Scalability and Efficiency Challenges in Large-Scale Web Search Engines

Ricardo Baeza-Yates
B. Barla Cambazoglu
Yahoo Labs
Barcelona, Spain

Disclaimer

- This talk presents the opinions of the authors. It does not necessarily reflect the views of Yahoo Inc. or any other entity.
- Algorithms, techniques, features, etc. mentioned here might or might not be in use by Yahoo or any other company.
- Some non-technical material (e.g., images) provided in this presentation were taken from the Web.

Yahoo Labs Barcelona



- Research topics
  - web data mining
  - semantic search
  - social media
  - web retrieval
- Web retrieval
  - distributed web retrieval
  - scalability and efficiency
  - opinion/sentiment retrieval
  - personalization

Outline of the Tutorial

- Background (35 minutes)
- Main sections
  - web crawling (75 minutes + 5 minutes Q/A)
  - indexing (75 minutes + 5 minutes Q/A)
  - query processing (90 minutes + 5 minutes Q/A)
  - caching (40 minutes + 5 minutes Q/A)
- Concluding remarks (10 minutes)
- Questions and open discussion (15 minutes)

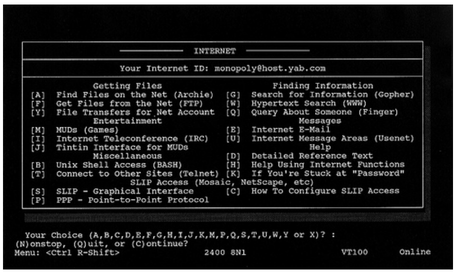
Structure of Main Sections

- Definitions
- Metrics
- Issues and techniques
  - single computer
  - cluster of computers
  - multiple search sites
- Research problems

Background

Brief History of Search Engines

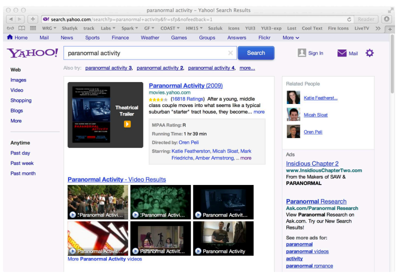
- Past
  - Before browsers
    - Gopher
  - Before the bubble
    - Altavista
    - Lycos
    - Infoseek
    - Excite
    - HotBot
  - After the bubble
    - Yahoo
    - Google
    - Microsoft



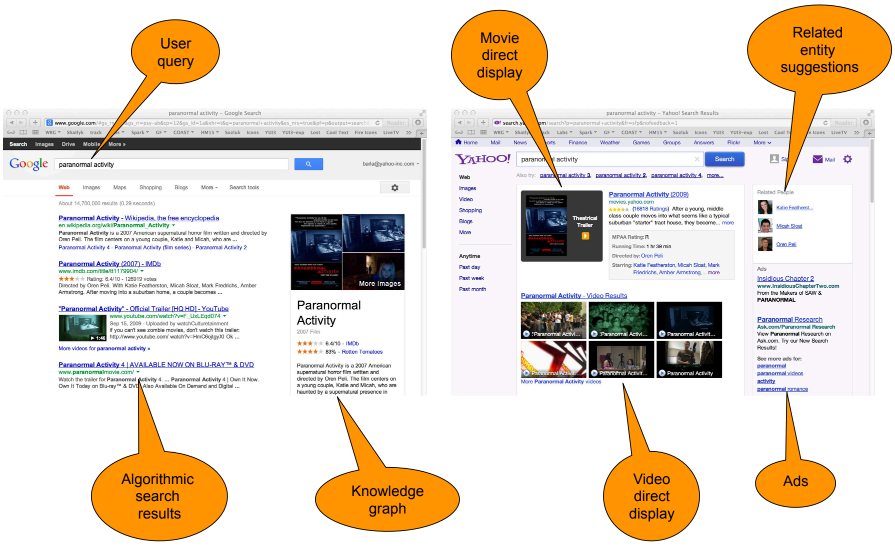
- Current
  - Global
    - Google, Bing
  - Regional
    - Yandex, Baidu
- Future
  - Facebook ?
  - …

Anatomy of a Search Engine Result Page

- Web search results ← Main focus of this tutorial
- Direct displays (vertical search results)
  - image
  - video
  - local
  - shopping
  - related entities
- Query suggestions
- Advertisements



Anatomy of a Search Engine Result Page



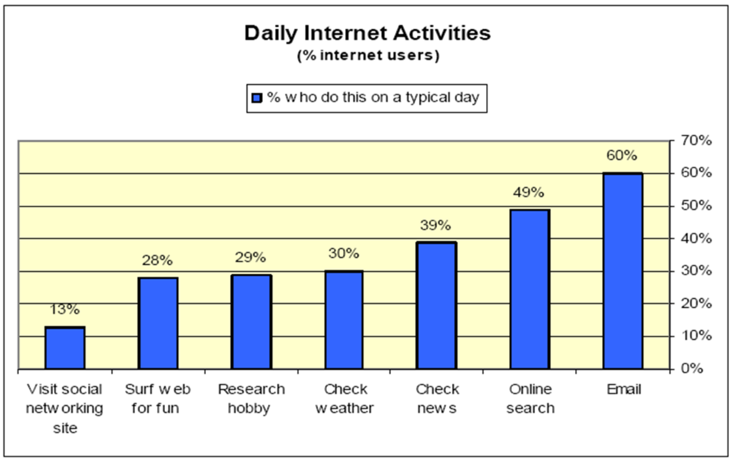
Actors in Web Search

- User's perspective: accessing information
  - relevance
  - speed

- Search engine's perspective: monetization
  - increase the ad revenue
  - attract more users
  - reduce the operational costs

- Advertiser's perspective: publicity
  - attract more users
  - pay little

Search Engine Usage



What Makes Web Search Difficult?

- Size
- Diversity
- Dynamicity

- All of these three features can be observed in
  - the Web
  - web users

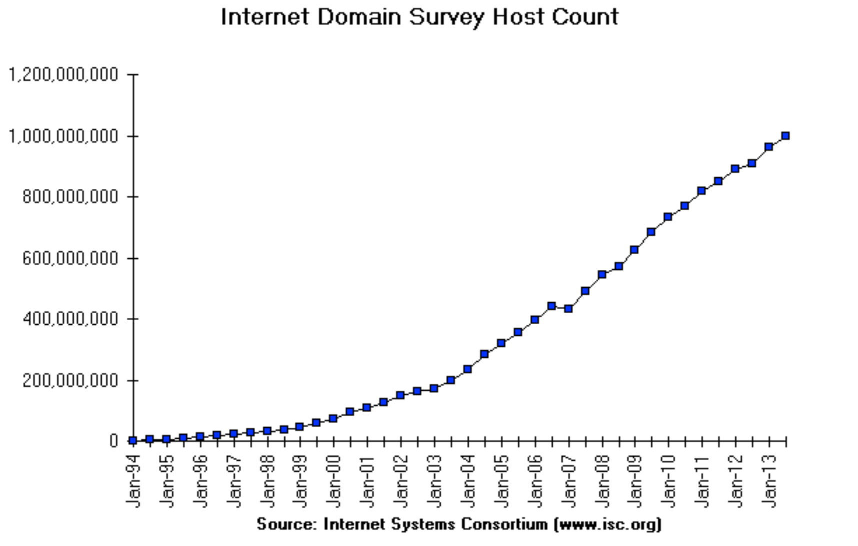
What Makes Web Search Difficult?

- The Web
  - more than 180 million Web servers and 950 million host names
    - compare with almost 1 billion computers directly connect to Internet
  - the largest data repository (estimated as 100 billion pages)
  - constantly changing
  - diverse in terms of content and data formats
- Users
  - too many! (over 2.5 billion at the end of 2012)
  - diverse in terms of their culture, education, and demographics
  - very short queries (hard to understand the intent)
  - changing information needs
  - little patience (few queries posed & few answers seen)

Expectations from a Search Engine

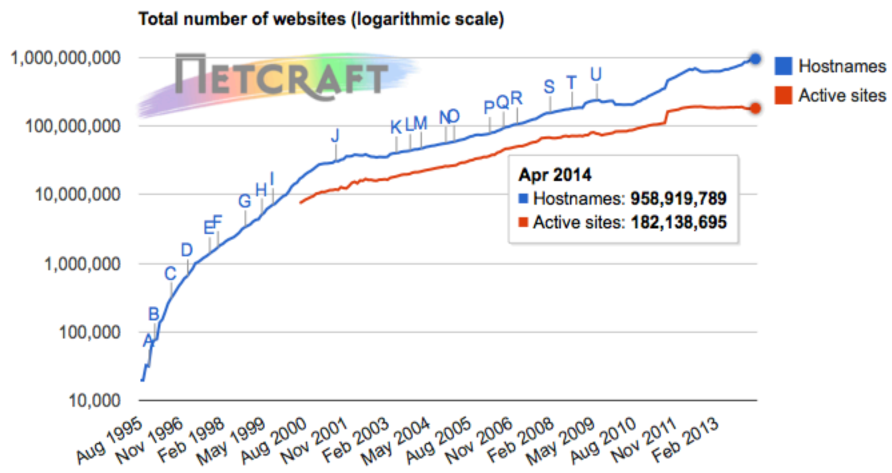
- Crawl and index a large fraction of the Web
- Maintain most recent copies of the content in the Web
- Scale to serve hundreds of millions of queries every day
- Evaluate a typical query under several hundred milliseconds
- Serve most relevant results for a user query

Internet Growth

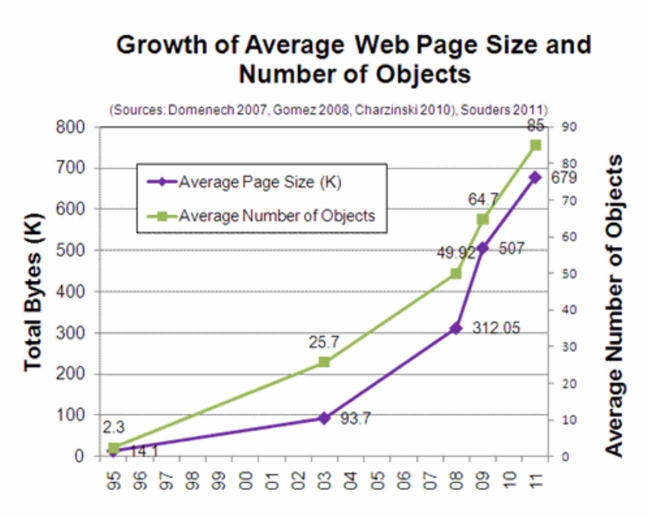


Source: VeriSign, January 2010

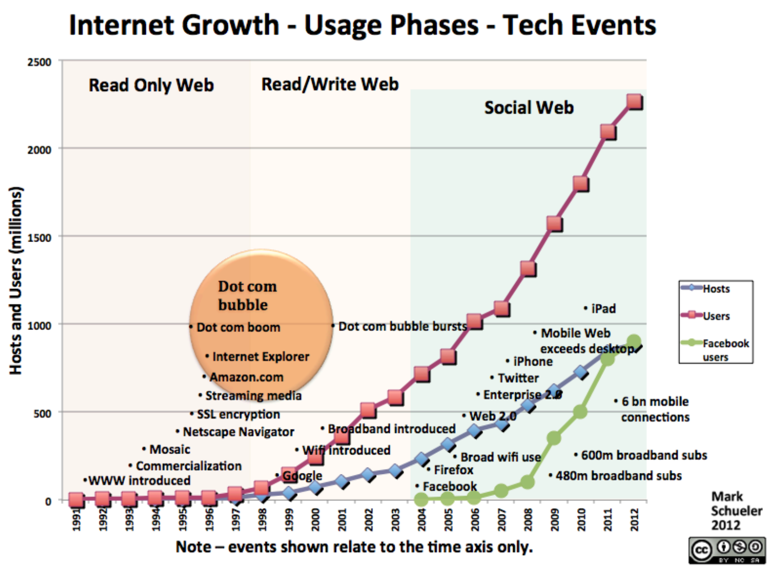
Web Growth



Web Page Size Growth



Web User Growth

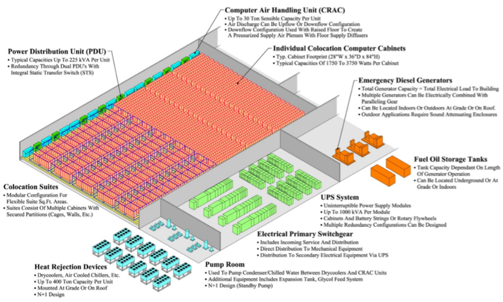


Search Data Centers

- Quality and performance requirements imply large amounts of compute resources, i.e., very large data centers
- High variation in data center sizes
  - hundreds of thousands of computers
  - a few computers

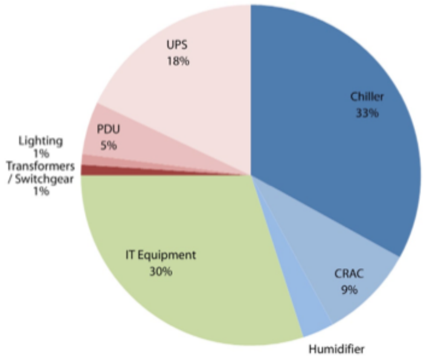
Cost of Data Centers

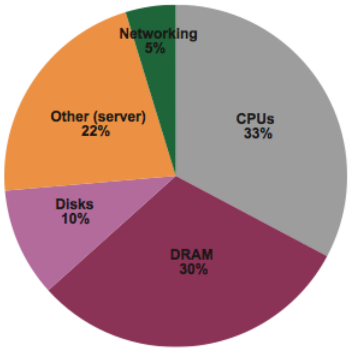
- Data center facilities are heavy consumers of energy, accounting for between 1.1% and 1.5% of the world's total energy use in 2010.



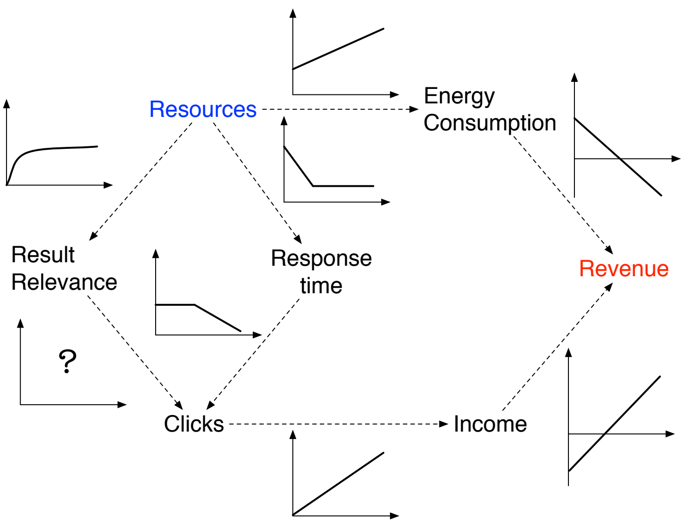
Financial Costs

- Costs
  - depreciation: old hardware need to be replaced
  - maintenance: failures need to be handled
  - operational: energy spending need to be reduced



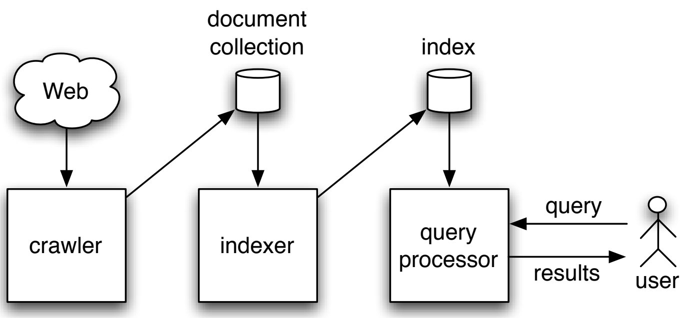


Impact of Resources on Revenue



Major Components in a Web Search Engine

- Web crawling
- Indexing
- Query processing



Q&A

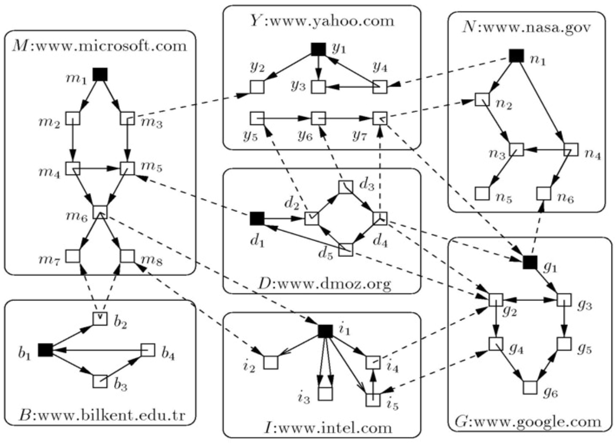
Web Crawling

Web Crawling

- Web crawling is the process of locating, fetching, and storing the pages available in the Web
- Computer programs that perform this task are referred to as
  - crawlers
  - spider
  - harvesters

Web Graph

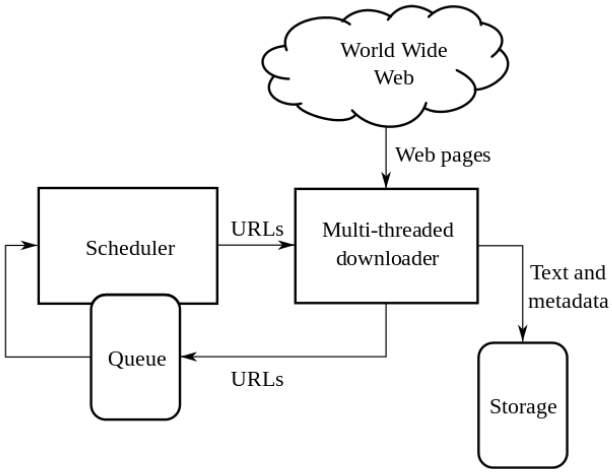
- Web crawlers exploit the hyperlink structure of the Web



Web Crawling Process

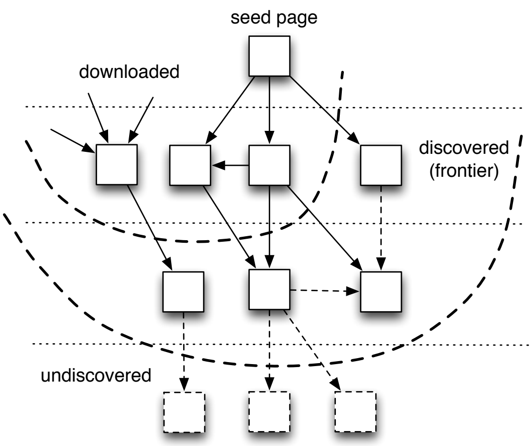
- A typical Web crawler
  - starts from a set of seed pages,
  - locates new pages by parsing the downloaded seed pages,
  - extracts the hyperlinks within,
  - stores the extracted links in a fetch queue for retrieval,
  - continues downloading until the fetch queue gets empty or a satisfactory number of pages are downloaded.

Web Crawling Process



Web Crawling

- Crawling divides the Web into three sets
  - downloaded
  - discovered
  - undiscovered

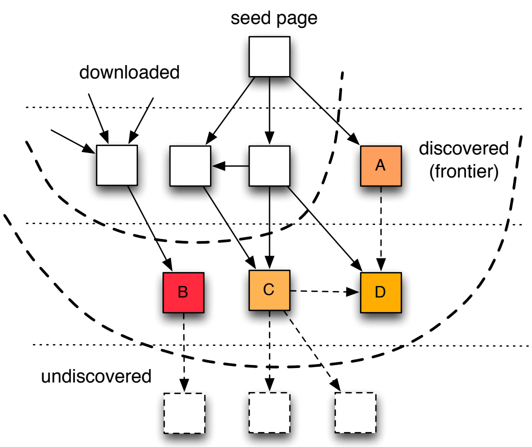


URL Prioritization

- A state-of-the-art web crawler maintains two separate queues for prioritizing the download of URLs
  - discovery queue
    - downloads pages pointed by already discovered links
    - tries to increase the coverage of the crawler
  - refreshing queue
    - re-downloads already downloaded pages
    - tries to increase the freshness of the repository

URL Prioritization (Discovery)

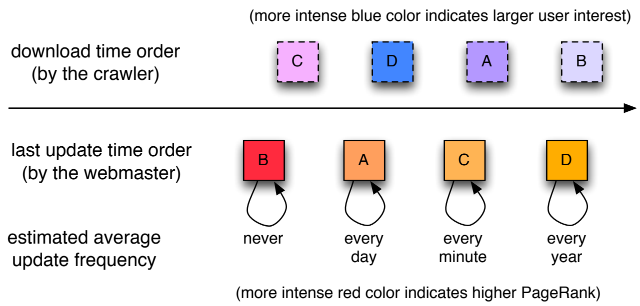
- Random (A, B, C, D)
- Breadth-first (A)
- In-degree (C)
- PageRank (B)



(more intense red color indicates higher PageRank)

URL Prioritization (Refreshing)

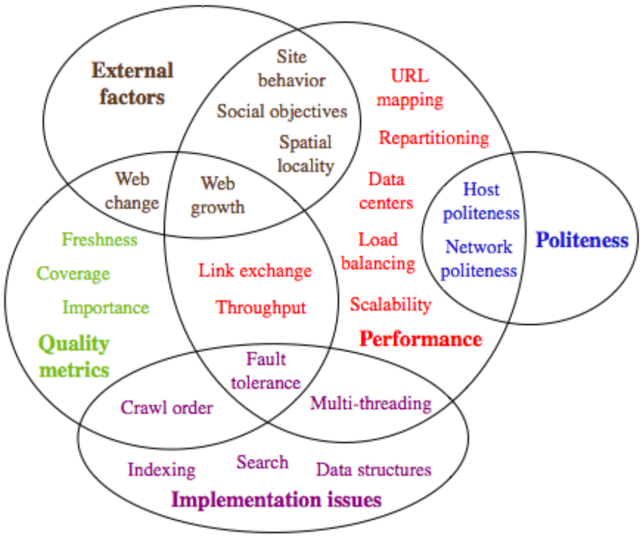
- Random (A, B, C, D)
- PageRank (B)
- Age (C)
- User feedback (D)
- Longevity (A)



Metrics

- Quality metrics
  - coverage: the percentage of the Web discovered or downloaded by the crawler
  - freshness: measure of out-datedness of the local copy of a page relative to the page’s original copy on the Web
  - page importance: percentage of important or popular pages in the repository
- Performance metrics
  - throughput: content download rate in bytes per unit of time

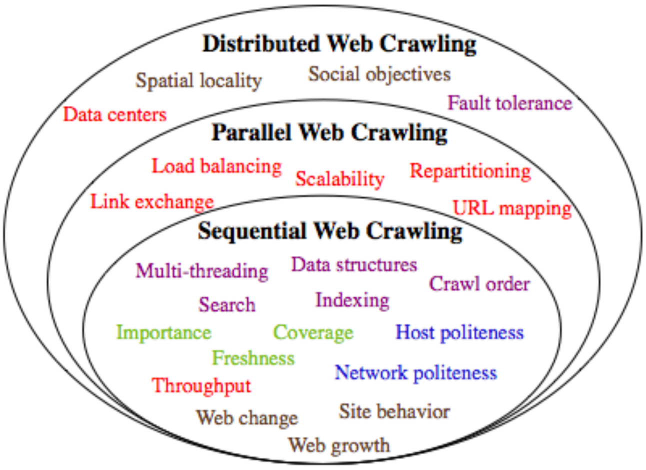
Concepts Related to Web Crawling



Crawling Architectures

- Single computer
  - CPU, RAM, and disk becomes a bottleneck
  - not scalable
- Parallel
  - multiple computers, single data center
  - scalable
- Geographically distributed
  - multiple computers, multiple data centers
  - scalable
  - reduces the network latency

Crawling Architectures



Issues in Web Crawling

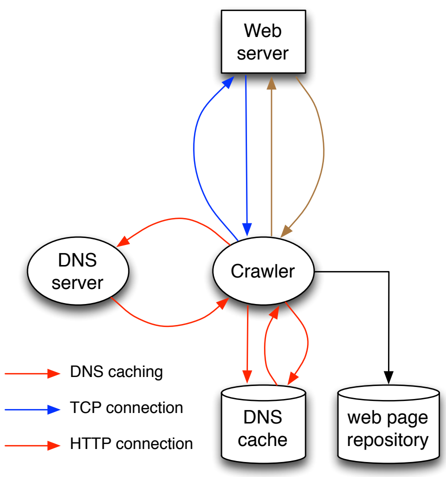
- Dynamics of the Web
  - Web growth
  - content change
- Malicious intent
  - hostile sites (e.g., spider traps, infinite domain name generators)
  - spam sites (e.g., link farms)

Issues in Web Crawling

- URL normalization (a.k.a. canonicalization)
  - case-folding
  - removing leading "www strings (e.g., www.cnn.com → cnn.com)
  - adding trailing slashes (e.g., cnn.com/a → cnn.com/a/)
  - relative paths (e.g., ../index.html)
- Web site properties
  - sites with restricted content (e.g., robot exclusion),
  - unstable sites (e.g., variable host performance, unreliable networks)
  - politeness requirements

DNS Caching

- Before a web page is crawled, the host name needs to be resolved to an IP address
- Since the same host name appears many times, DNS entries are locally cached by the crawler



Multi-threaded Crawling

- Multi-threaded crawling
  - crawling is a network-bound task
  - crawlers employ multiple threads to crawl different web pages simultaneously, increasing their throughput significantly
  - in practice, a single node can run around up to a hundred crawling threads
  - multi-threading becomes infeasible when the number of threads is very large due to the overhead of context switching

Politeness

- Multi-threading leads to politeness issues
- If not well-coordinated, the crawler may issue too many download requests at the same time, overloading
  - a web server
  - an entire sub-network
- A polite crawler
  - puts a delay between two consecutive downloads from the same server (a commonly used delay is 20 seconds)
  - closes the established TCP-IP connection after the web page is downloaded from the server

Robot Exclusion Protocol

- A standard from the early days of the Web
- A file (called robots.txt) in a web site advising web crawlers about which parts of the site are accessible
- Crawlers often cache robots.txt files for efficiency purposes

```
User-agent: googlebot          # all services
Disallow: /private/            # disallow this directory

User-agent: googlebot-news     # only the news service
Disallow: /                    # on everything

User-agent: *                  # all robots
Disallow: /something/          # on this directory

User-agent: *                  # all robots
Crawl-delay: 10                # wait at least 10 seconds

Disallow: /directory1/              # disallow this directory
Allow: /directory1/myfile.html      # allow a subdirectory

Host: www.example.com          # use this mirror
```

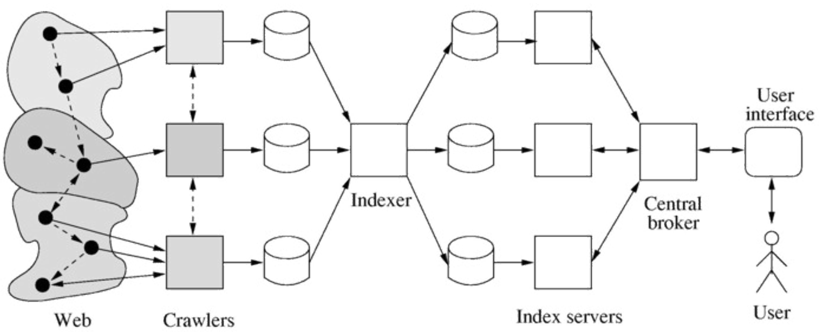
Mirror Sites

- A mirror site is a replica of an existing site, used to reduce the network traffic or improve the availability of the original site
- Mirror sites lead to redundant crawling and, in turn, reduced discovery rate and coverage for the crawler
- Mirror sites can be detected by analyzing
  - URL similarity
  - link structure
  - content similarity

Data Structures

- Good implementation of data structures is crucial for the efficiency of a web crawler
- The most critical data structure is the “seen URL” table
  - stores all URLs discovered so far and continuously grows as new URLs are discovered
  - consulted before each URL is added to the discovery queue
  - has high space requirements (mostly stored on the disk)
    - URLs are stored as MD5 hashes
    - frequent/recent URLs are cached in memory

Parallel Web Crawling

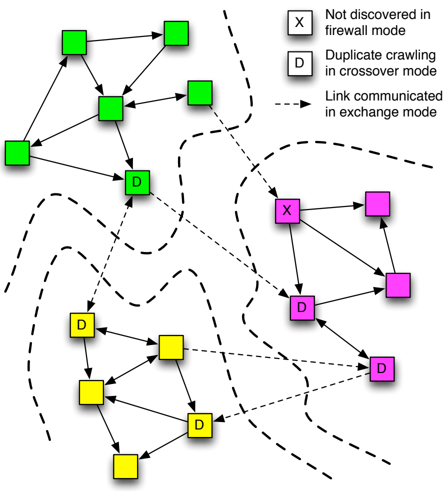


Web Partitioning and Fault Tolerance

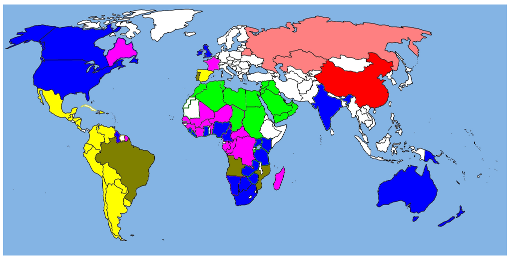
- Web partitioning
  - Typically based on the MD5 hashes of URLs or host names
  - site-based partitioning is preferable because URL-based partitioning may lead to politeness issues if the crawling decisions given by individual nodes are not coordinated
- Fault tolerance
  - when a crawling node dies, its URLs are partitioned over the remaining nodes

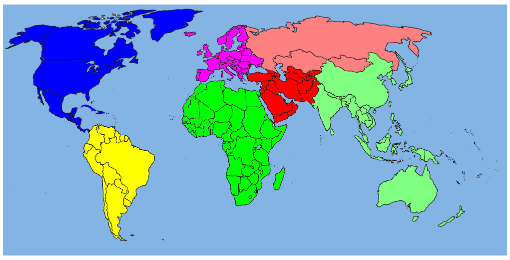
Parallelization Alternatives

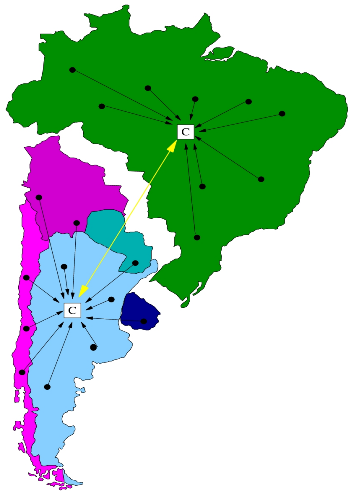
- Firewall mode
  - lower coverage
- Crossover mode
  - duplicate pages
- Exchange mode
  - communication overhead



Geographically Distributed Web Crawling







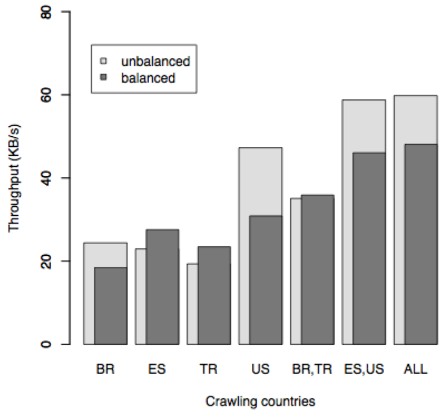
Geographically Distributed Web Crawling

- Benefits
  - higher crawling throughput
    - geographical proximity
    - lower crawling latency
  - improved network politeness
    - less overhead on routers because of fewer hops
  - resilience to network partitions
    - better coverage
  - increased availability
    - continuity of business
  - better coupling with distributed indexing/search
    - reduced data migration

Geographically Distributed Web Crawling

- Four crawling countries
  - US
  - Brazil
  - Spain
  - Turkey

- Eight target countries
  - US, Canada
  - Brazil, Chile
  - Spain, Portugal
  - Turkey, Greece

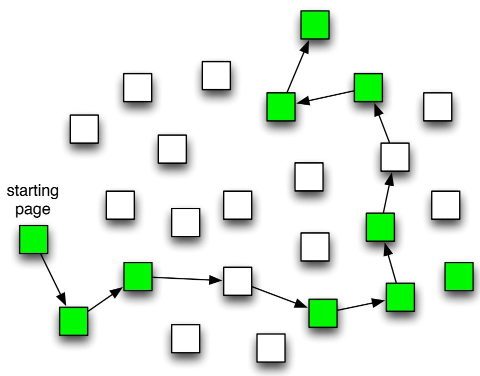


Focused Web Crawling

- The goal is to locate and download a large portion of web pages that match a given target theme as early as possible.

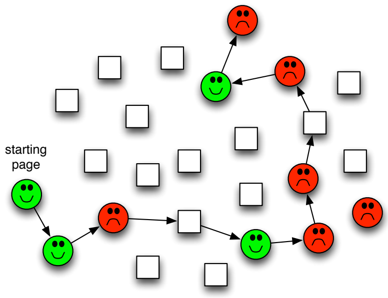
- Example themes
  - topic (nuclear energy)
  - media type (forums)
  - demographics (kids)

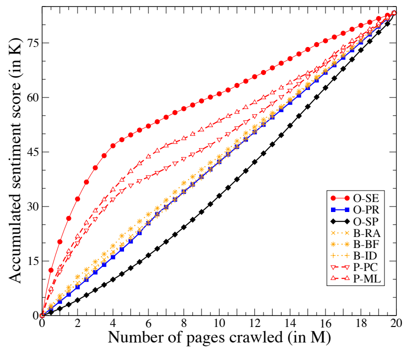
- Strategies
  - URL patterns
  - referring page content
  - local graph structure



Sentiment Focused Web Crawling

- Goal: to locate and download web pages that contain positive or negative sentiments (opinionated content) as early as possible

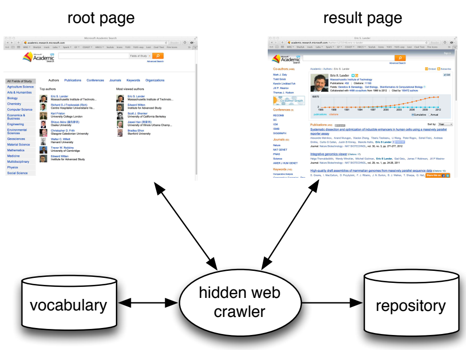




Hidden Web Crawling

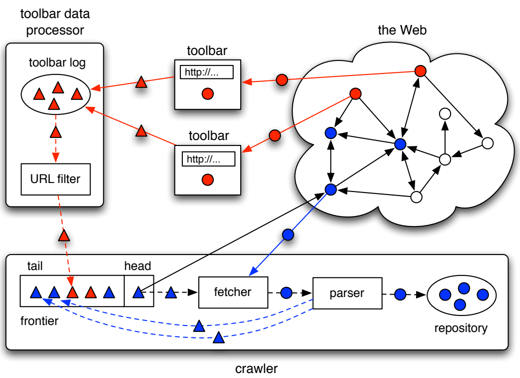
- Hidden Web: web pages that a crawler cannot access by simply following the link structure

- Examples
  - unlinked pages
  - private sites
  - contextual pages
  - scripted content
  - dynamic content



Research Problem – Passive Discovery

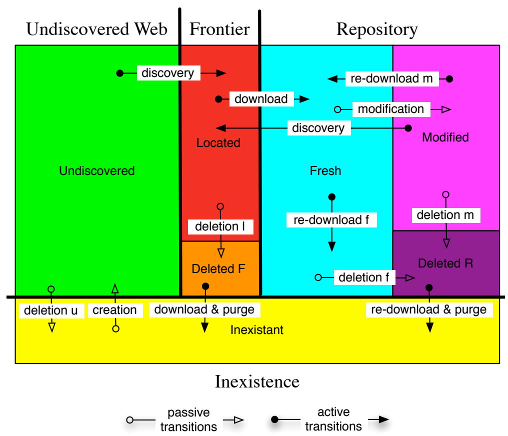
- URL discovery by external agents
  - toolbar logs
  - email messages
  - tweets
- Benefits
  - improved coverage
  - early discovery



Research Problem – URL Scheduling

- How to optimally allocate available crawling resources between the tasks of page discovery and refreshing

- Web master
  - create
  - modify
  - delete
- Crawler
  - discover
  - download
  - refresh



Research Problem – Web Partitioning

- Web partitioning/repartitioning: the problem of finding a Web partition that minimizes the costs in distributed Web crawling
  - minimization objectives
    - page download times
    - link exchange overhead
    - repartitioning overhead
  - constraints
    - coupling with distributed search
    - load balancing

Research Problem – Crawler Placement

- Crawler placement problem: the problem of finding the optimum geographical placement for a given number of data centers
  - geographical locations are now objectives, not constraints
- Problem variant: assuming some data centers were already built, find an optimum location to build a new data center for crawling

Research Problem – Coupling with Search

- Coupling with geographically distributed indexing/ search
  - crawled data may be moved to
    - a single data center
    - replicated on multiple data centers
    - partitioned among a number of data centers
  - decisions must be given on
    - what data to move (e.g., pages or index)
    - how to move (i.e., compression)
    - how often to move (i.e., synchronization)

Research Problem – Green Web Crawling

- Goal: reduce the carbon footprint generated by the web servers while handling the requests of web crawlers
- Idea
  - crawl web sites when they are consuming green energy (e.g., during the day when solar energy is more available)
  - crawl web sites consuming green energy more often as an incentive to promote the use of green energy

Published Web Crawler Architectures

- Bingbot: Microsoft's Bing web crawler
- FAST Crawler: Used by Fast Search & Transfer
- Googlebot: Web crawler of Google
- PolyBot: A distributed web crawler
- RBSE: The first published web crawler
- WebFountain: A distributed web crawler
- WebRACE: A crawling and caching module
- Yahoo Slurp: Web crawler used by Yahoo Search

Open Source Web Crawlers

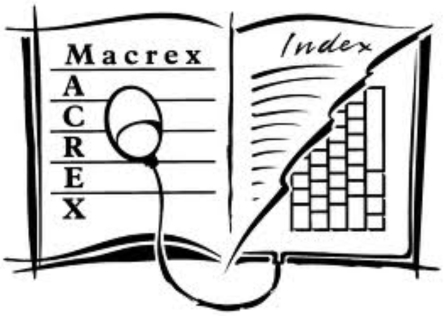
- DataparkSearch: GNU General Public License.
- GRUB: open source distributed crawler of Wikia Search
- Heritrix: Internet Archive's crawler
- ICDL Crawler: cross-platform web crawler
- Norconex HTTP Collector: licensed under GPL
- Nutch: Apache License
- Open Search Server: GPL license
- PHP-Crawler: BSD license
- Scrapy: BSD license
- Seeks: Affero general public license
- WIRE: Carlos Castillo’s PhD thesis

Key Papers

- Cho, Garcia-Molina, and Page, "Efficient crawling through URL ordering", WWW, 1998.
- Heydon and Najork, "Mercator: a scalable, extensible web crawler", World Wide Web, 1999.
- Chakrabarti, van den Berg, and Dom, "Focused crawling: a new approach to topic-specific web resource discovery", Computer Networks, 1999.
- Najork and Wiener, "Breadth-first crawling yields high-quality pages", WWW, 2001.
- Cho and Garcia-Molina, "Parallel crawlers", WWW, 2002.
- Cho and Garcia-Molina, "Effective page refresh policies for web crawlers", ACM Transactions on Database Systems, 2003.
- Lee, Leonard, Wang, and Loguinov, "IRLbot: Scaling to 6 billion pages and beyond", ACM TWEB, 2009.

Q&A

Indexing



Indexing

- Indexing is the process of converting crawled web documents into an efficiently (compressed) searchable form.
- An index is a representation for the document collection over which user queries will be evaluated.

Indexing

- Abandoned indexing techniques
  - suffix arrays
  - signature files
- Currently used indexing technique
  - inverted index (the oldest one!)

Signature File

- For a given piece of text, a signature is created by encoding the words in it
- For each word, a bit signature is computed
  - contains *F* bits
  - *m* out of *F* bits is set to 1 (decided by hash functions)
- In case of long documents
  - a signature is created for each logical text block
  - block signatures are concatenated to form the signature for the entire document

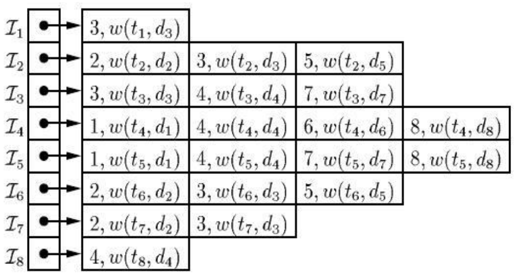
Signature File

- When searching, the signature for the query keyword is OR'ed with the document signature
- Example signature with $F = 6$ and $m = 2$

document terms:		query terms:	
apple	10 00 10	apple	10 00 10 (match)
orange	00 01 10	banana	01 00 01 (no match)
signature	10 01 10	peach	10 01 00 (false match)

Inverted Index

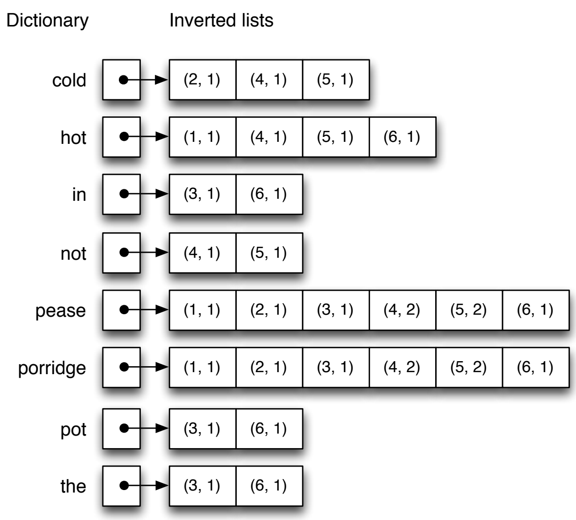
- An inverted index has two parts
  - inverted lists
    - posting entries
      - document id
      - term score
  - a vocabulary index (dictionary)



Sample Document Collection

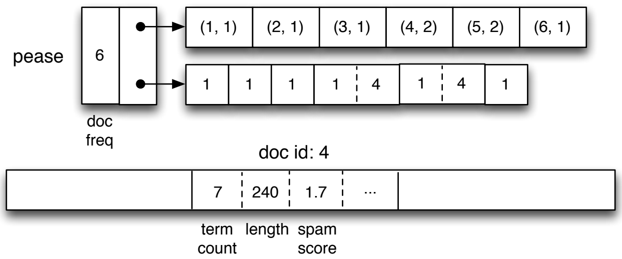
Doc id	Text
1	pease porridge hot
2	pease porridge cold
3	pease porridge in the pot
4	pease porridge hot, pease porridge not cold
5	pease porridge cold, pease porridge not hot
6	pease porridge hot in the pot

Inverted Index



Inverted Index

- Additional data structures
  - position lists: list of all positions a term occurs in a document
  - document array: document length, PageRank, spam score, …
- Sections
  - title, body, header, anchor text (inbound, outbound links)



Metrics

- Quality metrics
  - spam rate: fraction of spam pages in the index
  - duplicate rate: fraction of exact or near duplicate web pages present in the index
- Performance metrics
  - compactness: size of the index in bytes
  - deployment cost: time and effort it takes to create and deploy a new inverted index from scratch
  - update cost: time and space overhead of updating a document entry in the index

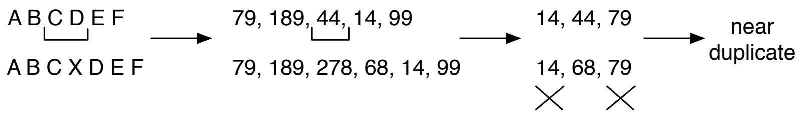
Indexing Documents

- Index terms are extracted from documents after some processing, which may involve
  - tokenization
  - stopword removal
  - case conversion
  - stemming

original text:	Living in America
applying all:	liv america
in practice:	living in america

Duplicate Detection

- Detecting documents with duplicate content
  - exact duplicates (solution: computing/comparing hash values)
  - near duplicates (solution: shingles instead of hash values)

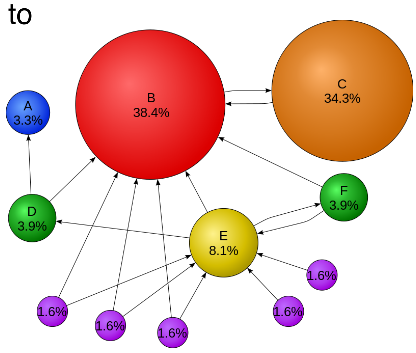


Features: Relevance Signals

- Offline computed features
  - content: spam score, domain quality score
  - web graph: PageRank, HostRank
  - usage: click count, CTR, dwell time
- Online computed features
  - query-document similarity: tf-idf, BM25
  - term proximity features

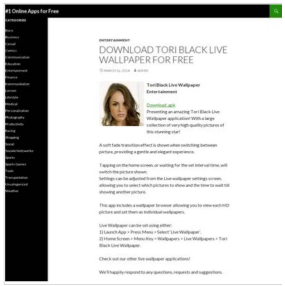
PageRank

- A link analysis algorithm that assigns a weight to each web page indicating its importance
- Iterative process that converges to a unique solution
- Weight of a page is proportional to
  - number of inlinks of the page
  - weight of linking pages
- Other algorithms
  - HITS
  - SimRank
  - TrustRank



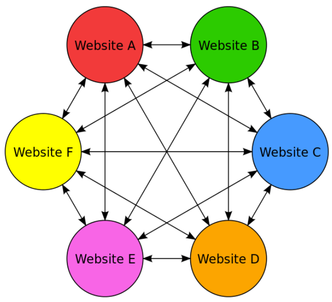
Spam Filtering

- Content spam
  - web pages with potentially many popular search terms and with little or no useful information
  - goal is to match many search queries to the page content and increase the traffic coming from search engines



Spam Filtering

- Link spam
  - a group of web sites that all link to every other site in the group
  - goal is to boost the PageRank score of pages and make them more frequently visible in web search results

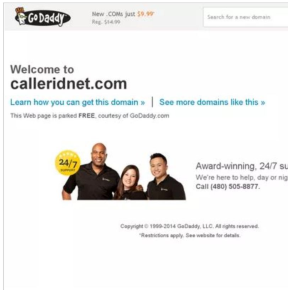


Spam Filtering

- PageRank is subject to manipulation by link farms

Top 10 PageRank sites

```
http://godaddy.com
http://twitter.com
http://facebook.com
http://blogger.com
http://google.com
http://mya.godaddy.com
http://adobe.com
http://community.godaddy.com
http://wordpress.org
http://youtube.com
```



Inverted List Compression

- Benefits
  - reduced space consumption
  - reduced transfer costs
  - increased posting list cache hit rate

- Gap encoding
  - original: 17 18 28 40 44 47 56 58
  - gap encoded: 17 1 10 12 4 3 9 2

gaps

Inverted List Compression

- Techniques
  - Unary encoding
  - Gamma encoding
  - Delta encoding
  - Variable byte encoding
  - Golomb encoding
  - Rice encoding
  - PforDelta encoding
  - Interpolative encoding

gap: 1000

unary: 11111111…11111110 (999 ones)
gamma: 1111111110:111101000
delta: 1110:010:111101000
vbe: 00000111 11101000

Document Identifier Reordering

- Goal: reassign document identifiers so that we obtain many small d-gaps, facilitating compression

- Example

old lists:	L1: 1 3 6 8 9	L2: 2 4 5 6 9	L3: 3 6 7 9
mapping:	1→1 2→9 3→2 4→7 5→8 6→3 7→5 8→6 9→4		
new lists:	L1: 1 2 3 4 6	L2: 3 4 7 8 9	L3: 2 3 4 5
old d-gaps:	2 3 2 1	2 1 1 3	3 1 2
new d-gaps:	1 1 1 2	1 3 1 1	1 1 1

Document Identifier Reordering

- Techniques
  - sorting URLs alphabetically and assigning ids in that order
    - idea: pages from the same site have high textual overlap
    - simple yet effective
    - only applicable to web page collections
  - clustering similar documents
    - assigns nearby ids to documents in the same cluster
  - traversal of document similarity graph
    - formulated as the traveling salesman problem

Index Construction

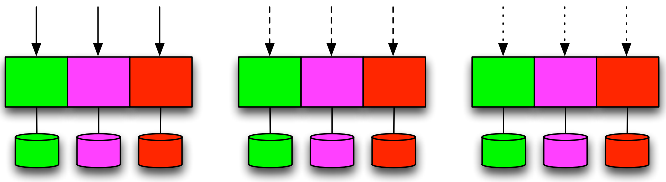
- Equivalent to computing the transpose of a matrix
- In-memory techniques do not work well with web-scale data
- Techniques
  - two-phase
    - first phase: read the collection and allocate a skeleton for the index
    - second phase: fill the posting lists
  - one-phase
    - keep reading documents and building an in-memory index
    - each time the memory is full, flush the index to the disk
    - merge all on-disk indexes into a single index in a final step

Index Maintenance

- Grow a new (delta) index in the memory; each time the memory is full, flush the in-memory index to disk
  - no merge
    - flushed index is written to disk as a separate index
    - increases fragmentation and query processing time
    - eventually requires merging all on-disk indexes or rebuilding
  - incremental indexing
    - each inverted list contains additional empty space at the end
    - new documents are appended to the empty space in the list
  - merging delta index
    - immediate merge: maintains only one copy of the index on disk
    - selective merge: maintains multiple generations of the index on disk

Inverted Index Partitioning/Replication

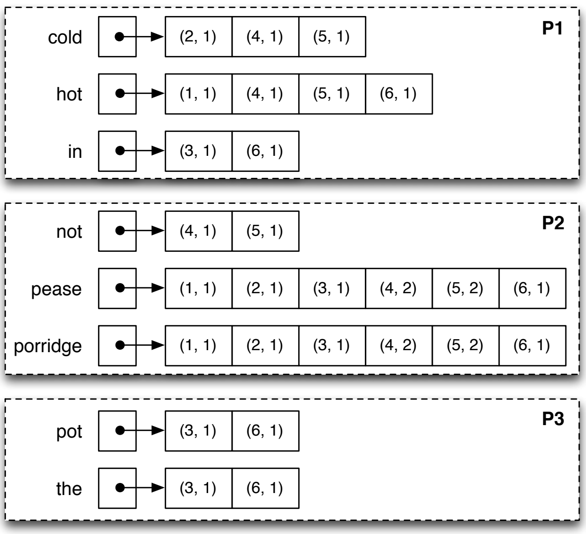
- In practice, the inverted index is
  - partitioned on thousands of computers in a large search cluster
    - reduces query response times
    - allows scaling with increasing collection size
  - replicated on tens of search clusters
    - increases query processing throughput
    - allows scaling with increasing query volume
    - provides fault tolerance



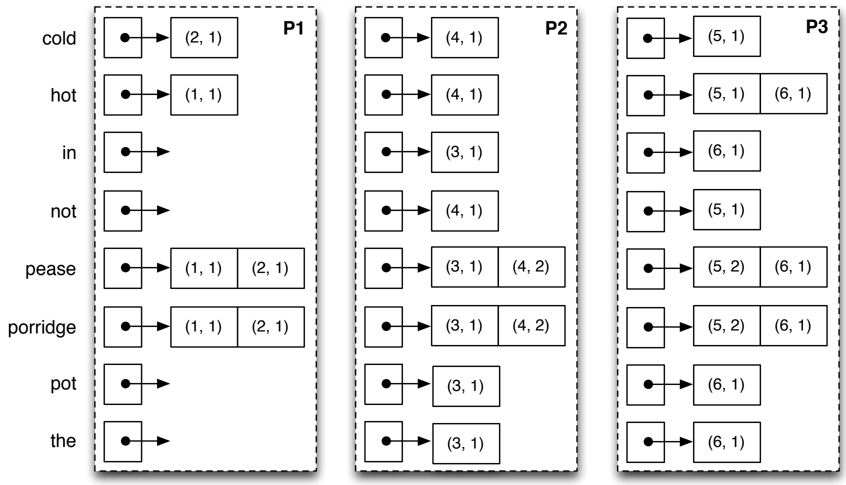
Inverted Index Partitioning

- Two alternatives for partitioning an inverted index
  - term-based partitioning
    - *T* inverted lists are distributed across *P* processors
    - each processor is responsible for processing the postings of a mutually disjoint subset of inverted lists assigned to itself
    - single disk access per query term
  - document-based partitioning
    - *N* documents are distributed across *P* processors
    - each processor is responsible for processing the postings of a mutually disjoint subset of documents assigned to itself
    - multiple (parallel) disk accesses per query term

Term-Based Index Partitioning



Document-Based Index Partitioning



Comparison of Index Partitioning Approaches

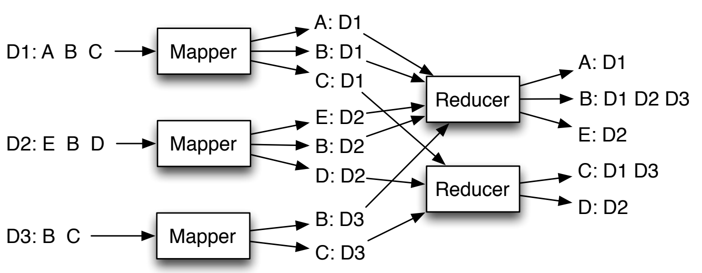
	Document-based	Term-based
Space consumption	Higher	Lower
Number of disk accesses	Higher	Lower
Concurrency	Lower	Higher
Computational load imbalance	Lower	Higher
Max. posting list I/O time	Lower	Higher
Cost of index building	**Lower**	**Higher**
Maintenance cost	**Lower**	**Higher**

Inverted Index Partitioning

- In practice, document-based partitioning is used
  - simpler to build and update
  - low inter-query-processing concurrency, but good load balance
  - low throughput, but high response time
  - high throughput is achieved by replication
  - easier to maintain
  - more fault tolerant
- Hybrid techniques are possible (e.g., term partitioning inside a document sub-collection)

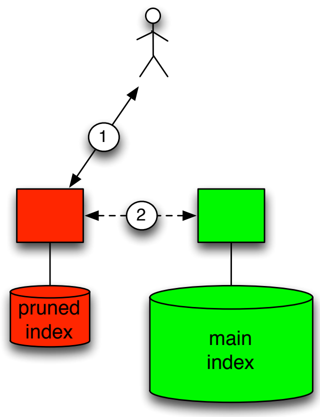
Parallel Index Creation

- Possible alternatives for creating an inverted index in parallel
  - message passing paradigm
    - doc-based: each node builds a local index using its documents
    - term-based: posting lists are communicated between the nodes
  - MapReduce framework



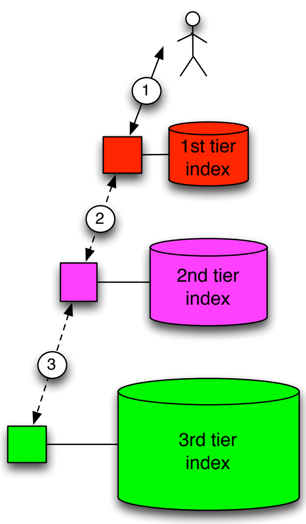
Static Index Pruning

- Idea: to create a small version of the search index that can accurately answer most search queries
- Techniques
  - term-based pruning
  - doc-based pruning
- Result quality
  - guaranteed
  - not guaranteed
- In practice, caching does the same better



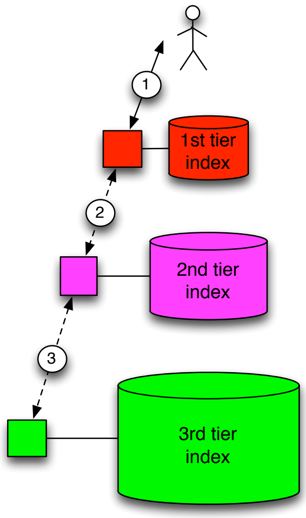
Tiered Index

- A sequence of sub-indexes
  - former sub-indexes are small and keep more important documents
  - later sub-indexes are larger and keep less important documents
  - a query is processed selectively only on the first *n* tiers
- Two decisions need to be made
  - tiering (offline): how to place documents in different tiers
  - fall-through (online): at which tier to stop processing the query



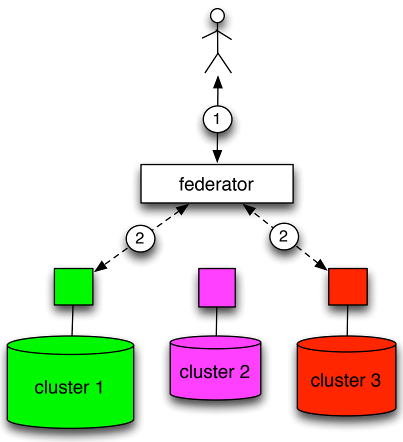
Tiered Index

- Tiering strategy is based on some document importance metric
  - PageRank
  - click count
  - spam score
- Fall-through strategy
  - query the next index until there are enough results
  - query the next index until search result quality is good
  - predict the next tier's result quality by machine learning



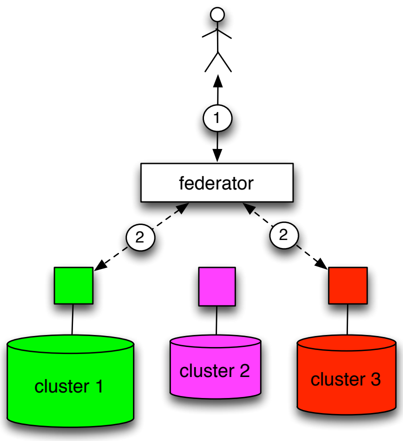
Document Clustering

- Documents are clustered
  - similarity between documents
  - co-click likelihood
- A separate index is built for each document cluster

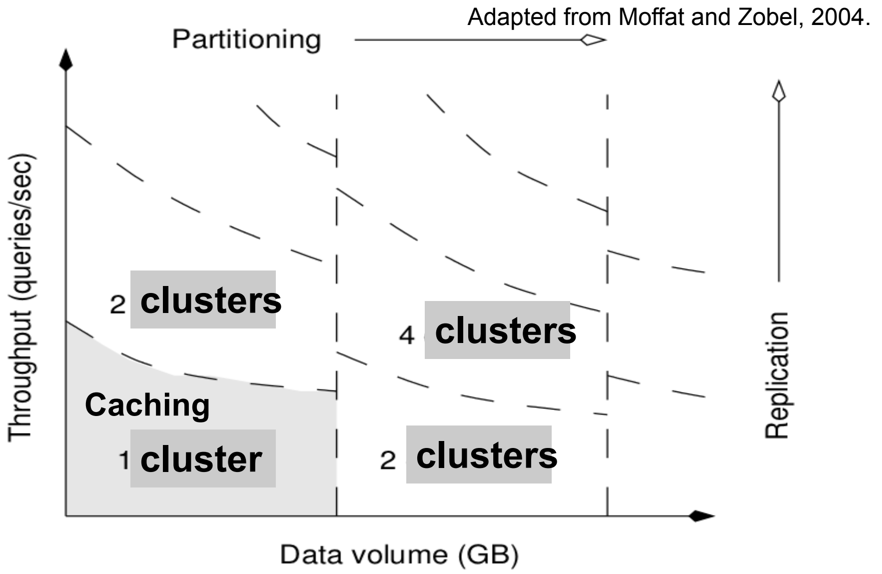


Document Clustering

- A query is processed on the indexes associated with the most similar *n* clusters
- Reduces the workload
- Suffers from the load imbalance problem
  - query topic distribution may be skewed
  - certain indexes have to be queried much more often

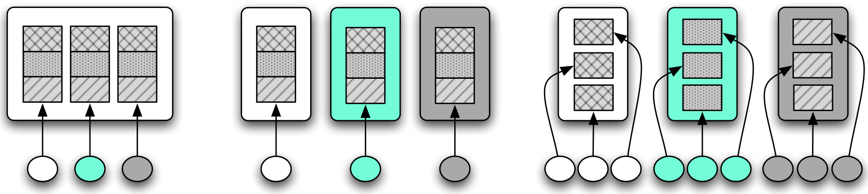


Scaling Up



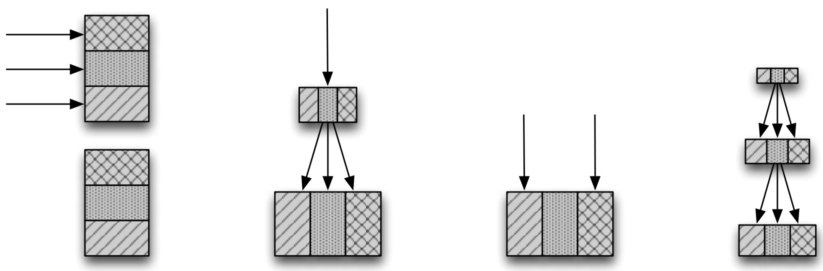
Multi-site Architectures

- Centralized
- Replicated
- Partitioned



Research Problem – Indexing Architectures

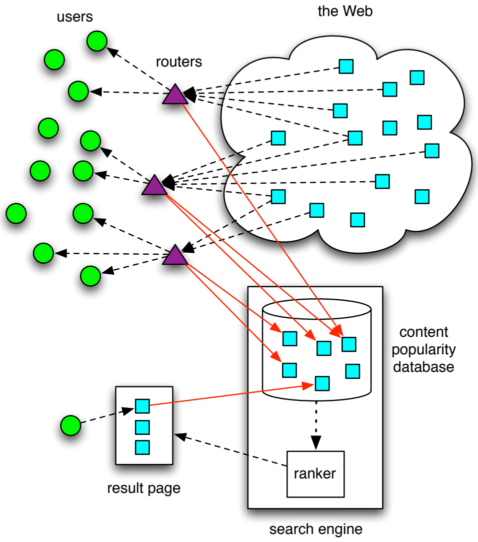
- Replicated
- Pruned
- Clustered
- Tiered



- Missing research
  - a thorough performance comparison of these architectures that includes all possible optimizations
  - scalability analysis with web-scale document collections and large numbers of nodes

Research Problem – Off-network Popularity

- Traditional features
  - statistical analysis
  - link analysis
  - proximity
  - spam
  - clicks
  - session analysis
- Off-network popularity
  - social signals
  - network



Key Papers

- Brin and Page, "The anatomy of a large-scale hypertextual Web search engine", Computer Networks and ISDN Systems, 1998.
- Zobel, Moffat, and Ramamohanarao, "Inverted files versus signature files for text indexing". ACM Transactions on Database Systems, 1998.
- Page, Brin, Motwani, and Winograd, "The PageRank citation ranking: bringing order to the Web", Technical report, 1998.
- Kleinberg, "Authoritative sources in a hyperlinked environment", Journal of the ACM, 1999.
- Ribeiro-Neto, Moura, Neubert, and Ziviani, "Efficient distributed algorithms to build inverted files", SIGIR, 1999.
- Carmel, Cohen, Fagin, Farchi, Herscovici, Maarek, and Soffer, "Static index pruning for information retrieval systems, SIGIR, 2001.
- Scholer, Williams, Yiannis, and Zobel, "Compression of inverted indexes for fast query evaluation", SIGIR, 2002.

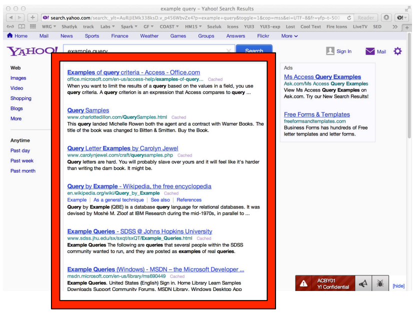
Q&A

Query Processing

Query Processing

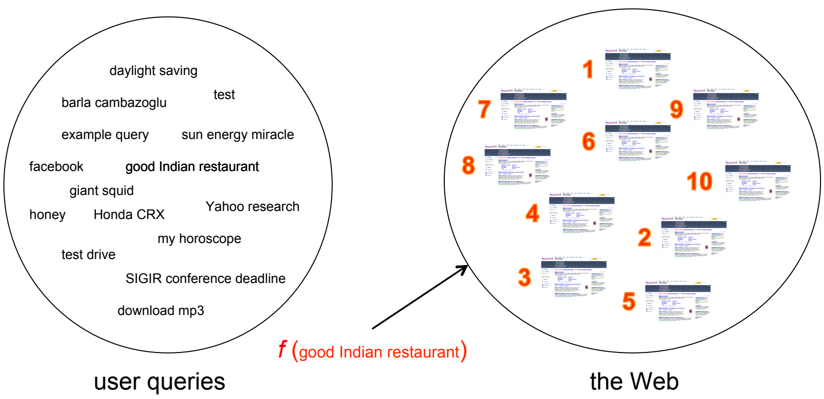
- Query processing is the problem of generating the best-matching answers (typically, top 10 documents) to a given user query, spending the least amount of time

- Our focus: creating 10 blue links as an answer to a user query



Web Search

- Web search is a sorting problem!

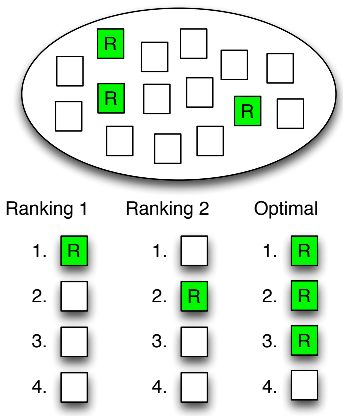


Metrics

- Quality metrics
  - relevance: the degree to which returned answers meet user's information need.
- Performance metrics
  - latency: the response time delay experienced by the user
  - peak throughput: number of queries that can be processed per unit of time without any degradation on other metrics

Measuring Relevance

- It is not always possible to know the user's intent and his information need
- Commonly used relevance metrics in practice
  - recall
  - precision
  - DCG
  - NDCG



	Ranking 1	Ranking 2	Optimal
Recall:	1/3	1/3	1
Precision:	1/4	1/4	3/4
DCG:	1	0.63	1+0.63+0.5=2.13
NDCG:	1/2.13	0.63/2.13	

Estimating Relevance

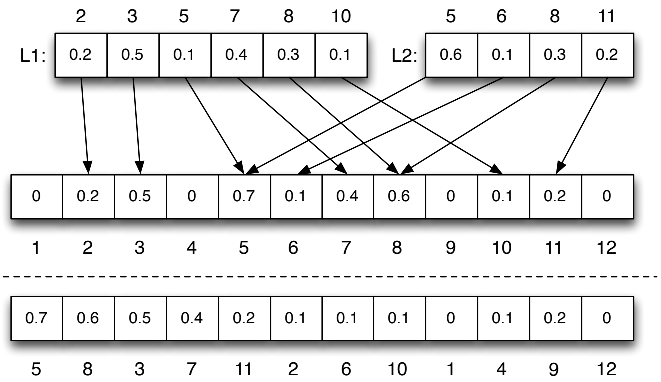
- How to estimate the relevance between a given document and a user query?
- Alternative models for score computation
  - vector-space model
  - statistical models
  - language models
- They all pretty much boil down to the same thing

Example Scoring Function

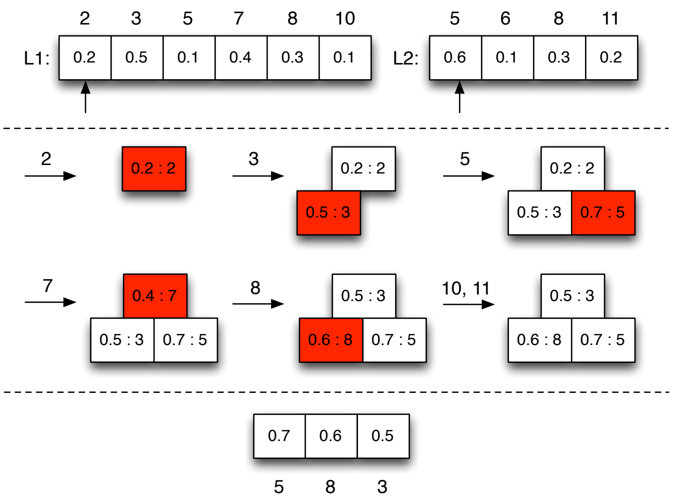
- Notation
  - q : a user query
  - d : a document in the collection
  - t : a term in the query
  - N : number of documents in the collection
  - $tf(t,d)$: number of occurrences of the term in the document
  - $df(t)$: number of documents containing the term
  - $|d|$: number of unique terms in the document
- Commonly used scoring function: tf-idf

$$s(q,d) = \sum_{t \in q} w(t,d) \times \log \frac{N}{df(t)} \qquad w(t,d) = \frac{tf(t,d)}{|d|}$$

Score Computation (using an accumulator array)



Efficient Score Computation (using a min heap)



Design Alternatives in Ranking

- Document matching
  - conjunctive (AND) mode
    - the document must contain all query terms
    - higher precision, lower recall
  - disjunctive (OR) mode
    - document must contain at least one of the query terms
    - lower precision, higher recall

Design Alternatives in Ranking

- Inverted list organizations
  - increasing order of document ids
  - decreasing order of weights in postings
    - sorted by term frequency
    - sorted by score contribution (impact)
      - within the same impact block, sorted in increasing order of document ids

2	3	5	7	8	10
0.2	0.5	0.1	0.4	0.3	0.1

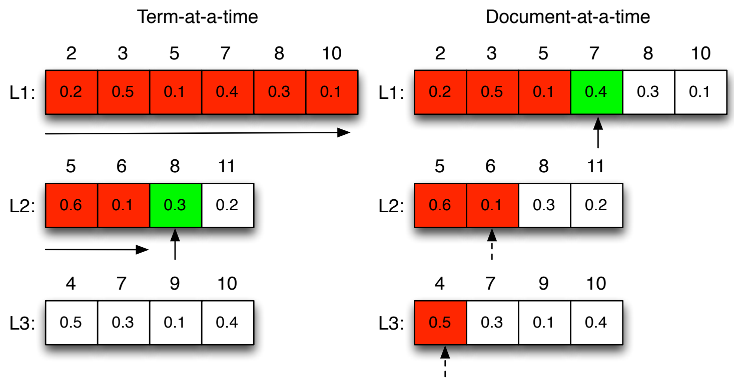
doc id ordered

3	7	8	2	5	10
0.5	0.4	0.3	0.2	0.1	0.1

weight ordered

Design Alternatives in Ranking

- Traversal order
  - term-at-a-time (TAAT)
  - document-at-a-time (DAAT)



Design Alternatives in Ranking

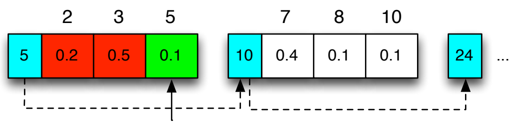
- Weights stored in postings
  - term frequency
    - suitable for compression
  - normalized term frequency
    - no need to store the document length array
    - not suitable for compression
  - precomputed score
    - no need to store the idf value in the vocabulary
    - no need to store the document length array
    - not suitable for compression

Design Alternatives in Ranking

- In practice
  - AND mode: faster and leads to better results in web search
  - doc-id sorted lists: enables compression
  - document-at-a-time list traversal: enables better optimizations
  - term frequencies: enables compression

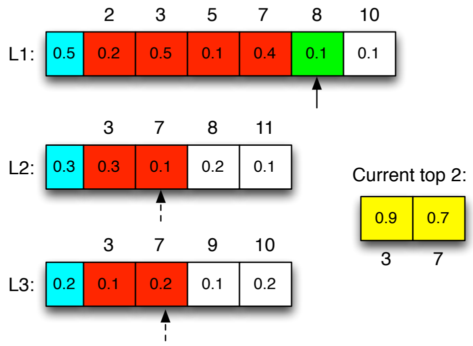
Scoring Optimizations

- Skipping
  - list is split into blocks linked with pointers called skips
  - store the maximum document id in each block
  - skip a block if it is guaranteed that sought document is not within the block
  - gains in decompression time
  - overhead of skip pointers



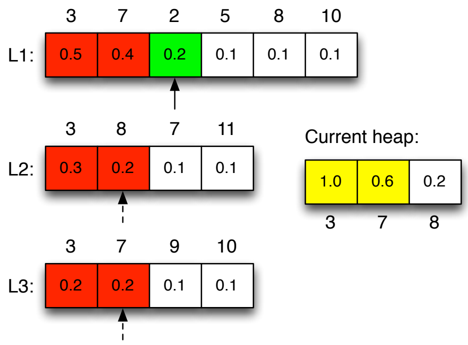
Scoring Optimizations

- Dynamic index pruning
  - store the maximum possible score contribution of each list
  - compute the maximum possible score for the current document
  - compare with the lowest score in the heap
  - gains in scoring and decompression time



Scoring Optimizations

- Early termination
  - stop scoring documents when it is guaranteed that neither document can make it into the top *k* list
  - gains in scoring and decompression time



Snippet Generation

- Search result snippets (a.k.a., summary or abstract)
  - important for users to correctly judge the relevance of a web page to their information need before clicking on its link
- Snippet generation
  - a forward index is built providing a mapping between pages and the terms they contain
  - snippets are computed using this forward index and only for the top 10 result pages
  - efficiency of this step is important
  - entire page as well as snippets can be cached

Parallel Query Processing

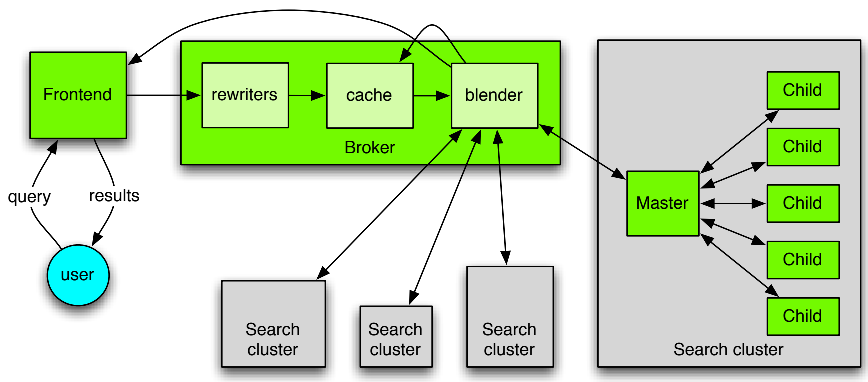
- Query processing can be parallelized at different granularities
  - parallelization within a search node (intra-query parallelism)
  - multi-threading within a search node (inter-query parallelism)
  - parallelization within a search cluster (intra-query parallelism)
  - replication across search clusters (inter-query parallelism)
  - distributed across multiple data centers

Query Processing Architectures

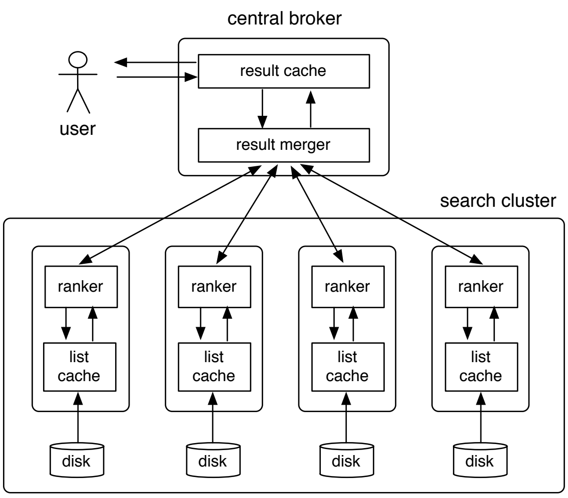
- Single computer
  - not scalable in terms of response time
- Search cluster
  - large search clusters (low response time)
  - replicas of clusters (high query throughput)
- Multiple search data centers
  - reduces user-to-center latencies

Query Processing in a Data Center

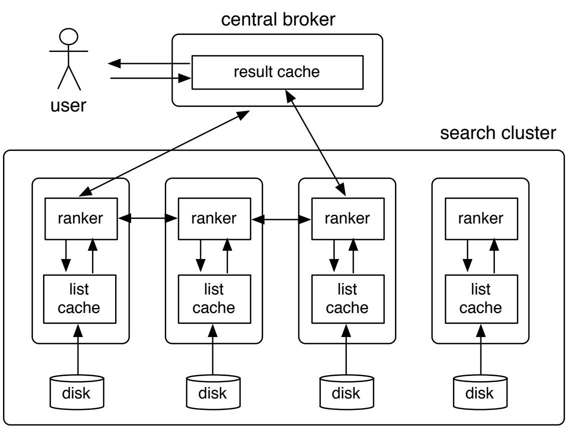
- Multiple node types
  - frontend, broker, master, child



Parallel Query Processing (central broker)

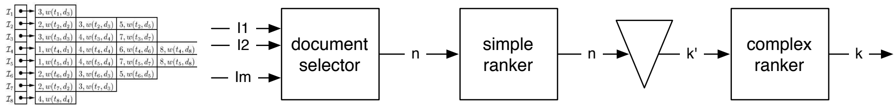


Parallel Query Processing (pipelined)



Query Processing on a Search Node

- Two-phase ranking
  - simple ranking
    - linear combination of query-dependent and query-independent scores potentially with score boosting
    - main objective: efficiency
  - complex ranking
    - machine learned
    - main objective: quality



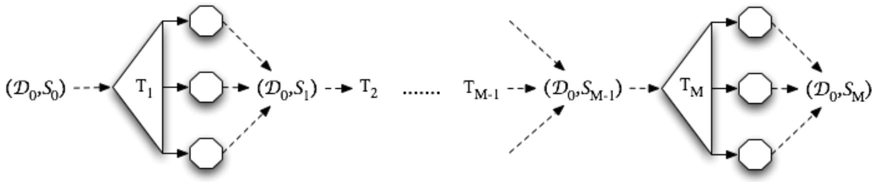
Machine Learned Ranking

- Many features
  - term statistics (e.g., BM25)
  - term proximity
  - link analysis (e.g., PageRank)
  - spam detection
  - click data
  - search session analysis

- Popular learners used by commercial search engines
  - neural networks
  - boosted decision trees

Machine Learned Ranking

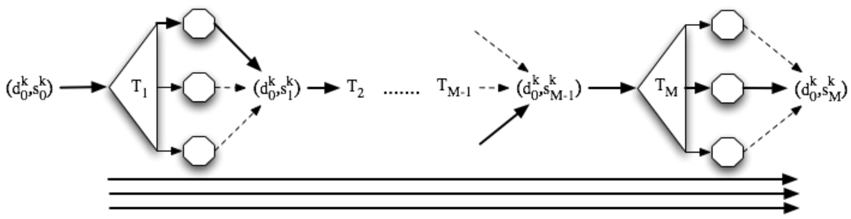
- Example: gradient boosted decision trees
  - chain of weak learners
  - each learner contributes a partial score to the final document score



- Assuming
  - 1000 trees
  - an average tree depth of 10
  - 100 documents scored per query
  - 1000 search nodes
- Expensive
  - 1000*10*100 = 1 million operations per query and per node
  - around 1 billion comparison for the entire search cluster

Machine Learned Ranking

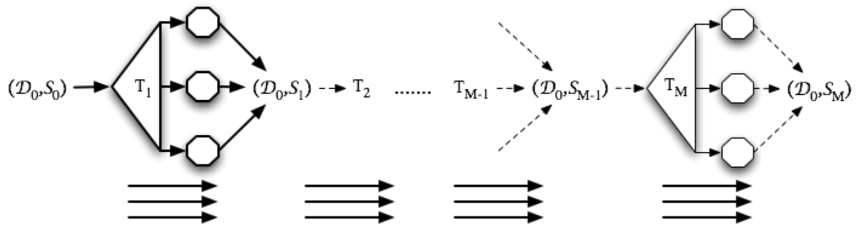
- Document-ordered traversal (DOT)
  - scores are computed one document at a time over all scorers
  - an iteration of the outer loop produces the complete score information for a partial set of documents



- Disadvantages
  - poor branch prediction because a different scorer is used in each inner loop iteration
  - poor cache hit rates in accessing the data about scorers (for the same reason)

Machine Learned Ranking

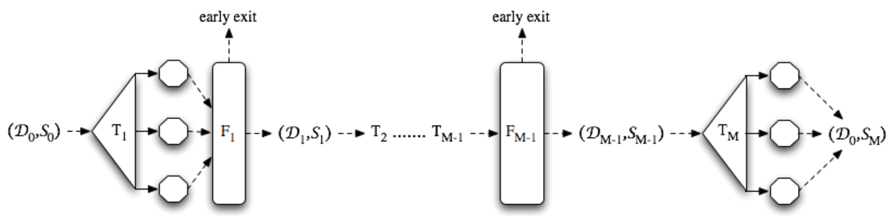
- Scorer-ordered traversal (SOT)
  - scores are computed one score at a time over all documents
  - an iteration of the outer loop produces the partial score information for the complete set of documents



- Disadvantages
  - memory requirement (feature vectors of all documents need to be kept in memory)
  - poor cache hit rates in accessing features as a different document is used in each inner loop iteration

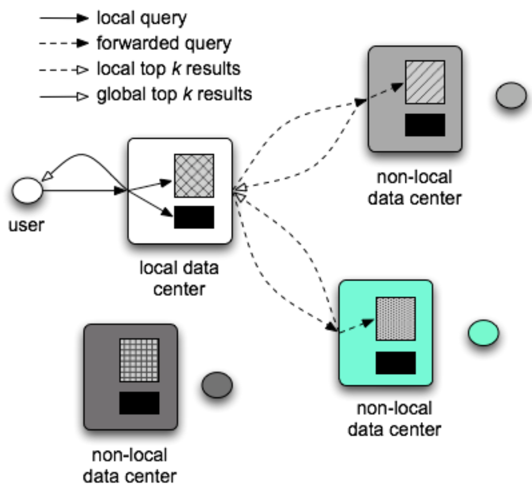
Machine Learned Ranking

- Early termination
  - idea: place predictive functions between scorers
    - predict during scoring whether a document will enter the final top *k* and
    - quit scoring, accordingly

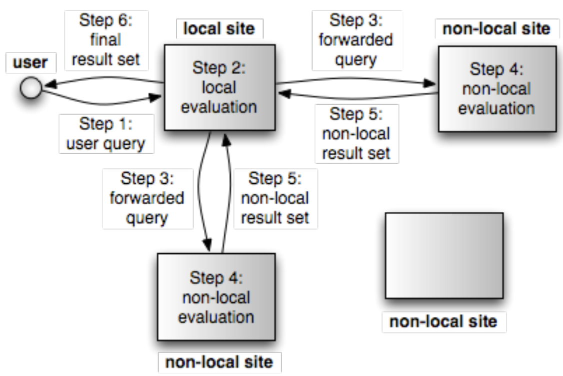


Multi-site Web Search Architecture

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



Multi-site Distributed Query Processing



- Local query response time
  - 2 × user-to-site latency
  - local processing time
- Forwarded query response time
  - local query response time
  - 2 × site-to-site latency
  - non-local query processing time

Query Forwarding

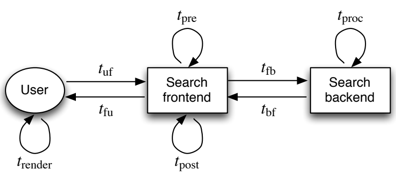
- Problem
  - selecting a subset of non-local sites which the query will be forwarded to
- Objectives
  - reducing the loss in search quality w.r.t. to evaluation over the full index
  - reducing average query response times and/or query processing workload

→ no change ↑ increase ↓ decrease

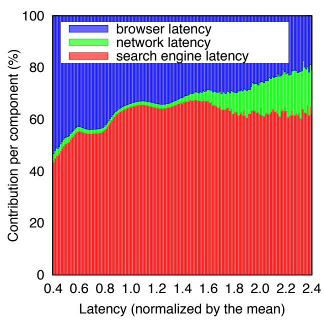
Black: no work Blue: useful work Red: useless work

	Result quality	Response time	Workload
True positive	→ (blue)	↑ (blue)	↑ → (blue)
False positive	→ (red)	↑ (red)	↑ → (red)
True negative	→	→	→
False negative	↓	→	→

Web Search Response Latency



- Constituents of response latency
  - user-to-center network latency
  - search engine latency time
    - query features
    - caching
    - current system workload
  - page rendering time in the browser

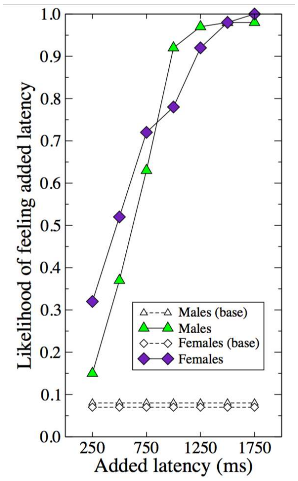


Response Latency Prediction

- Problem: Predict the response time of a query before processing it on the backend search system
- Useful for making query scheduling decisions
- Solution: Build a machine learning model with many features
  - number of query terms
  - total number of postings in inverted lists
  - average term popularity
  - …

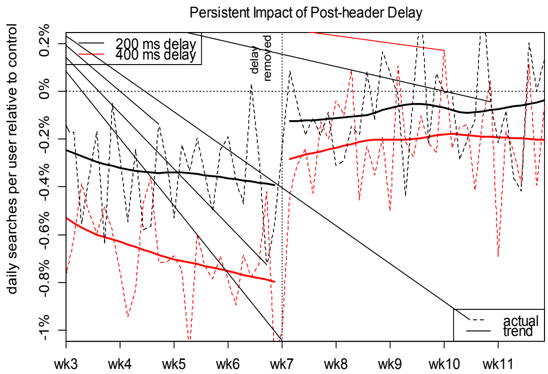
Impact of Response Latency on Users

- Impact of latency on the user varies depending on
  - context: time, location
  - user: demographics
  - query: intent



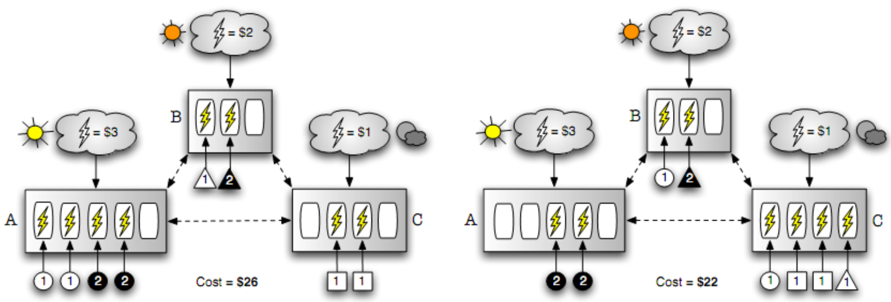
Impact of Response Latency on Users

- A/B test by Google and Microsoft



Research Problem – Energy Savings

- Observation: electricity prices vary across data centers and depending the time of the day
- Idea: forward queries to cheaper search data centers to reduce the electricity bill under certain constraints



Research Problem – Green Search Engines

- Goal: reduce the carbon footprint of the search engine
- Query processing
  - shift workload from data centers that consume brown energy to those green energy
  - constraints:
    - response latency
    - data center capacity

Open Source Search Engines

- DataparkSearch: GNU general public license
- Lemur Toolkit & Indri Search Engine: BSD license
- Lucene: Apache software license
- mnoGoSearch: GNU general public license
- Nutch: based on Lucene
- Seeks: Affero general public license
- Sphinx: free software/open source
- Terrier Search Engine: open source
- Zettair: open source

Key Papers

- Turtle and Flood, "Query evaluation: strategies and optimizations", Information Processing and Management, 1995.
- Barroso, Dean, and Holzle, "Web search for a planet: the Google cluster architecture", IEEE Micro, 2003.
- Broder, Carmel, Herscovici, Soffer, and Zien, "Efficient query evaluation using a two-level retrieval process", CIKM, 2003.
- Chowdhury and Pass, “Operational requirements for scalable search systems”, CIKM, 2003.
- Moffat, Webber, Zobel, and Baeza-Yates, "A pipelined architecture for distributed text query evaluation", Information Retrieval, 2007.

Key Papers

- Turpin, Tsegay, Hawking, and Williams, "Fast generation of result snippets in web search. SIGIR, 2007.
- Baeza-Yates, Gionis, Junqueira, Plachouras, and Telloli, "On the feasibility of multi-site web search engines", CIKM, 2009.
- Cambazoglu, Zaragoza, Chapelle, Chen, Liao, Zheng, and Degenhardt, "Early exit optimizations for additive machine learned ranking systems", WSDM, 2010.
- Wang, Lin, and Metzler, "Learning to efficiently rank", SIGIR, 2010.
- Macdonald, Tonellotto, Ounis, "Learning to predict response times for online query scheduling", SIGIR, 2012.

Q&A

Caching

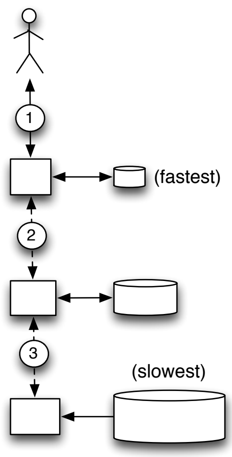
Caching

- Cache: quick-access storage system
  - may store data to eliminate the need to fetch the data from a slower storage system
  - may store precomputed results to eliminate redundant computation in the backend

	Main backend	Cache
speed	slower	faster
workload	higher	lower
capacity	larger	smaller
cost	cheaper	more expensive
freshness	more fresh	more stale

Caching

- Often appears in a hierarchical form
  - OS: registers, L1 cache, L2 cache, memory, disk
  - network: browser cache, web proxy, server cache, backend database
- Benefits
  - reduces the workload
  - reduces the response time
  - reduces the financial cost



Metrics

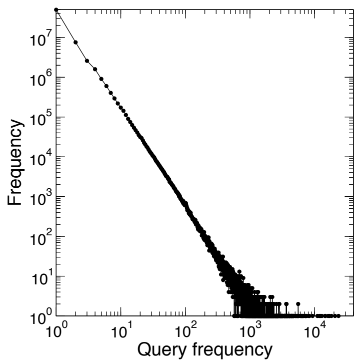
- Quality metrics
  - freshness: average staleness of the data served by the cache
- Performance metrics
  - hit rate: fraction of total requests that are answered by the cache
  - cost: total processing cost incurred to the backend system

Caching

- Items preferred for caching
  - more popular over time
  - more recently requested

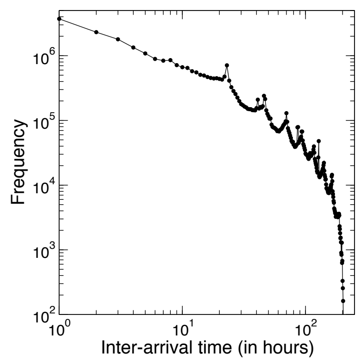
Query Frequency

- Skewed distribution in query frequency
  - Few queries are issued many times (head queries)
  - Many queries are issued rarely (tail queries)



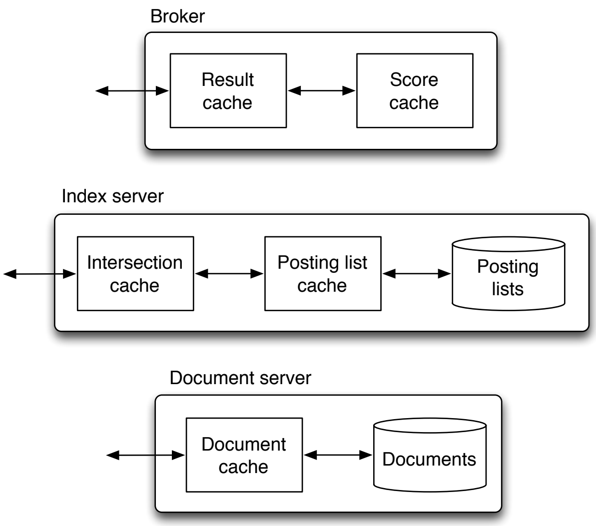
Inter-Arrival Time

- Skewed distribution in query inter-arrival time
  - low inter-arrival time is for many queries
  - high inter-arrival time for few queries



Caches Available in a Web Search Engine

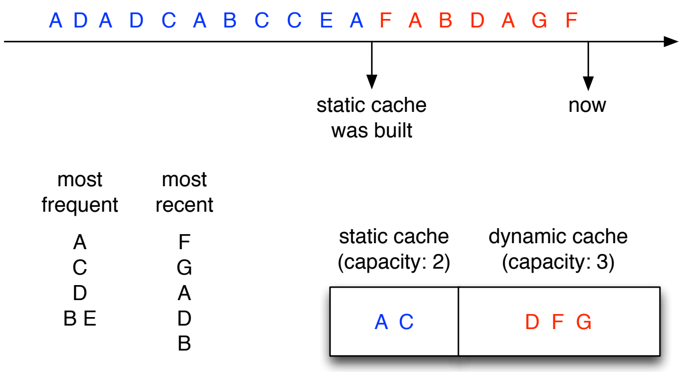
- Main caches
  - result
  - score
  - intersection
  - posting list
  - document



Caching Techniques

- Static caching
  - built in an offline manner
  - prefers items that are accessed often in the past
  - periodically re-deployed
- Dynamic caching
  - maintained in an online manner
  - prefers items that are recently accessed
  - requires removing items from the cache (eviction)
- Static/dynamic caching
  - shares the cache space between a static and a dynamic cache

Static/Dynamic Caching



Techniques Used in Caching

- Admission: items are prevented from being cached
- Prefetching: items are cached before they are requested
- Eviction: items are removed from the cache when it is full

Admission

- Idea: certain items may be prevented from being cached forever or until confidence is gained about their popularity
- Example admission criteria
  - query length
  - query frequency

Minimum frequency threshold for admission: 2
Maximum query length for admission: 4

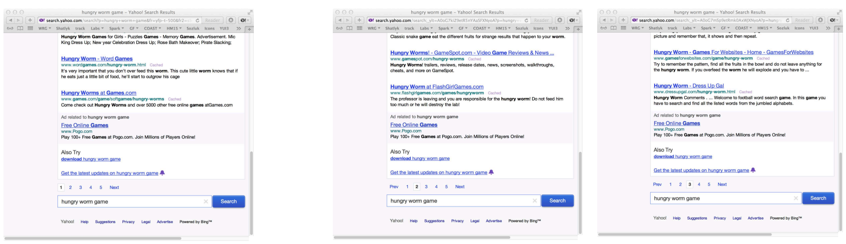
Query stream: ABC IJKLMN ABC ABC XYZ Q XYZ

Query cache	Result cache
XYZ Q	ABC

Prefetching

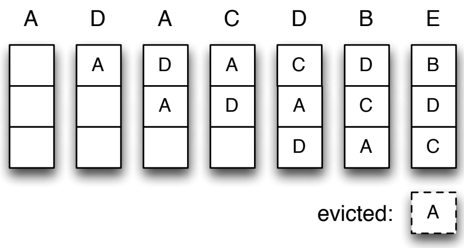
- Idea: certain items can be cached before they are actually requested if there is enough confidence that they will be requested in the near future.
- Example use case: result page prefetching

Reqested: page 1   Prefetch: page 2 and page 3



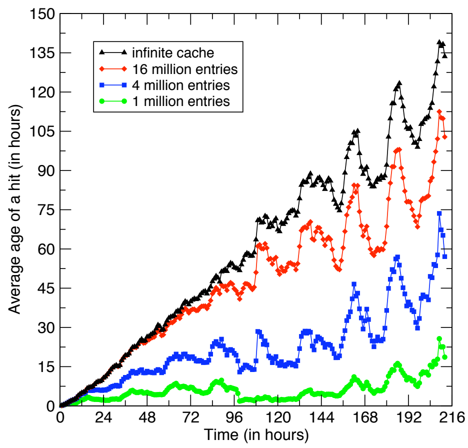
Eviction

- Goal: remove items that are not likely to lead to a cache hit from the cache to make space for items that are more useful.
- Ideal policy: evict the item that will be requested in the most distant future.
- Policies: FIFO, LFU, LRU, SLRU
- Most common: LRU



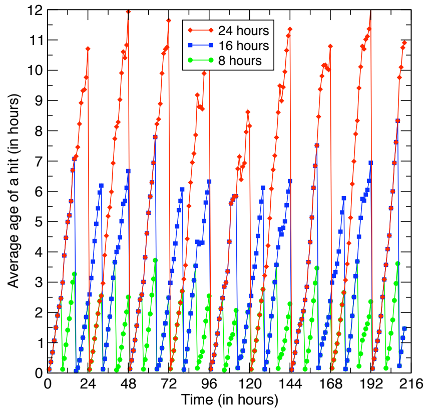
Result Cache Freshness

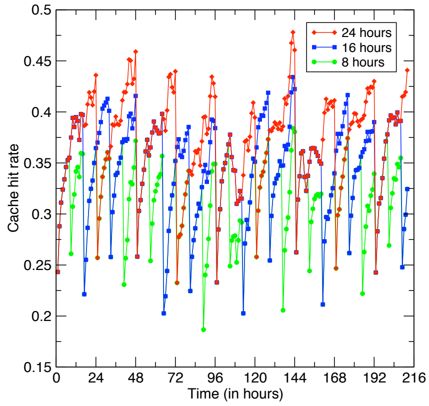
- In web search engines
  - index is continuously updated or re-built
  - result caches are almost infinite capacity
  - staleness problem



Flushing

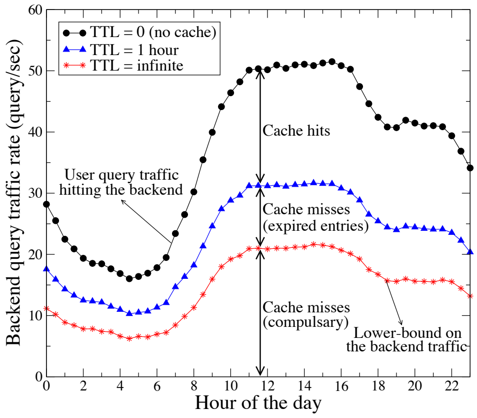
- Naïve solution: flushing the cache at regular time intervals



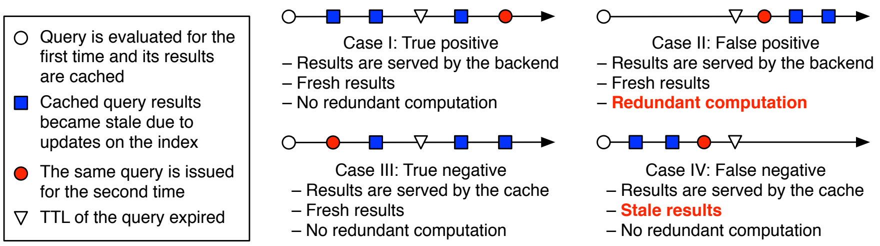


Time-to-Live (TTL)

- Common solution: setting a time-to-live value for each item
- TTL strategies
  - fixed: all items are associated with the same TTL value
  - adaptive: a separate TTL value is set for each item



Result Cache Freshness



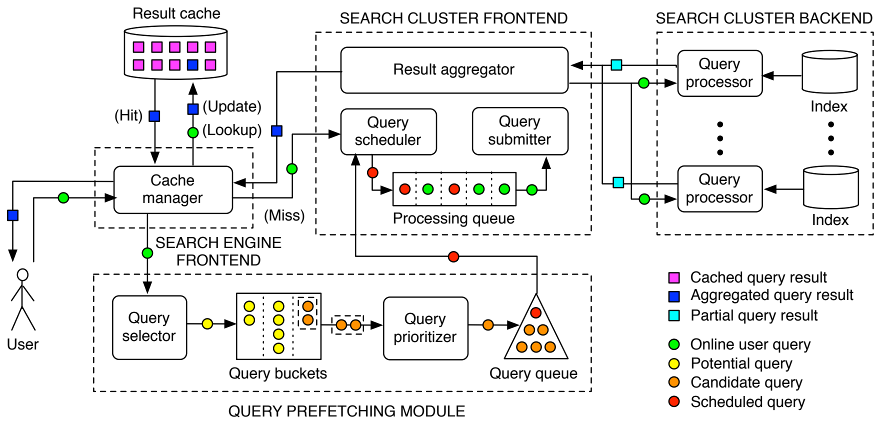
Advanced Solutions

- Cache invalidation: the indexing system provides feedback to the caching system upon an index update so that the caching system can identify the stale results
  - hard to implement
  - incurs communication and computation overheads
  - highly accurate

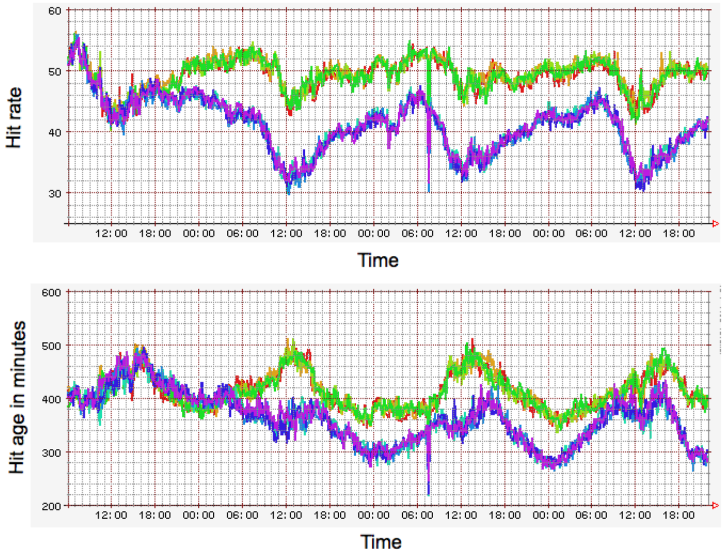
Advanced Solutions

- Cache refreshing: stale results are predicted and scheduled for re-computation in idle cycles of the backend search system
  - easy to implement
  - little computational overhead
  - not very accurate

Result Cache Refreshing Architecture

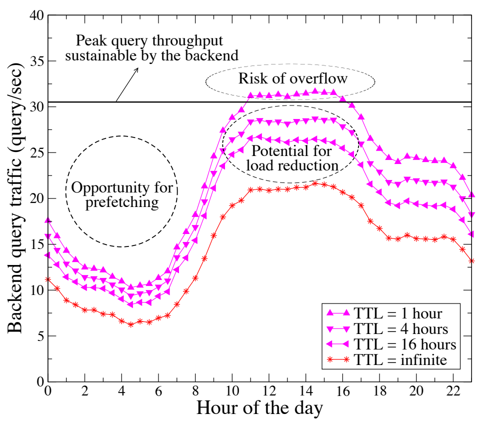


Impact on a Production System



Research Problem - Financial Perspective

- Past work optimizes
  - hit rate
  - backend workload
- Optimizing financial cost
  - result degradation
  - staleness
  - current query traffic
  - peak sustainable traffic
  - current electricity price



Key Papers

- Markatos, “On caching search engine query results”, Computer Communications, 2001.
- Long and Suel, “Three-level caching for efficient query processing in large web search engines”, WWW, 2005.
- Fagni, Perego, Silvestri, and Orlando, “Boosting the performance of web search engines: caching and prefetching query results by exploiting historical usage data”, ACM Transactions on Information Systems, 2006.
- Altingovde, Ozcan, and Ulusoy, “A cost-aware strategy for query result caching in web search engines”, ECIR, 2009.
- Cambazoglu, Junqueira, Plachouras, Banachowski, Cui, Lim, and Bridge, “A refreshing perspective of search engine caching”, WWW, 2010.

Q&A

Concluding Remarks

Summary

- Presented a high-level overview of important scalability and efficiency issues in large-scale web search
- Provided a summary of commonly used metrics
- Discussed a number of potential research problems in the field
- Provided references to available software and key research works in literature

Observations

- Unlike past research, the current research on scalability is mainly driven by the needs of commercial search engine companies
- Scalability of web search engines is likely to be a research challenge for some more time (at least, in the near future)
- Lack of hardware resources and large datasets render scalability research quite difficult, especially for researchers in academia

Suggestions to Newcomers

- Follow the trends in the Web, user bases, and hardware parameters to identify the real performance bottlenecks
- Watch out newly emerging techniques whose primary target is to improve the search quality and think about their impact on search performance
- Re-use or adapt existing solutions in more mature research fields, such as databases, computer networks, and distributed computing
- Know the key people in the field (the community is small) and follow their work

Important Surveys/Books

- Web search engine scalability and efficiency
  - B. B. Cambazoglu and Ricardo Baeza-Yates, "Scalability Challenges in Web Search Engines", The Information Retrieval Series, 2011.
- Web crawling
  - C. Olston and M. Najork: "Web Crawling", Foundations and Trends in Information Retrieval, 2010.
- Indexing
  - J. Zobel and A. Moffat, "Inverted files for text search engines", ACM Computing Surveys, 2006.
- Query processing
  - R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval (2nd edition), 2011.

Q&A