Notes 12: Distributed Information Retrieval
Web Search Engine

- Crawling
- Indexing
- Computing ranking features
- Serving queries

Crawling

Fetch the content of web pages

Crawling Issues

**Scope and freshness**
- Not enough space/time to crawl "all" pages
- Page importance, quality, and update frequency
- Site mirrors and (near) duplicate pages
- Dynamic content and crawler traps

**Load at visited web sites**
- Rules in robots.txt
- Limit number of visits per day
- Limit depth of crawl

![Diagram of web crawling process](image)
Crawling Issues

Load at crawler
- Variance of fetch latency/bandwidth

Parallelization and scalability
- Multiple agents
- Partitioning URL lists
- Communication between agents
- Recovering from agent failure

Crawl Partitioning

Requirements
- Each URL assigned to a single agent
- Locally computable URL-to-agent mapping
- Balanced distribution of URLs across agents
- Contravariance

Contravariance

Agent A    Agent B    Agent A    Agent B    Agent C
url₁, url₂, url₃  →  url₁, url₂, url₃
url₄, url₅       url₄, url₅
url₆

Contravariance

Agent A    Agent B    Agent C
url₁, url₂, url₃  →  url₁, url₂, url₃
url₄, url₅       url₄, url₅
url₆

Agent A    Agent B    Agent C
url₁, url₂, url₃  →  url₁, url₂, url₃
url₄, url₅       url₄, url₅
url₆

**URL Assignment**

**Consistent hashing**
- Hash function: URL → agent
- Each agent “replicated” k times
- Each replica mapped randomly on unit circle
- Mapping persistent across agent restarts
- Lookup: map URL on unit circle; find closest live replica

---

**Crawl Partitioning**

**Ideas**
- URL normalization
  - E.g., relative to absolute URL
- Host-based partitioning
  - Reduces communication between agents
  - Small vs. large hosts
- Geographic distribution
 Fault Tolerance

Repartitioning ✔
Permanent failures
- Recovering list of URLs to visit
- Checkpoints
- Communication logs
Transient failures
- Avoiding re-visiting URLs
  - E.g., before fetch, check with near neighbor agents

Web Search Engine

Crawling ✔
Indexing
- Computing ranking features
- Serving queries

Indexing

Build the term-document index

Architecture

Distributors

Web pages

Map

Intermediate runs

Reduce

Inverted index files

Indexers

Query servers
Indexing Issues

Index partitioning
Efficient query processing
  Query routing
  Result retrieval

Document Partitioning

Split the collection of documents

Advantages
Easy to add new documents
Load balanced
High processing throughput

Disadvantages
Communication with all query servers

Term Partitioning
**Term Partitioning**

Split the lexicon

**Advantages**
Reduced communication with query servers

**Disadvantages**
More processing before partitioning
Adding new documents is hard
Load balancing is hard
Processing parallelism is limited by query length

**Advanced Partitioning**

Topical partitioning using clustering
Documents clustered by term-similarity
Partitions made up of one or more clusters

Usage-induced partitioning
Queries extracted from logs
Documents clustered by query-similarity
Partitions made up of one or more clusters

**Web Search Engine**

- Crawling ✔
- Indexing ✔
- Computing ranking features
- Serving queries

**Ranking Feature Computation**

Parallel/distributed computation tasks
- Text/language processing
- Document classification/clustering
- Web graph analysis
Example: PageRank

Link-based global (query-independent) importance metric

Random surfer model
Start at a random page
With probability $d$, visit a new page following a random link on the current page
With probability $(1 - d)$, restart at a random page

$\Rightarrow$ PageRank score $\sim$ expected fraction of time spent at a page

PageRank Formula

$$p(x) = d \times \sum_{y \rightarrow x} p(y) / \text{out}(y) + (1 - d) / n$$

PageRank Algorithm

\[i = 0\]
\[p[i](x) = (1 - d) / n\]
repeat
\[i += 1\]
\[p[i](x) = (1 - d) / n\]
for all $y \rightarrow x$
\[p[i](x) += d \times p[i-1](y) / \text{out}(y)\]
until $|p[i] - p[i-1]| < \varepsilon$
PageRank Implementation

Two vectors, current and next

Initialize vectors
Iterate over all pages y, distribute PageRank from current(y) to
next(x) for all links y → x

current = next, re-initialize next

Go back to iteration over pages or stop

PageRank Distribution

MapReduce for each iteration

Map
Take <y, (current(y), edges(y))>
For each y → x in edges(y)
emit <x, current(y) / |edges(y)|>
Also emit <y, edges(y)>

Reduce
Take <x, val> and <x, edges(x)>
Sum (d × val) into next(x), add (1 - d) / n
Emit <x, (next(x), edges(x))>

Web Search Engine

Crawling ✔
Indexing ✔
Computing ranking features ✔
Serving queries
**Query Processing**

Locate, retrieve, process, and serve query results

**Serving Architecture**

Multiple sites connected by WAN
- Site = coordinator + servers + cache

Partitioning
- Parallel processing
- Distributed storage of data
- Based on index partitioning

Replication
- Availability
- Throughput
- Response time

**Serving Issues**

Routing the query
- To sites
  - E.g., identical sites + routing by dynamic DNS lookup
- Within sites

Merging the results
- Caching

**Routing** | **Merging**
--- | ---
Document partition | All servers | Results selected by servers; ranking by coordinator
Term partition | Servers containing query terms | Selection and ranking by coordinator
Caching

What to cache?
Query answers
Term postings

Caching

Query terms repeated more frequently than whole queries

What to cache?
Query answers
  → Faster response
Term postings ✔
  → More hits

Caching

Policy
Terms most frequent in queries
  → High hit ratio
Terms most frequent in documents
  → Requires more cache space (longer postings)
Summary

Crawling
- Partitioning: balancing and contravariance
  - Consistent hashing

Indexing
- Document, term, topical, and usage-induced partitioning

Computing ranking features
- PageRank with MapReduce

Serving queries
- Routing queries, merging results, and caching postings