>
<th>Cache size</th>
<th>Optimal</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AV (UK)</td>
<td>AV (UK)</td>
</tr>
<tr>
<td>50k</td>
<td>67.49 (32.46)</td>
<td>59.97 (17.58)</td>
</tr>
<tr>
<td>100k</td>
<td>69.23 (36.36)</td>
<td>62.24 (21.08)</td>
</tr>
<tr>
<td>250k</td>
<td>70.21 (41.34)</td>
<td>65.14 (26.65)</td>
</tr>
</tbody>
</table>
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

Altavista log

- Infinite
- LRU, 100k
- SDC, 100k
- AC, 100k

UK log

- Infinite
- LRU, 500k
- SDC, 500k
- AC, 500k

Hit ratio (%)

Frequency threshold
Results for Stateless features

- AC with stateless features outperforms LRU
- Stateless features offer high recall but low precision

<table>
<thead>
<tr>
<th></th>
<th>AV</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>72.32</td>
<td>51.78</td>
</tr>
<tr>
<td>Sizes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50k</td>
<td>61.43</td>
<td>61.88</td>
</tr>
<tr>
<td>100k</td>
<td>61.43</td>
<td>61.88</td>
</tr>
<tr>
<td>LRU</td>
<td>59.49</td>
<td>21.03</td>
</tr>
<tr>
<td>SDC</td>
<td>62.25</td>
<td>29.61</td>
</tr>
<tr>
<td>AC $k_c=10$</td>
<td>60.01</td>
<td>17.07</td>
</tr>
<tr>
<td>AC $k_c=20$</td>
<td>58.05</td>
<td>22.85</td>
</tr>
<tr>
<td>AC $k_c=30$</td>
<td>56.73</td>
<td>21.60</td>
</tr>
<tr>
<td>AC $k_c=40$</td>
<td>56.39</td>
<td>21.19</td>
</tr>
<tr>
<td>AC $k_w=2$</td>
<td>59.92</td>
<td>23.10</td>
</tr>
<tr>
<td>AC $k_w=3$</td>
<td>59.55</td>
<td>21.94</td>
</tr>
<tr>
<td>AC $k_w=4$</td>
<td>59.18</td>
<td>21.16</td>
</tr>
<tr>
<td>AC $k_w=5$</td>
<td>59.01</td>
<td>20.81</td>
</tr>
</tbody>
</table>
Index Pruning

Query processing:
3. from the pruned index

• Results caching and index pruning together
• … to reduce latency and load on back-end servers
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.

![Histogram showing the fraction of queries and misses by number of terms in a query.](image)
All queries vs. Misses:
Query result size distribution

- Randomly selected 2000 queries from all queries and misses:
- Avg. result size for misses is \(~100\) times smaller than for all queries
- Approx. half of the misses returns less than 5000 results – SMALL!
- Similar results with a “small” UK document collection (78M)
**All queries vs. Misses:**

**Term popularity distribution**

- Each point -> avg. popularity of **1000** consecutive terms
- Popularity is normalized by the size of the log
- The order of terms for *misses* is the same as for *all queries*
- Term popularity **does not** change much!

Log sizes: 185M – *all queries*, 41M - *misses*
**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

---

### Diagram

<table>
<thead>
<tr>
<th>Full index</th>
<th>Term pruning</th>
<th>Document pruning</th>
<th>T+D pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>t_1</td>
<td>t_1</td>
<td></td>
</tr>
<tr>
<td>t_2</td>
<td>t_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_3</td>
<td>t_3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_4</td>
<td>t_4</td>
<td>t_4</td>
<td></td>
</tr>
</tbody>
</table>

*Posting list*
Analysis of Results

- **Static index pruning**: addition to results caching, not replacement
  - **Term pruning** performs well for *misses* also
  - **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 and
  – 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
Trade-off Analysis (Baeza-Yates et al., 2008)

\[ T_P = T_S - (f - e_{FN})t_A \]
\[ = T_{\text{min}} + e_{FN} t_A \]

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

Is it worth it?

\[ T_S \frac{T_P}{T_P} > \frac{C_P}{C_S} \]

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

\[ \beta > \frac{e_{FP}}{f - e_{FN}} \]
\[ e_{FN} < f - \frac{e_{FP}}{f + e_{FP}} \]
## Experimental Results

### Centralized case:

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>Centralized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier Accuracy</td>
<td>0.714 ±0.008</td>
<td>0.789±0.009</td>
</tr>
<tr>
<td>Precision</td>
<td>n/a</td>
<td>0.983±0.006</td>
</tr>
<tr>
<td>Recall</td>
<td>na</td>
<td>0.265±0.022</td>
</tr>
</tbody>
</table>

### Distributed case:

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier Accuracy</td>
<td>0.539 ±0.006</td>
<td>0.776±0.006</td>
</tr>
<tr>
<td>Precision</td>
<td>n/a</td>
<td>0.675±0.006</td>
</tr>
<tr>
<td>Recall</td>
<td>n/a</td>
<td>0.991±0.003</td>
</tr>
</tbody>
</table>
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
Star Topology (Baeza-Yates et al, CIKM 2009 Best paper award)

$n$ sites

Global queries

Local queries (x)
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
A Search Engine Architecture with Partial Index Replication and Query Forwarding

- **Features**
  - several data centers
  - users are assigned to local data centers
  - documents
    - partitioned
    - partially replicated
  - queries
    - locally processed
    - forwarded on-demand

- **Parameters**
  - fraction of replicated index: $\beta$
  - fraction of queries forwarded: $\alpha$
  - avg. # of sites a query is forwarded: $\gamma$
  - local queries are processed over an index of size: $I (1 - \beta) / S + \beta$
  - remote $(\gamma \alpha)$ queries are processed over an index of size: $I (1 - \beta) / S$
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

![Graph showing the optimal number of sites versus cost ratio with different traces for various x values.](image-url)
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 $q$:

- **Retrieve top-$n$ local results**
- **Find score $s(d,q)$ of $n$-th local result**
- **Merge results (if True)**
  - **Forward query to site $S_j$**
- **Return results to users (if False)**

**Optimizations:**

- **Caching**
- **Replication of set $G$ of most frequently retrieved documents**
- **Slackness factor $\varepsilon$ 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

<table>
<thead>
<tr>
<th>Query Processing</th>
<th>Power Cost</th>
<th>Bandwidth Cost</th>
<th>Cost of distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1.483</td>
<td>0.019</td>
<td>1.502</td>
</tr>
<tr>
<td>BC</td>
<td>1.278</td>
<td>0.016</td>
<td>1.294</td>
</tr>
<tr>
<td>BCG</td>
<td>1.156</td>
<td>0.013</td>
<td>1.169</td>
</tr>
<tr>
<td>BCG(\epsilon_{0.1})</td>
<td>1.103</td>
<td>0.012</td>
<td>1.115</td>
</tr>
<tr>
<td>BCG(\epsilon_{0.3})</td>
<td>0.970</td>
<td>0.010</td>
<td>0.980</td>
</tr>
<tr>
<td>BCG(\epsilon_{0.5})</td>
<td>0.835</td>
<td>0.008</td>
<td>0.843</td>
</tr>
<tr>
<td>BCG(\epsilon_{0.7})</td>
<td>0.719</td>
<td>0.006</td>
<td>0.725</td>
</tr>
<tr>
<td>BCG(\epsilon_{0.9})</td>
<td>0.652</td>
<td>0.005</td>
<td>0.657</td>
</tr>
</tbody>
</table>
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
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
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!

Second edition appeared in 2011

Questions?
rbaeza@acm.org

SPIRE 2011, October, Pisa, Italy
WSDM 2012, February, Seattle, USA
ECIR 2012, April, Barcelona, Spain
ACM SIGIR 2012, July, Portland, USA