Architecture of a Search Engine

406.424 Internet Applications

Jonghun Park

jonghun@snu.ac.kr

Dept. of Industrial Eng.

Seoul National University

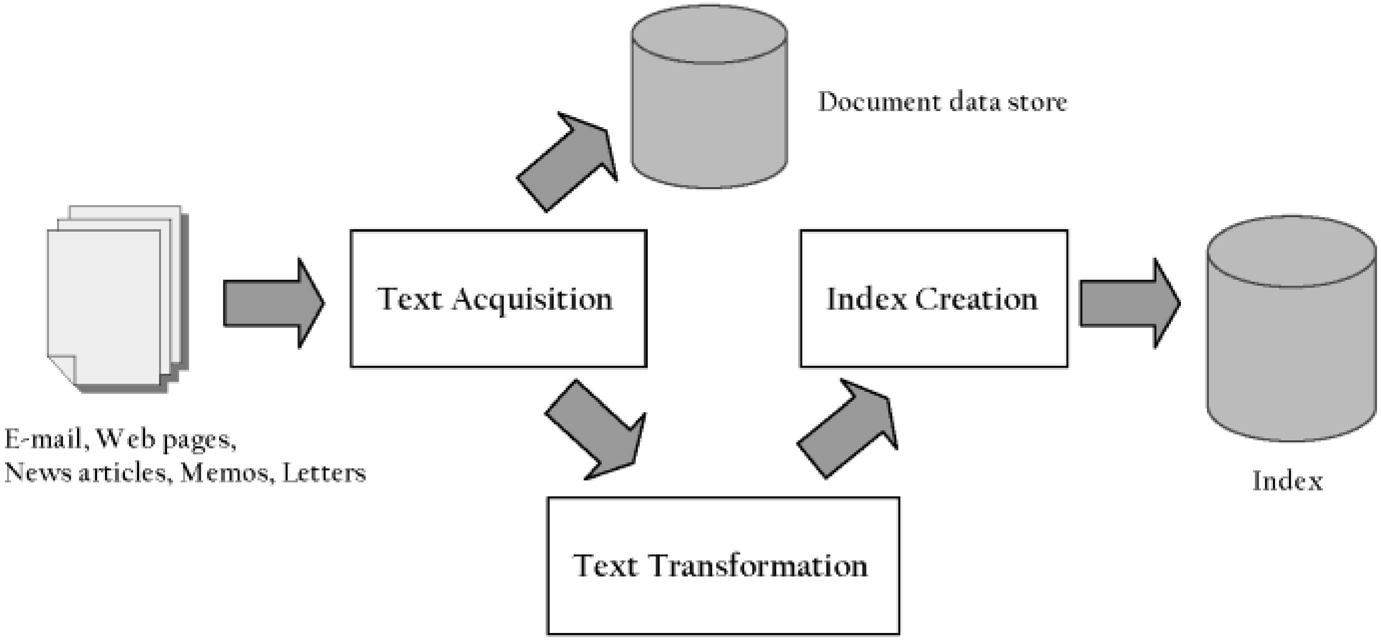
9/1/2010

search engine architecture

- software architecture consists of software **components**, the **interfaces** provided by those components, and the **relationships** between them
  - describes a system at a particular level of abstraction
- architecture of a search engine determined by 2 requirements
  - effectiveness: **quality** of results
  - efficiency: **response time** and **throughput**
- 2 major components
  - indexing process
  - query process

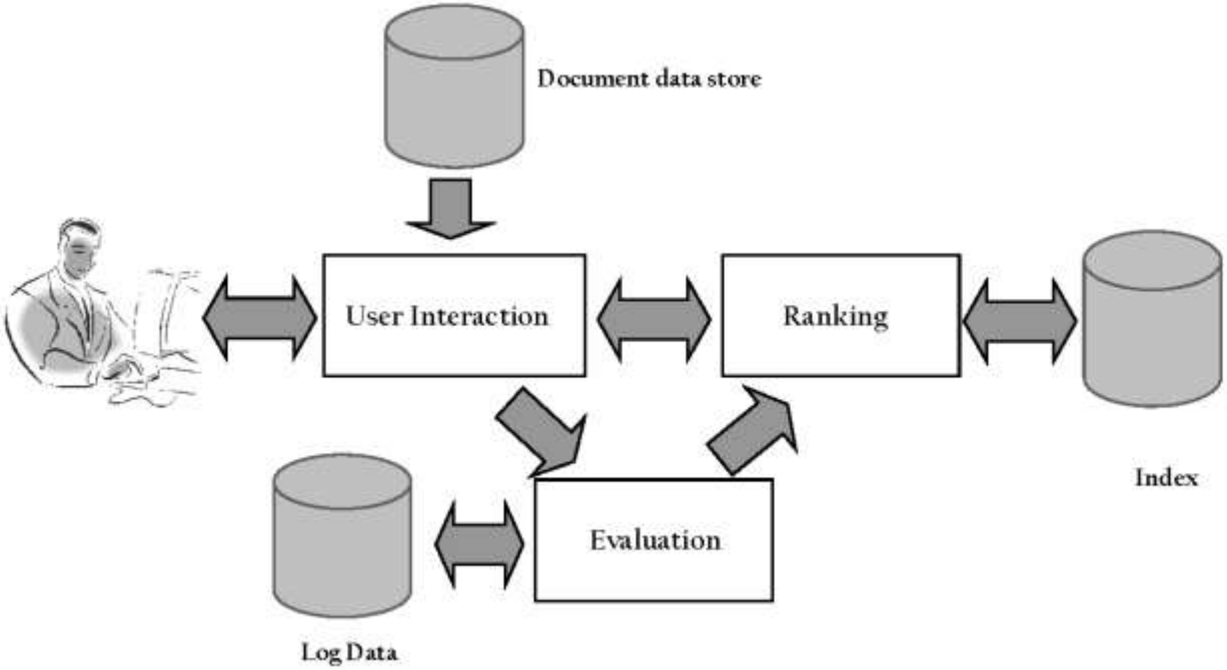
indexing process

- text acquisition: identifies and stores documents for indexing
- text transformation: transforms documents into **index terms** or **features**
  - e.g., word, phrase, name, date, link, …
- index creation: takes index terms and creates data structures (indexes) to support fast searching
  - **inverted index**: a list for every index term of the documents that contain that index term



query process

- user interaction: supports creation and refinement of query, display of results
- ranking: uses query and indexes to generate **ranked list of documents**
- evaluation: monitors and measures effectiveness and efficiency
    - primarily offline



text acquisition

- crawler
  - identifies and acquires documents for search engine
  - many types: web, enterprise, desktop
  - web crawlers **follow links** to find documents
    - must efficiently find huge numbers of web pages (**coverage**) and keep them up-to-date (**freshness**)
    - single site crawlers for **site search**
      - e.g., “site:snu.ac.kr” for google search
    - topical or focused crawlers for **vertical search**
      - needs classification techniques
  - document crawlers for enterprise and desktop search
    - follow links and scan directories

text acquisition

- feeds
  - a mechanism for accessing a real-time stream of documents
    - e.g., web feeds for news, blogs, video, radio, tv
  - RSS is common standard
    - RSS “reader” can provide new XML documents to search engine
    - cf: atom feed
- conversion
  - convert variety of documents into a consistent **text plus metadata** format
    - e.g. HTML, XML, Word, PDF, etc. → XML
  - convert text encoding for different languages
    - using a Unicode standard like UTF-8

text acquisition

- document data store
  - stores text, metadata, and other related content for documents
    - metadata: information about document such as type and creation date
    - other content includes links & anchor text
  - provides fast access to document contents for search engine components
    - e.g. **result list generation**
  - could use relational database system
    - more typically, a simpler, more efficient storage system is used due to huge numbers of documents

text transformation

- parser
  - processing the sequence of text **tokens** in the document to recognize **structural elements**
    - e.g., titles, links, headings, etc.
  - **tokenizer** recognizes “**words**” in the text
    - must consider issues like capitalization, hyphens, apostrophes, non-alpha characters, separators
    - e.g., 장동건결혼, on-line, Ph.D., …
  - both doc and query must be transformed into tokens in the same manner so that they can be easily compared
  - markup languages such as HTML, XML often used to specify structure
    - tags: e.g., <h2> Overview </h2>
    - tags and other control sequences must be treated appropriately

text transformation

- stopping
  - remove common words: non-topical, function words
    - e.g., “and”, “or”, “the”, “in”
  - some impact on efficiency and effectiveness
  - can be a problem for some queries
    - e.g., “to be or not to be”
- stemming
  - group words derived from a common stem
    - e.g., “computer”, “computers”, “computing”, “compute”
  - increase the likelihood that words used in queries and docs will match
  - usually effective, but not for all queries
    - e.g., “fish”, “fishing”

text transformation

- link analysis
  - makes use of **links** and **anchor text** in web pages
    - indexed separately from general text content
  - link analysis identifies popularity and community information
    - e.g., PageRank
  - anchor text can significantly enhance the representation of pages pointed to by links
  - significant impact on web search
    - less importance in other applications

text transformation

- information extraction
  - identify index terms that are more complex than single words
  - e.g., noun phrases: require POS (part-of-speech) tagging
  - e.g., **named entity** recognizers identify classes such as people, locations, companies, dates, etc.
- classifier
  - identifies class-related metadata for documents
    - i.e., assigns labels to documents for categorization
    - e.g., topics, reading levels, sentiment, genre
    - e.g., spam filtering, identifying non-content parts
  - use depends on application

index creation

- document statistics
  - gathers statistical information about words, features, and documents
    - e.g., term occurrences, positions, lengths of docs
  - used in ranking algorithm
- weighting
  - computes weights for index terms
  - used in ranking algorithm
  - needs to be done **during indexing process** as much as possible
  - **query dependent** vs. **query independent**
  - e.g., **tf.idf** weight
    - combination of term frequency in document and inverse document frequency in the collection

index creation

- inversion
  - core of indexing process
  - converts **document-term** information to **term-document** for creating inverted indexes
    - difficult for very large numbers of documents
  - format of inverted file is designed for fast query processing
    - must also handle index updates
    - compression used for efficiency
- index distribution
  - distributes indexes across multiple computers and/or multiple sites
  - essential for fast query processing with large numbers of documents
  - many variations: document distribution, term distribution, replication
  - indexing & query processing can be done **in parallel**

user interaction

- query input
  - provides **interface** and **parser** for query language
  - most web queries are very simple, other applications may use forms
    - keyword: a word that is important for specifying the topic of a query
    - a small # of operators: “ ”, |, …
  - query language used to describe more complex queries and results of query transformation
    - e.g., boolean queries, Indri and Galago query languages
    - e.g., AND, OR, NOT, proximity operator
    - similar to SQL language used in database applications

user interaction

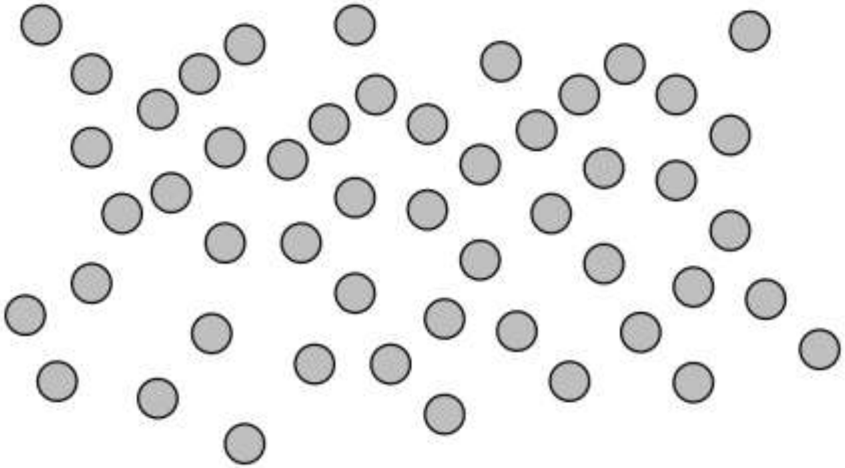
- query transformation
  - involves some of the same text transformation techniques used on doc text, including tokenizing, stopping, and stemming
  - **spell checking** and **query suggestion** provide alternatives to original query
  - **query expansion** and **relevance feedback** modify the original query with additional terms
    - relevance feedback: expands queries based on term occurrences in docs that are **identified as relevant by the user**

user interaction

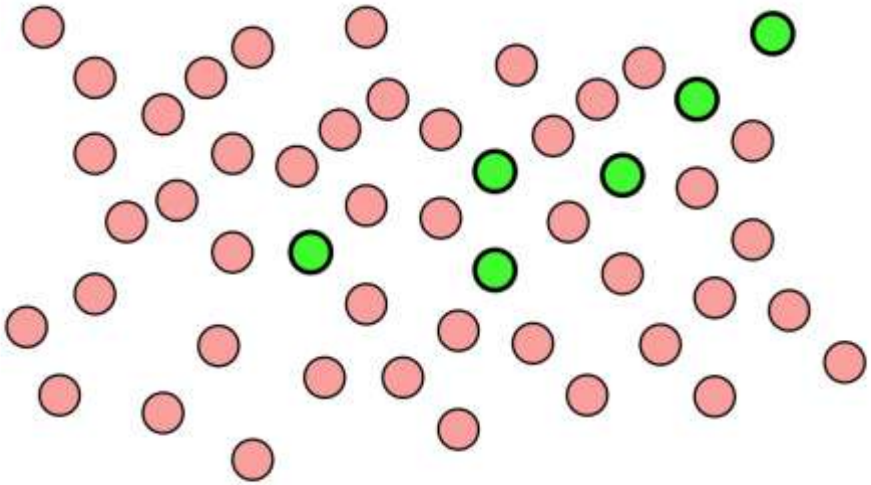
- results output
  - constructs the display of ranked documents for a query
  - generates **snippets** to show how queries match documents
  - highlights important words and passages
  - retrieves appropriate advertising in many applications
  - may provide clustering and other visualization tools

ranking

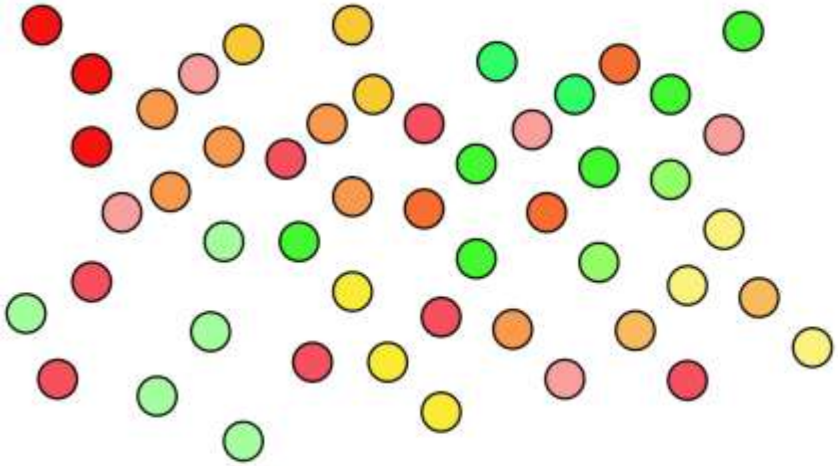
no retrieval



boolean retrieval



ranked retrieval



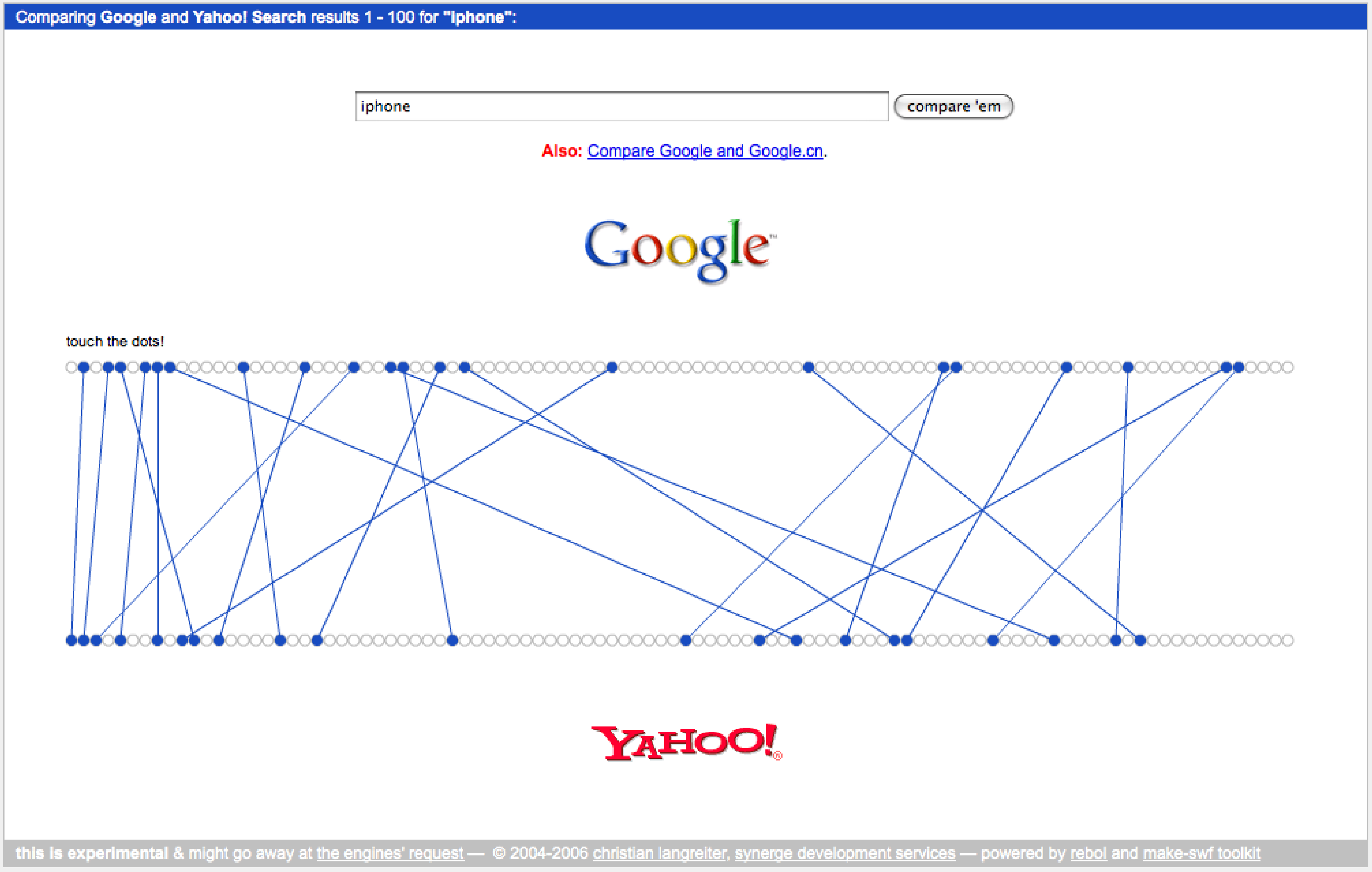
ranking issues

- scoring
  - calculates **scores for documents** using a ranking algorithm
  - a basic form of score is $\sum q_i d_i$
    - q_i and d_i are query and document term weights for term i
    - e.g., tf.idf weight
  - many variations of ranking algorithms and retrieval models
- performance optimization
  - designing ranking algorithms and the associated indexes for efficient processing
  - term-at-a time vs. document-at-a-time scoring
  - safe vs. unsafe optimizations
    - safe optimization guarantees that the scores calculated will be the same as the scores without optimization
- distribution
  - ranking can also be distributed
  - **query broker** distributes queries and assembles results
  - **caching** is a form of distributed searching

popular ranking features

- term matching
- term frequency
- inverse document frequency
- term proximity: e.g., white house vs. white …. house
- term location: title? heading?
- quality: authority, popularity, …
- web specific
  - url text, url length
  - anchor text

ranking experiments



Comparing **Google** and **Yahoo! Search** results 1 - 100 for **"iphone"**:

Also: Compare Google and Google.cn.

this is experimental & might go away at the engines' request — © 2004-2006 christian langreiter, synerge development services — powered by rebol and make-swf toolkit

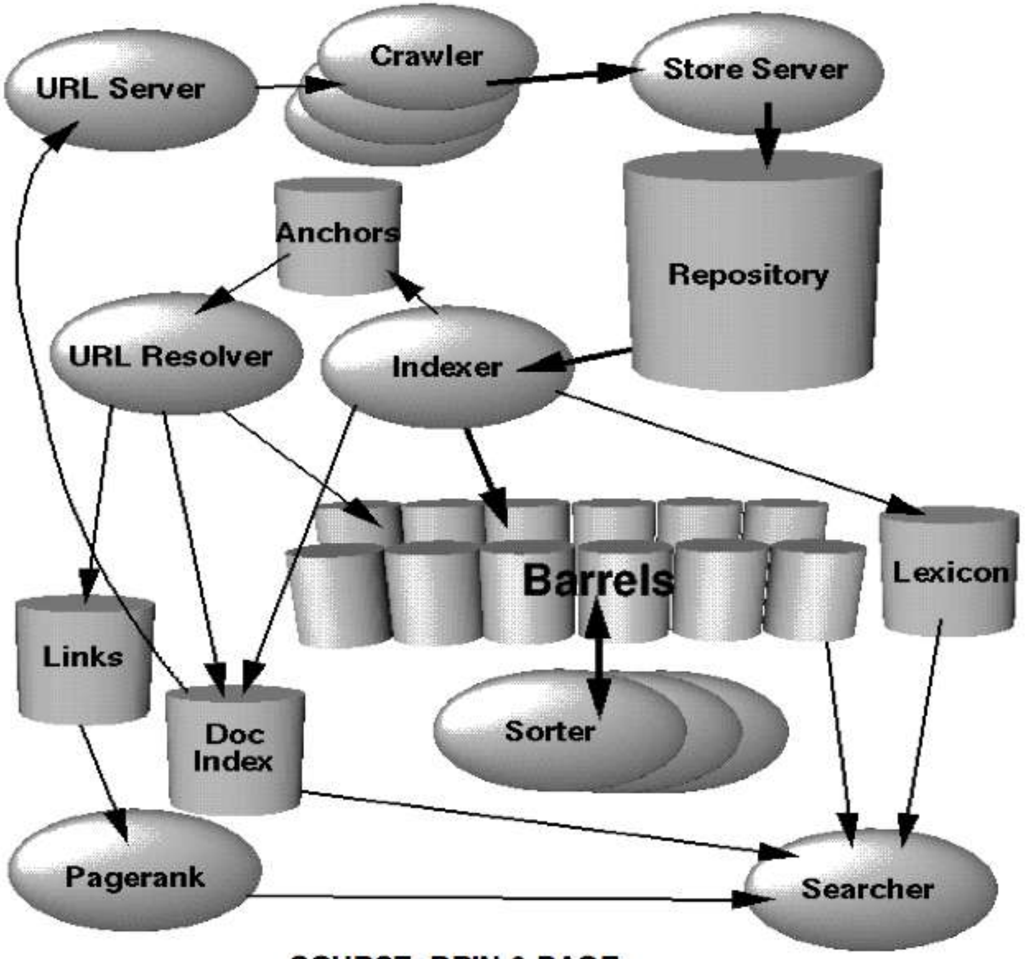
- http://www.langreiter.com/exec/yahoo-vs-google.html

evaluation

- logging
  - logging user queries and interaction is crucial for improving search effectiveness and efficiency
    - documents in a result list that are clicked on and browsed tend to be relevant
  - query logs, clickthrough data, dwell time can be used for query suggestion, spell checking, query caching, ranking, advertising search, and other components
- ranking analysis
  - measuring and tuning ranking effectiveness
  - emphasis on the **top-ranked documents**
- performance analysis
  - measuring and improving system efficiency
  - cf: round trip from US to China is 250ms while human “instantaneous” window is 150ms

Google search engine architecture (2001)

Google Search Engine Architecture



SOURCE: BRIN & PAGE

URL Server- Provides URLs to be fetched

Crawler is distributed

Store Server - compresses and stores pages for indexing

Repository - holds pages for indexing (full HTML of every page)

Indexer - parses documents, records words, positions, font size, and capitalization

Lexicon - list of unique words found

Barrels hold results of distribution sort

Anchors - keep information about links found in web pages

URL Resolver - converts relative URLs to absolute

Sorter - generates Doc Index

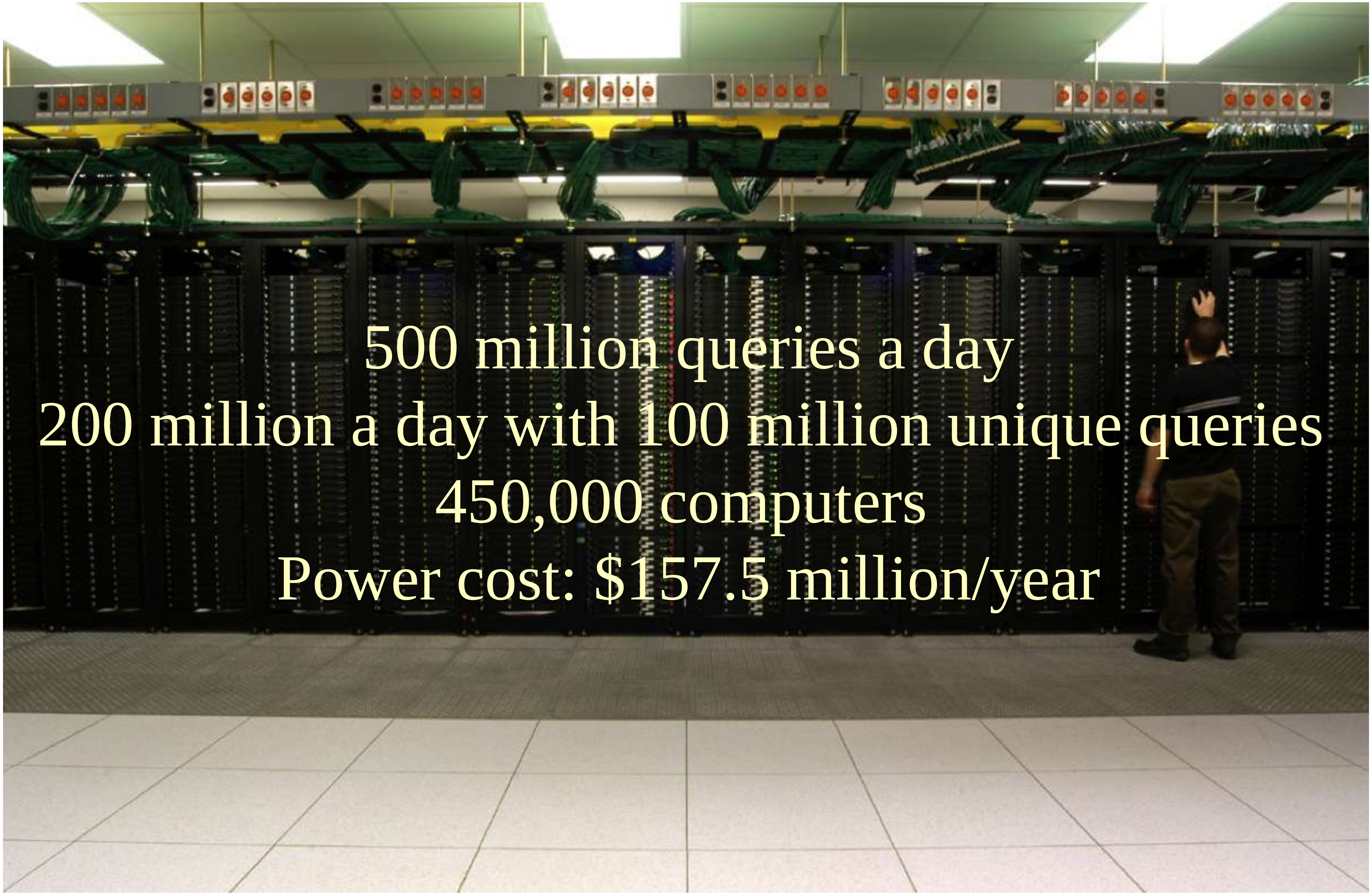
Doc Index - inverted index of all words in all documents (except stop words)

Links - stores info about links to each page (used for Pagerank)

Pagerank - computes a rank for each page retrieved

Searcher - answers queries

Google facts (as of 2006)



500 million queries a day
200 million a day with 100 million unique queries
450,000 computers
Power cost: $157.5 million/year