

Peer-to-Peer – Crawling and Indexing

State-of-the-Art
&
New Aspects



Talk Outline

1. Some distributed, high-performance WebCrawler
2. Classifications and Measurements of parallel Crawlers
3. UBICrawler (totally distributed and high scalable system)
4. Apoidea (decentralized P2P architecture for crawling the www)
5. Main Requirements

1. Some distributed high-performance WebCrawler

Mercator (Compaq Research Center)

- ▶ scalable: designed to crawl the entire web
- ▶ extensible: designed in a modular way
 - Not really distributed (only an extended version)
 - Central coordination is used
 - Interesting: datastructures for content-seen-test (document fingerprint set), URL-seen-test (stored mostly on disk) and URL Frontier Queue
 - Performance: 77,4 million HTTP requests in 8 days (1999)

1. Some distributed high-performance WebCrawler

PolyBot (Polytechnic University NY)

Design and Implementation of a High-Performance Distributed Web Crawler

▶ Crawling System

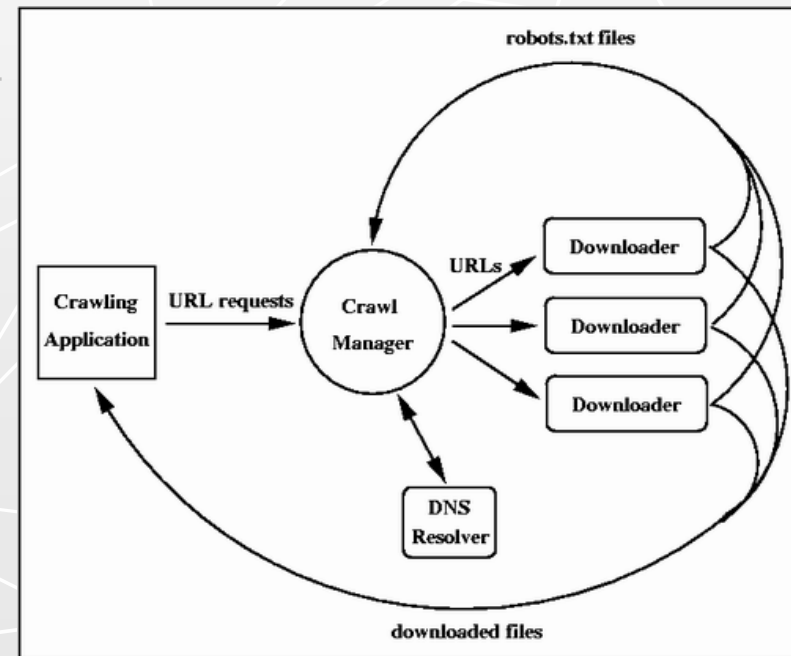
- Manager, downloaders, DNS-resolver

▶ Crawling Application

- Link extraction by parsing and URL-seen-checking

▶ Performance

- 18 days / 120 million pages /
 - 5 million hosts / 140 pages/second
- ▶ Components can be distributed on different systems -> distributed but not P2P



1. Some distributed high-performance WebCrawler

Further Prototypes:

WebRace (California / Cyprus)

WebGather (Beijing)

- ▶ Also a distributed crawler
- ▶ Not P2P

2. Classification of parallel Crawlers

Issues of parallel Crawlers:

- **overlap:** minimization of multiple downloaded pages
- **quality:** depends on the crawl strategy
- **communication bandwidth:** minimization

Advantages of parallel Crawlers:

- **scalability:** for large-scale web-crawls
- **costs:** use of cheaper machines
- **network-load dispersion and reduction:** by dividing the web into regions and crawling only the nearest pages

2. Classification of parallel Crawlers

A parallel crawler consists of multiple crawling processes communicating via local network (**intra-site parallel crawler**) or Internet (**distributed crawler**).

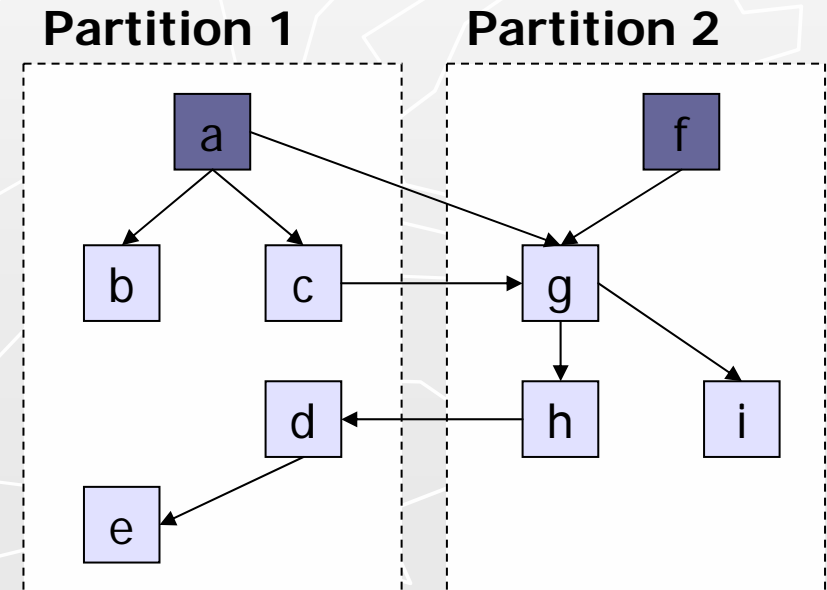
Coordination of the communication:

- 1. Independent:** no coordination, every process follows its extracted links
- 2. Dynamic assignment:** a central coordinator dynamically divides the web into small partitions and assigns each partition to a process
- 3. Static assignment:** web is partitioned and assigned without central coordinator before the crawl starts

2. Classification of parallel Crawlers

By using static assignment links from one partition to another (**inter-partition links**) there are three different modes:

1. **Firewall mode:**
a process does not follow any inter-partition link
2. **Cross-over mode:**
a process follows also inter-partition links and discovers also more pages in its partition
3. **Exchange mode:**
processes exchange inter-partition URLs; mode needs communication



2. Classification of parallel Crawlers

If exchange mode is used, we have to reduce the communication by the following techniques:

- **Batch communication:** every process collects some URLs and send them in a batch
- **Replication:** the k most popular URLs are replicated at each process and are not exchanged (previous crawl or on the fly)

Some ways to Partition the web:

- **URL-hash based:** many inter-partition links
- **Site-hash based:** reduces the inter partition links
- **Hierarchical:** .com domain, .net domain ...

2. Classification of parallel Crawlers

Evaluation metrics (1):

1. **Overlap:** $Overlap = \frac{N - I}{N}$

N: total number of pages downloaded by overall crawler
I: number of unique pages

▶ minimize the overlap

2. **Coverage:** $Coverage = \frac{U}{I}$

U: total number of pages overall crawler has to download

▶ maximize the coverage

2. Classification of parallel Crawlers

Evaluation metrics (2):

3. **Communication Overhead:** $Overhead = \frac{M}{P}$

M: number of exchanged messages (URLs)

P: number of downloaded pages

- ▶ minimize the overhead

4. **Quality (importance metrics):** $Quality = \frac{|A_N \cap P_N|}{P_N}$

P_N: set of the *N* most important pages

A_N: set of the *N* downloaded pages of the actual crawler

- ▶ maximize the coverage
- ▶ *backlink count / oracle crawler*

2. Classification of parallel Crawlers

Comparison of the three crawling modes

	Coverage	Overlap	Quality	Communication
Firewall	Bad	Good	Bad	Good
Cross-over	Good	Bad	Bad	Good
Exchange	Good	Good	Good	Bad

3. UBI Crawler

Scalable fully distributed web crawler:

- platform-independant (Java)
- fault-tolerant
- effective assignment function for partitioning the web
- complete decentralization (no central coordination)
- scalability

3. UBI Crawler

Design requirements and goals:

- **Full distribution:** identical agents / no central coordinator
- **Balanced locally computable assignment:**
 - each URL is assigned to one agent
 - every agent can compute the responsible agent locally
 - distribution of URLs is balanced
- **Scalability:** number of crawled pages per second and agent should be independent of the number of agents
- **Politeness:** parallel crawler should never fetch more than one page at a time from a given host
- **Fault tolerance:**
 - URLs are not statically distributed
 - distributed reassignment protocol not reasonable

3. UBI Crawler

Assignment Function

A : set of agent identifiers

L : set of alive agents

$$L \subseteq A$$

m : total number of hosts

assignment function δ delegates for each nonempty set L of alive agents and for each host h the responsibility of fetching h to a agent:

$$\delta_L(h) \in L$$

Requirements:

- **Balancing:** each agent should be responsible for approximately the same number of hosts:

$$\left| \delta_L^{-1}(a) \right| \approx \frac{m}{|L|}$$

- **Contravariance:** if the number of agents grows, the portion of the web crawled by each agent must shrink:

$$L \subseteq L' \Rightarrow \delta_L^{-1}(a) \supseteq \delta_{L'}^{-1}(a)$$

$$L \subseteq L' \wedge \delta_{L'}(h) \in L \Rightarrow \delta_{L'}(h) = \delta_L(h)$$

3. UBI Crawler

Consistent Hashing as Assignment Function

- **Typical hashing:** adding a new bucket is a catastrophic event
- **Consistent hashing:** each bucket is replicated k times and each replica is mapped randomly on the unit circle
- **Hashing a key:** compute a point on the unit circle and find the nearest replica
- In Our case:
 - ▶ buckets = agents
 - ▶ keys = hosts

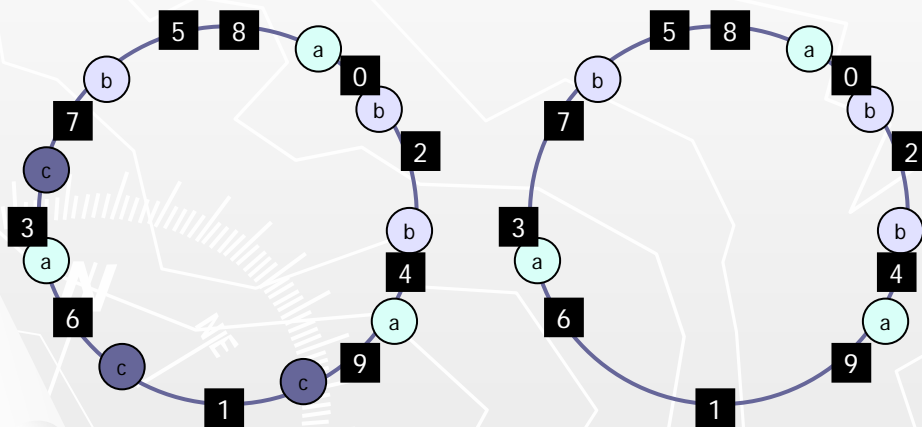
▶ balancing and contravariance

- Set of replicas is derived from a random number generator (Mersenne Twister) seeded with the agent identifier
 - ▶ identifier-seeded consistent hashing
- **Birthday-paradoxon:** already assigned replicas → choose another identifier

3. UBI Crawler

Example of Consistent Hashing:

$L' = agents' = \{a,b,c\}$, $L = agents = \{a,b\}$, $k = 3$, $hosts = \{0,1,\dots,9\}$



Contravariance:

$$L \subseteq L' \Rightarrow \delta_L^{-1}(a) \supseteq \delta_{L'}^{-1}(a)$$

$$L \subseteq L' \wedge \delta_{L'} \in L \Rightarrow \delta_{L'}(h) = \delta_L(h)$$

Balancing:

Hash function and random number generator

$$\delta_L^{-1}(a) = \{4, 5, 6, 8\}$$

$$\delta_L^{-1}(b) = \{0, 2, 7\}$$

$$\delta_L^{-1}(c) = \{1, 3, 9\}$$

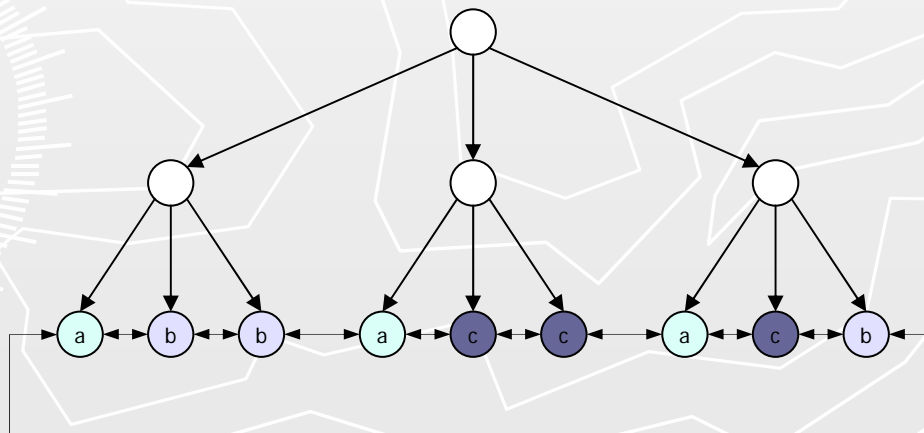
$$\delta_{L'}^{-1}(a) = \{1, 4, 5, 6, 8, 9\}$$

$$\delta_{L'}^{-1}(b) = \{0, 2, 3, 7\}$$

3. UBI Crawler

Implementation of Consistent Hashing

- Unit circle is mapped on the whole set of integers
- All replicas are stored in a balanced tree
- Hashing hosts in logarithmic time of alive agents
- Leaves are kept in a doubly linked chain to search the next nearest replica very fast
- Number of replicas depends on the capacity of hardware



3. UBI Crawler

Performance Evaluation of UBI Crawler:

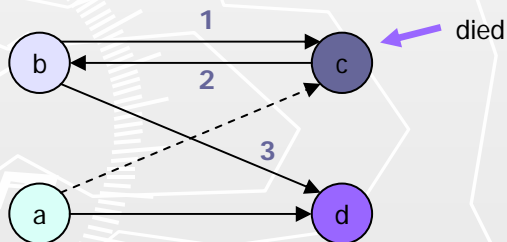
- ▶ **Degree of distribution:** distributed / any kind of network
- ▶ **Coordination:** dynamic, but without central coordination
 - ▶ *distributed dynamic coordination*
- ▶ **Partitioning technique:** host-based hashing / consistent hashing
- ▶ **Coverage:** optimal coverage of 1
- ▶ **Overlap:** optimal overlap of 0
- ▶ **Communication overhead:** independent of the number of agents / depends only of the number of crawled pages
- ▶ **Quality:** only BFS implemented
 - ▶ *BFS tends to visit high-quality pages first*

3. UBI Crawler

Fault tolerance of UBI Crawler:

Up to now: no metrics for estimating the fault tolerance of distributed crawlers

- Every agent has its own view of the set of alive agents (views can be different) but two agents will never dispatch hosts to two different agents.



- Agents can be added dynamically in a self-stabilizing way

3. UBI Crawler

Conclusions:

- UBI Crawler is the first completely distributed crawler with identical agents
- Crawl performance depends on the number of agents
- Consistent Hashing completely decentralizes the coordination logic
- **But:** not really high-scalable
- **But:** no concepts to realize a distributed search engine
 - UBI Crawler only used for studies of the web (African web)
- **But:** no P2P Routing Protocol (Chord, CAN...)

4. Apoidea

Decentralized P2P model for building a web crawler

- ▶ Design Goals
 - Decentralized system
 - Self-managing and robust
 - Low resource utilization per peer

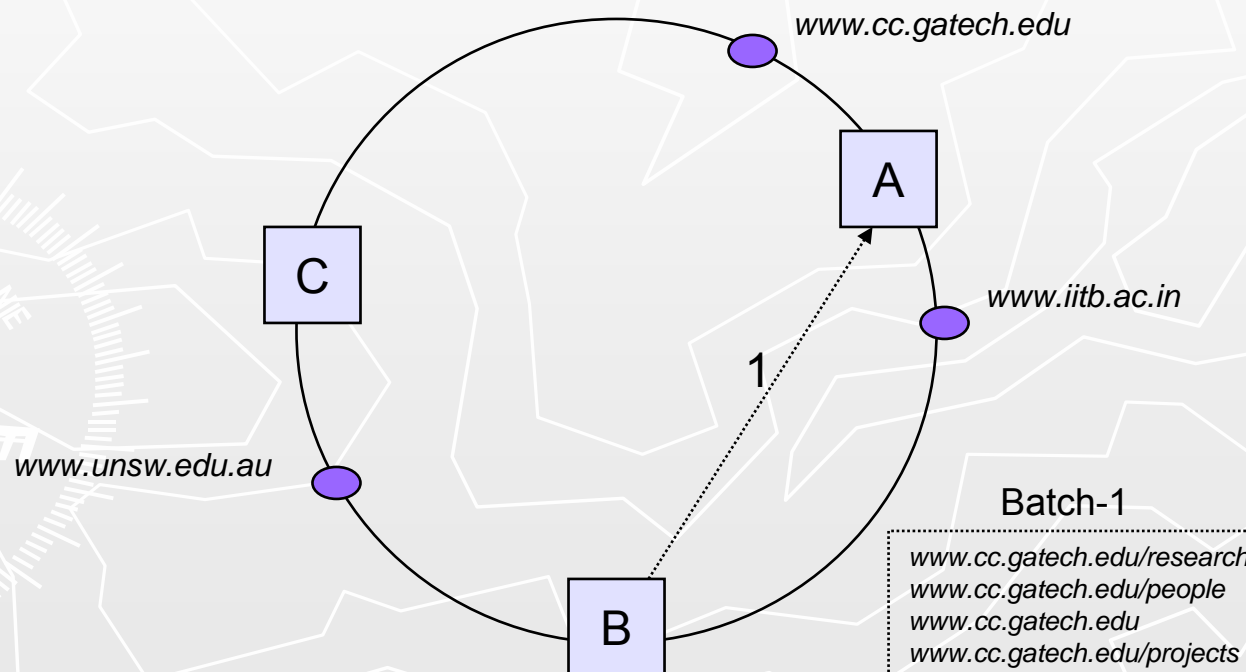
- ▶ Challenges
 - Division of labor
 - Duplicate tests

- ▶ Use of DHT (Chord)
 - Bounded number of hops
 - Guaranteed location of data
 - Handling peer dynamics

4. Apoidea

Division of labor

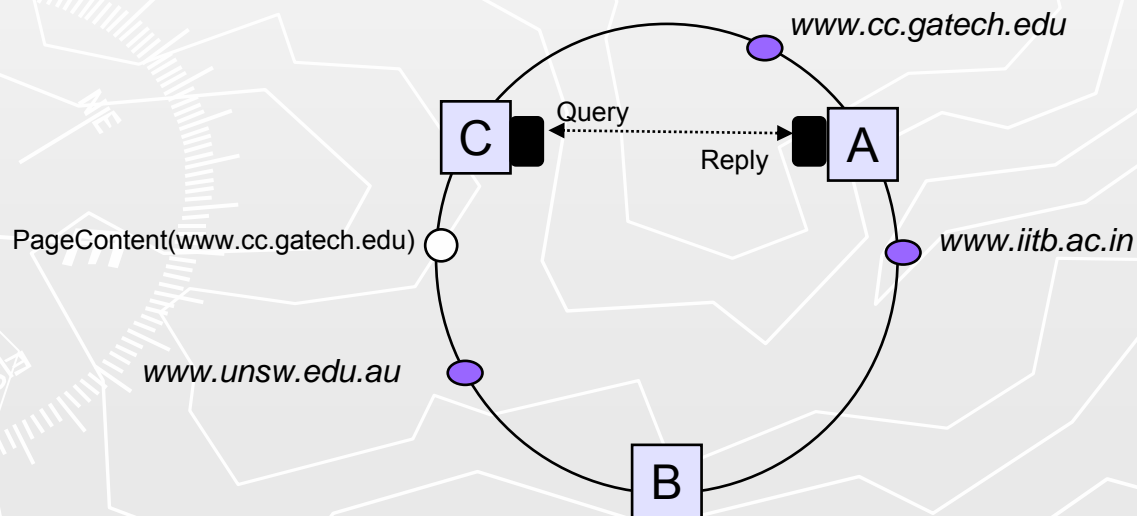
- ▶ Each peer responsible for a distinct set of URLs
- ▶ Site-Hash-based



4. Apoidea

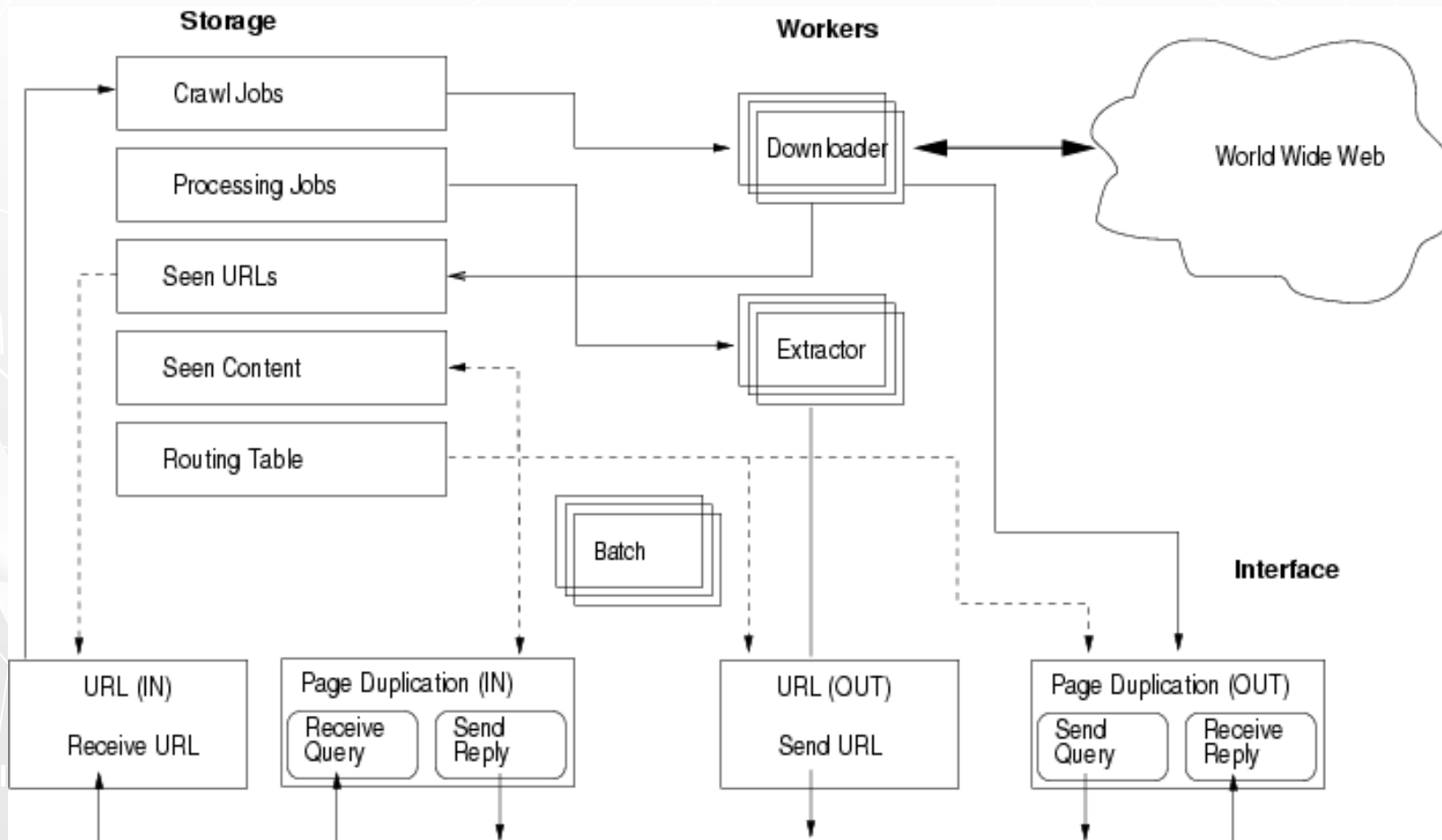
Duplicate Tests

- ▶ URL duplicate detection
 - Only the responsible peer needs to check for URL duplication
- ▶ Page content duplicate detection
 - Independent hash of the page content
 - Unique mapping of the content-hash to a peer



4. Apoidea

Per Peer Architecture



4. Apoidea

Data Structures

▶ Bloom Filters

- Efficient way to answer membership queries
- Assuming 4 billion pages, size of bloom filter = 5 GB
 - ▶ Impractical for a single machine to hold in memory
- Apoidea distributes the bloom filter
 - ▶ For 1000 peers, memory required per peer = 5 MB

▶ Per domain bloom filters

- Easy to transfer information to handle peer dynamics

5. Main Requirements

- P2P-like:
 - ▶ Identical agents & no central coordinator
- High scalability (data structures, routing)
- Dynamic assignment of host to peers (in a balanced way)
- Modular design with arbitrary P2P Lookup-System (Chord, CAN,...)
- No Overlap, low communication overhead, maximal coverage
- Fault tolerance (leaving and joining peers)
- Extension to a distributed search engine:
 - ▶ Decentralized index structures
 - ▶ Decentralized query processing

Literature

- **Apoidea:** A decentralized Peer-to-Peer Architecture for crawling the world wide web
 - ▶ Singh, Srivatsa, Liu, Miller (Atlanta)
- **UbiCrawler:** a scalable fully distributed web crawler
 - ▶ Boldi, Codenotti, Santini, Vigna (Italy)
- **Parallel Crawlers**
 - ▶ Cho, Garcia-Molina (Los Angeles, Stanford)
- **PolyBot:** Design and implementation of a high-performance distributed web crawler
 - ▶ Shkapenyuk, Suel (New York)
- **Mercator:** A scalable extensible Web Crawler
 - ▶ Heydon, Najork (Compaq)

Thank You!

Questions

