Parallel Crawlers

Junghoo Cho, Hector Garcia-Molina

Proceedings of the 11th International Conference on World Wide Web. ACM, 2002.

Efficient URL Caching for WWW Crawling

Andrei Broder, Marc Najork, and Janet L. Wiener

Proceedings of the 12th International Conference on World Wide Web. ACM, 2003.

Presented by Matt Chaney
CS 834 - Presentation 1

What is a Web Crawler?

1. Downloads web pages

2. Extracts links from pages

3. Builds a view of the web

Designs of the time of this paper (2003) were single-threaded

Parallelize to maximize data throughput

One crawl → many C-procs

Advantages of Parallel Architecture

1. Scalability
2. Network-load dispersion
3. Network-load reduction

Parallelization Issues

Coverage - How much of the web is downloaded per crawl?

Overlap - Don’t re-download the same page twice

- Parallel crawler may find same URL more than once

Quality - Don’t download poor pages, only “good” ones

- Parallel crawler may not see “big picture”; calculate quality well

Communication overhead - Minimize extra book-keeping work

Parallel Architecture

Coordination between C-procs

- Independent - No coordination, just download the pages man

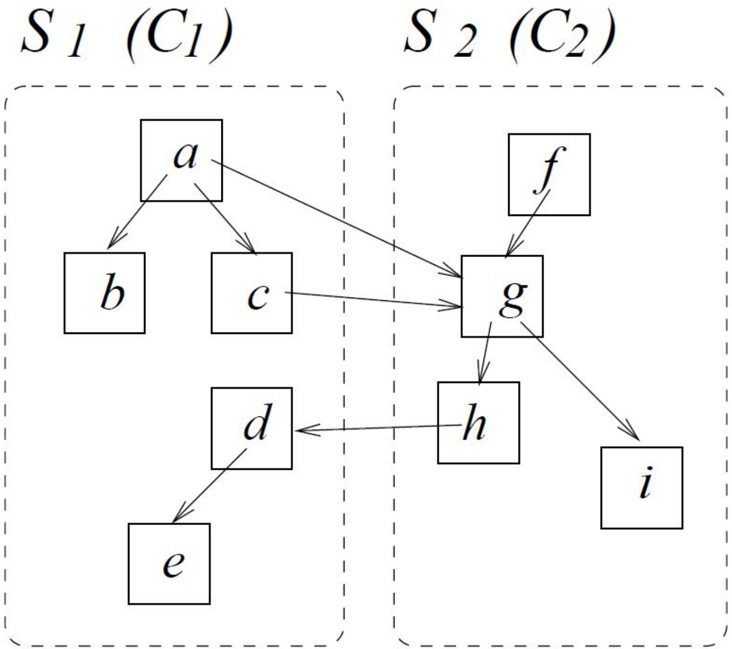
- Dynamic assignment - Use central controller to partition web so each C-proc has a slice to work on

- **Static assignment** - Web partitioned before crawl begins
    - Inter-Partition links can be an issue

Static Assignment

Handling Inter-partition links

- Firewall
  - \+ No communication overhead, no overlap
  - − Possibly missed pages, poor quality
- Cross-over
  - \+ No communication overhead, good coverage
  - − Overlap possible, still bad quality
- **Exchange**
  - \+ No overlap, good coverage, high quality
  - − *Some* communication overhead required



Exchange Mode

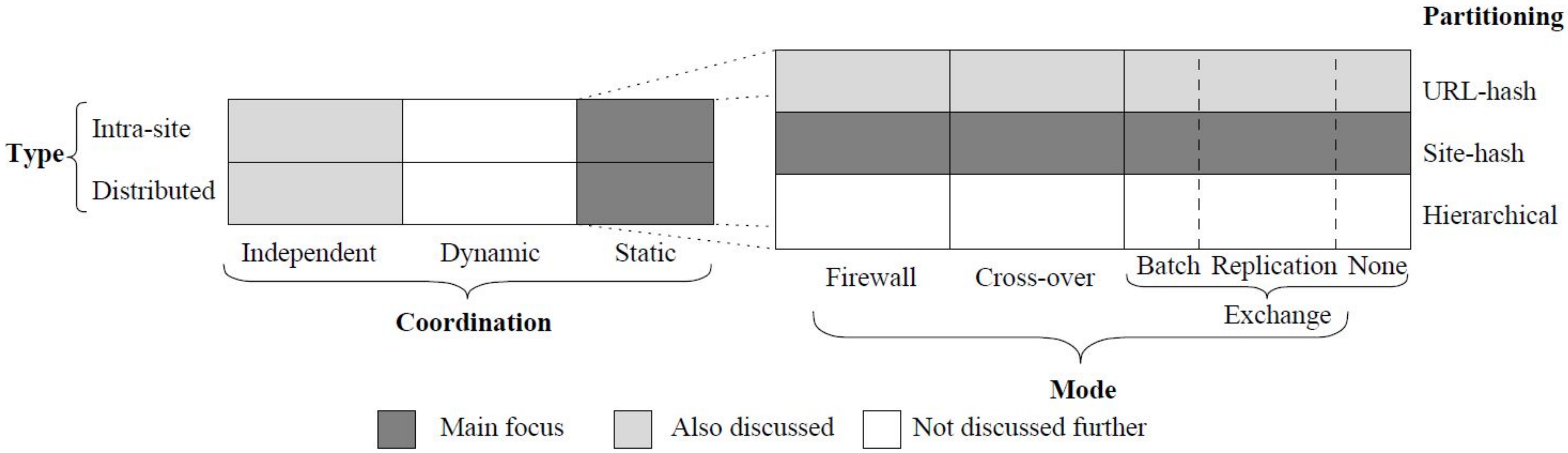
- Batch communication - wait until many inter-partition links are discovered and send out as a group
    - Reduced network load

- Replication - store *n* most popular URLs in each C-proc and ignore them when encountered (unless they are in your partition)

Partitioning Methods

- URL-hash - Entire URL hashed and assigned to a C-proc
  - http://www.homestarrunner.com/sbemail.html
- **Site-hash** - Based on host domain
  - www.cnn.com
  - www.odu.edu
- Hierarchical
  - www.*.com
  - www.*.net

Summary

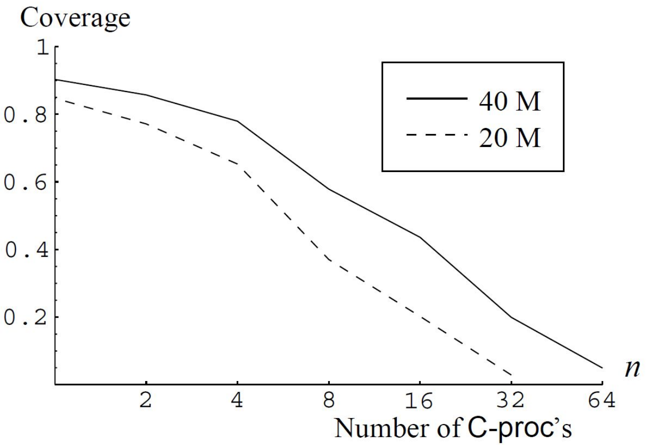

Partitioning
URL-hash
Site-hash
Hierarchical
Type
Intra-site
Distributed
Independent
Dynamic
Static
Coordination
Firewall
Cross-over
Batch
Replication
None
Exchange
Mode
Main focus
Also discussed
Not discussed further

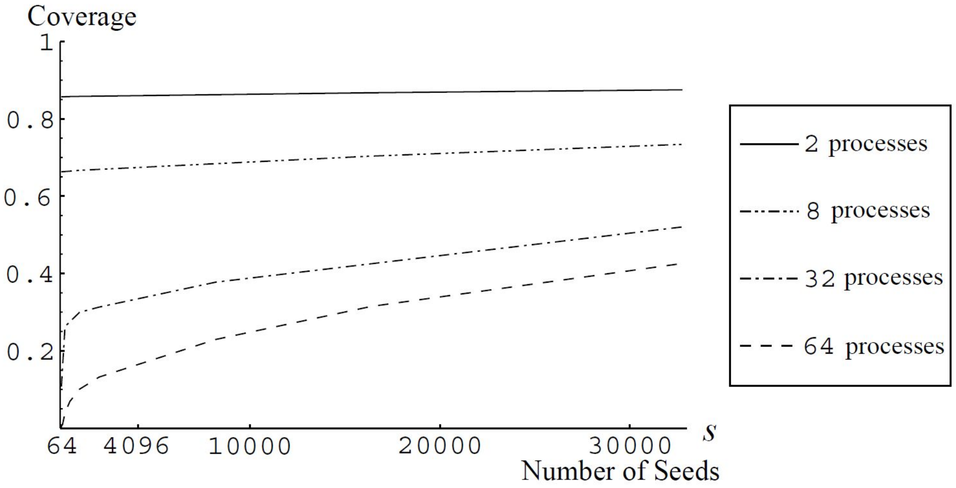
Experiment Description

- 40 million websites
- Downloaded with Stanford WebBase crawler in 1999 over a 2 week period
- Seeded with Open Directory (www.dmoz.org) URLs
- Totaled around 1 million seed pages

Firewall Mode Evaluation

- Coverage $$\frac{\#\ actually\ downloaded}{\#\ download\ required}$$



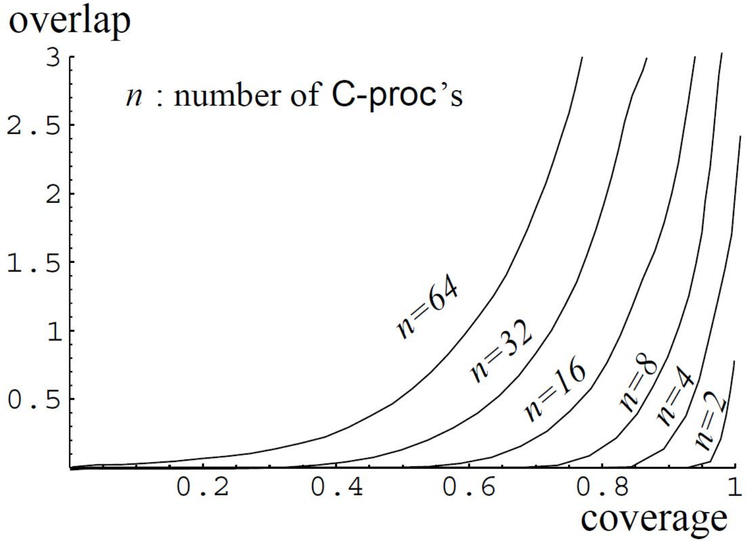


Crossover Mode - Overlap vs Coverage

- Overlap $$\frac{\#\ \textit{total pages downloaded} - \#\ \textit{unique pages downloaded}}{\#\ \textit{unique pages downloaded}}$$

- Coverage

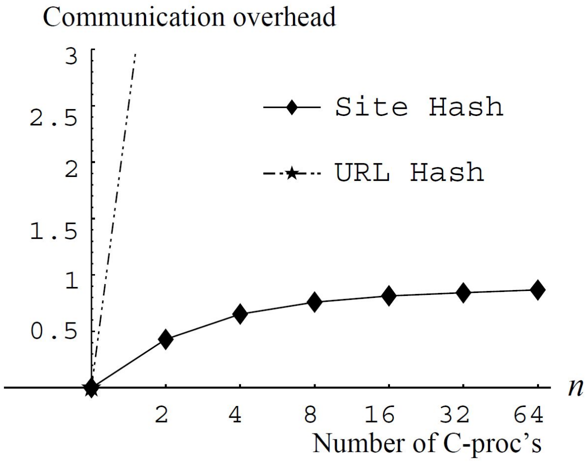
$$\frac{\#\ \textit{actually downloaded}}{\#\ \textit{download required}}$$

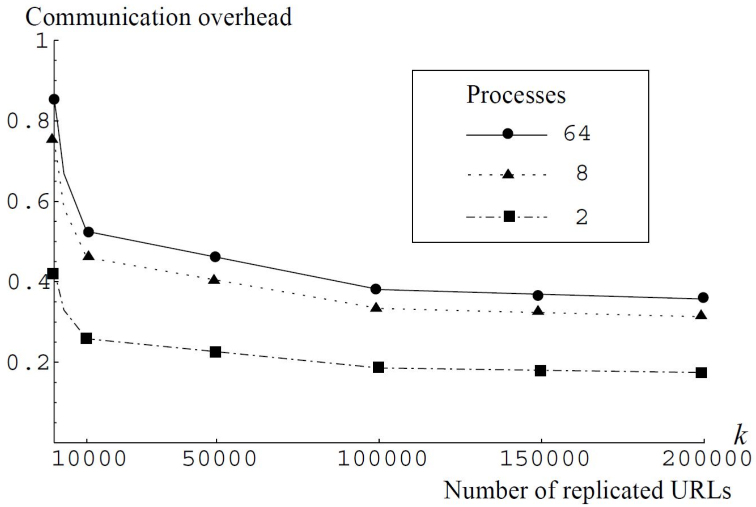


Exchange Mode

- Communication Overhead

$$\frac{\#\ inter\text{-}partition\ links\ exchanged}{\#\ pages\ downloaded}$$

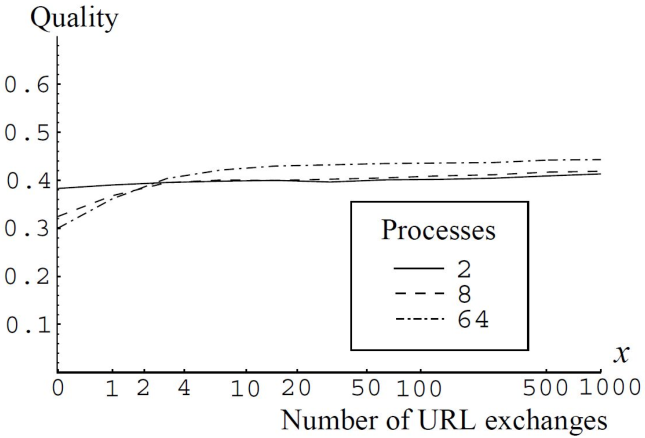


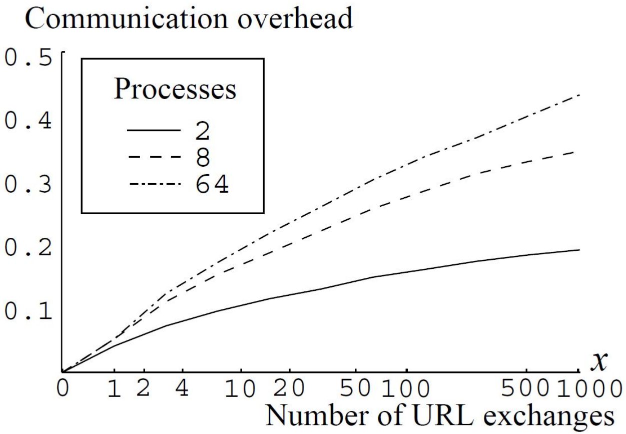


Exchange Mode

- Quality

$$\frac{|\#\ \textit{actually downloaded} \cap \#\ \textit{perfect downloaded}|}{|\#\ \textit{perfect downloaded}|}$$





Conclusions

- Small number of crawlers (less than 4) in firewall mode provide good coverage
  - If we need more than 4 crawlers firewall mode will lose coverage
  - If quality matters firewall mode is a bad choice
- URL exchange consumes around 1% bandwidth for communication overhead
- Exchange mode maximizes quality, even with exchanging backlink messages less than 100 times during entire crawl
- Replicating URLs reduced communication overhead by about 40% while not significantly affecting overhead

Recall Web Crawler Algorithm

1. Fetch a page
2. Parse it to extract all URLs
3. For each unique, not-yet-encountered URL from 2, repeat steps 1-3.

Why caching?

- Non-uniformity of requests
  - Not all web pages were created equal

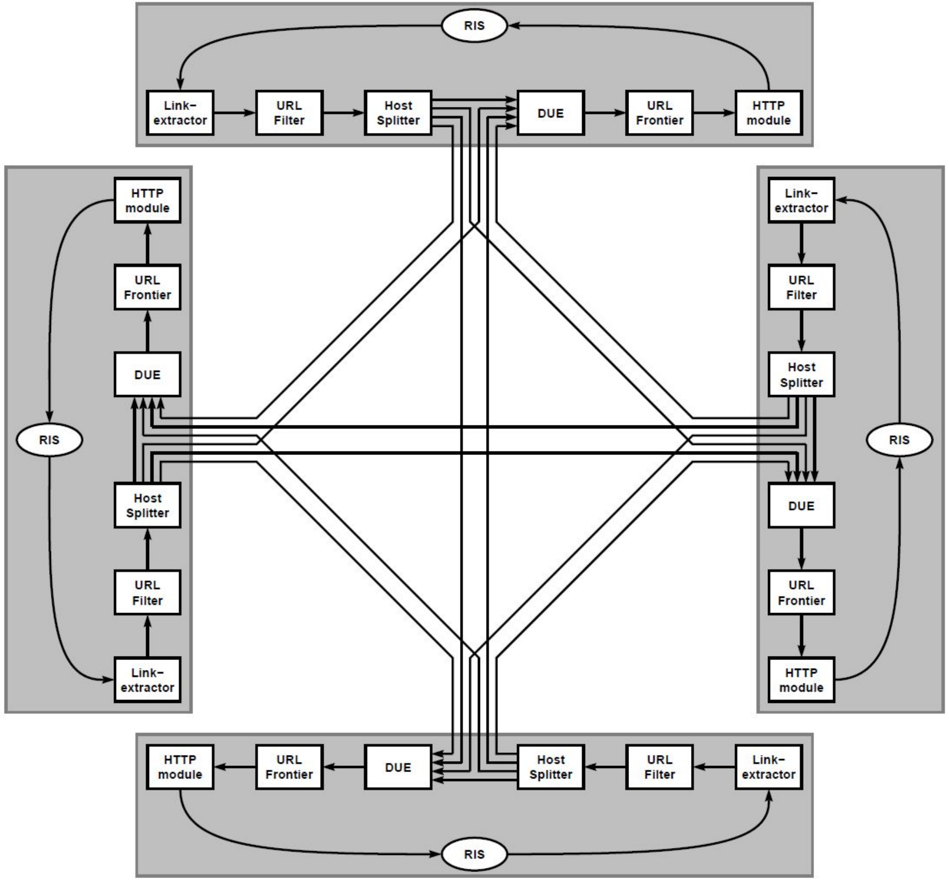
- Temporal correlation / Locality of Reference
  - Current requests are likely to be needed again soon
  - Links on same page likely to be repeated (30% are duplicates)

- Cache Strategy
  - Store popular pages
  - Store recent pages

Mercator

- **Parallel** crawler
- **Host-based** URL hashing
- **Exchange** mode to maximize coverage
- Exchanges in **batches** to reduce communication overhead
- Minimizes overlap with a **cached URL collection**

Mercator


RIS
Link-extractor
URL Filter
Host Splitter
DUE
URL Frontier
HTTP module
RIS
HTTP module
URL Frontier
DUE
Host Splitter
URL Filter
Link-extractor
Link-extractor
URL Filter
Host Splitter
DUE
URL Frontier
HTTP module
RIS
HTTP module
URL Frontier
DUE
Host Splitter
URL Filter
Link-extractor
RIS

Eviction Selection

- *Infinite size cache*
- *Clairvoyant (MIN)*
    - Evict the request that is the furthest away in time
- Least Recently Used (LRU)
    - Evict the oldest item in the cache
- CLOCK
    - Array with mark-bit, pointer to spot following last eviction
    - Got a new item?
        - In cache ➔ mark it
        - Not in cache ➔ follow items, turning mark bits off, until unmarked item is found
- Random replacement
- Static
    - Pre-load cache with *n* most popular URLs

Algorithm Implementations

- In all cases a hash table is used
- Separate structure used for each cache item for eviction victim selection

Implementation	Data Structure
Random	List
Clock	List + Clock handle + mark bit / item
LRU	Heap sorted on last used time
Static	None; static doesn’t evict
MIN	Heap sorted on next use time

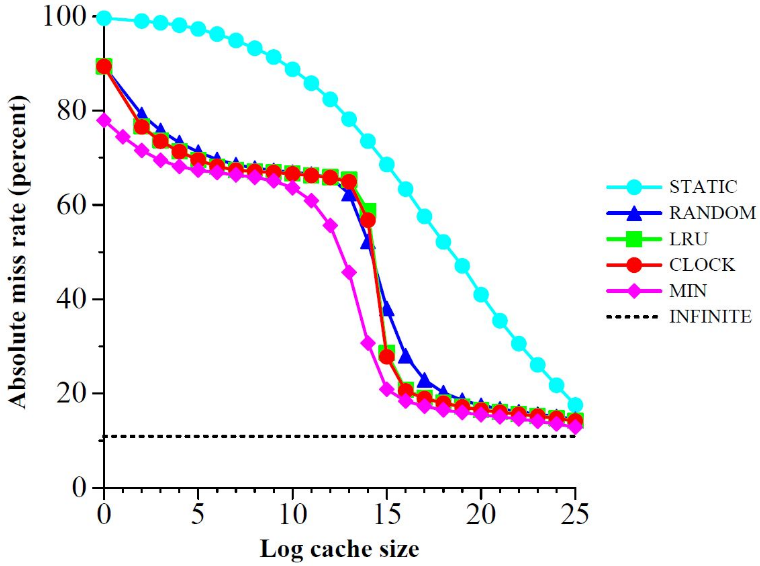
Experiment Description

- Four Compaq XP1000 workstations
  - 667 Mhz processor
  - 1.5 GB RAM
  - 144 GB HDD
  - 100 Mbit/sec Ethernet
- Ran from July 12 to September 3, 2002
- 1.04 billion download attempts, 784 million successful
- 429 million of these were web pages
  - 26.83 billion links
  - Average 62.55 links/page
  - Median was only 23

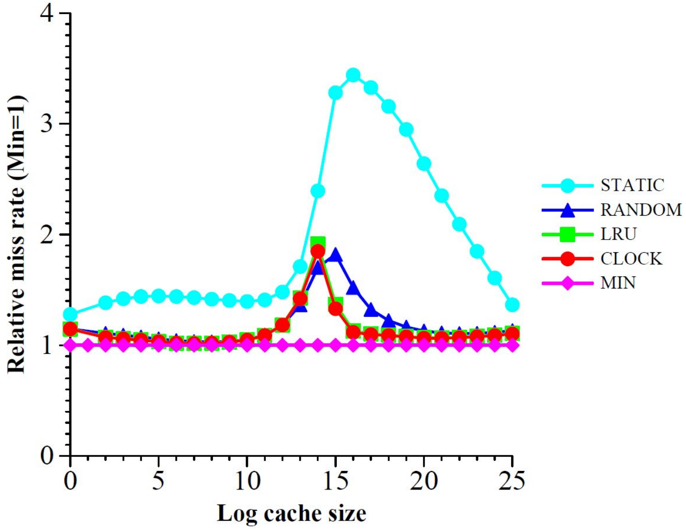
Datasets

- *Full Trace* - All URLs assigned to a particular machine
- *Cross Sub-Trace* - URLs discovered by one machine that another machine is responsible for
- Each trace was fed into each of the 5 mentioned implementations
    - Infinite, Clairvoyant (MIN), LRU, CLOCK, and Static
- In total, about 1,800 separate experiments run

Results



100
80
60
40
20
0
0
5
10
15
20
25
Absolute miss rate (percent)
Log cache size
STATIC
RANDOM
LRU
CLOCK
MIN
INFINITE

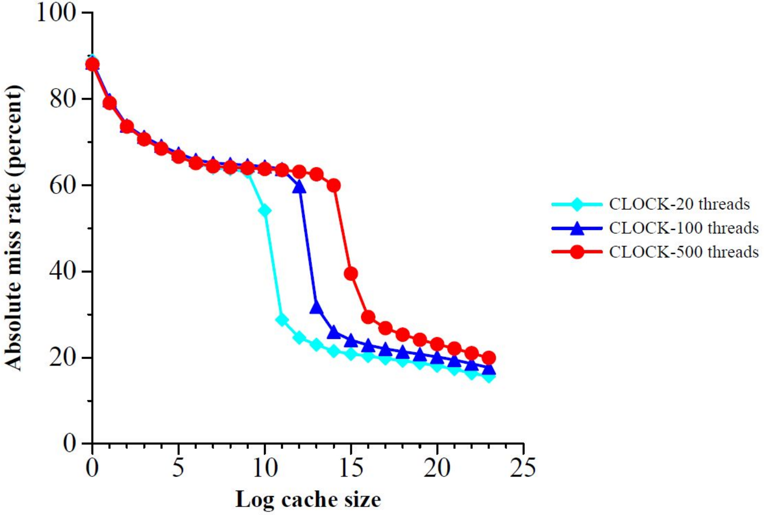


4
3
2
1
0
0
5
10
15
20
25
Relative miss rate (Min=1)
Log cache size
STATIC
RANDOM
LRU
CLOCK
MIN

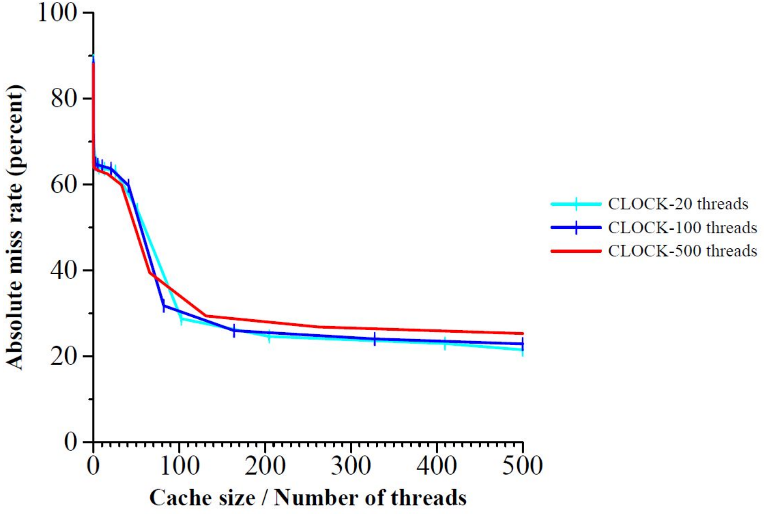
Conclusions

- URL caching is very effective
  - A cache of roughly 50,000 entries achieved a hit rate of 80%
  - There is a “sweet spot”
    - A significantly smaller cache is not effective
    - A significantly larger cache does not provide much tangible benefit
- Locality of Reference confirmed
  - Static showing generally terrible results
  - LRU and CLOCK showing good results
- Critical Region observed for cache sizes from 2^{14} to 2^{18}

Results



Absolute miss rate (percent)
100
80
60
40
20
0
0
5
10
15
20
25
Log cache size
CLOCK-20 threads
CLOCK-100 threads
CLOCK-500 threads



Absolute miss rate (percent)
100
80
60
40
20
0
0
100
200
300
400
500
Cache size / Number of threads
CLOCK-20 threads
CLOCK-100 threads
CLOCK-500 threads