Yioop Full Historical Indexing In Cache Navigation

Akshat Kukreti

Agenda

- Introduction
- History Feature
- Cache Page Validation Feature
- Conclusion
- Demo

Introduction

- Project goals
  - History feature for enabling access to all versions of cached pages
  - Cache page validation feature using ETags and Expires
    - Experiment to determine effect on crawl speed and bandwidth

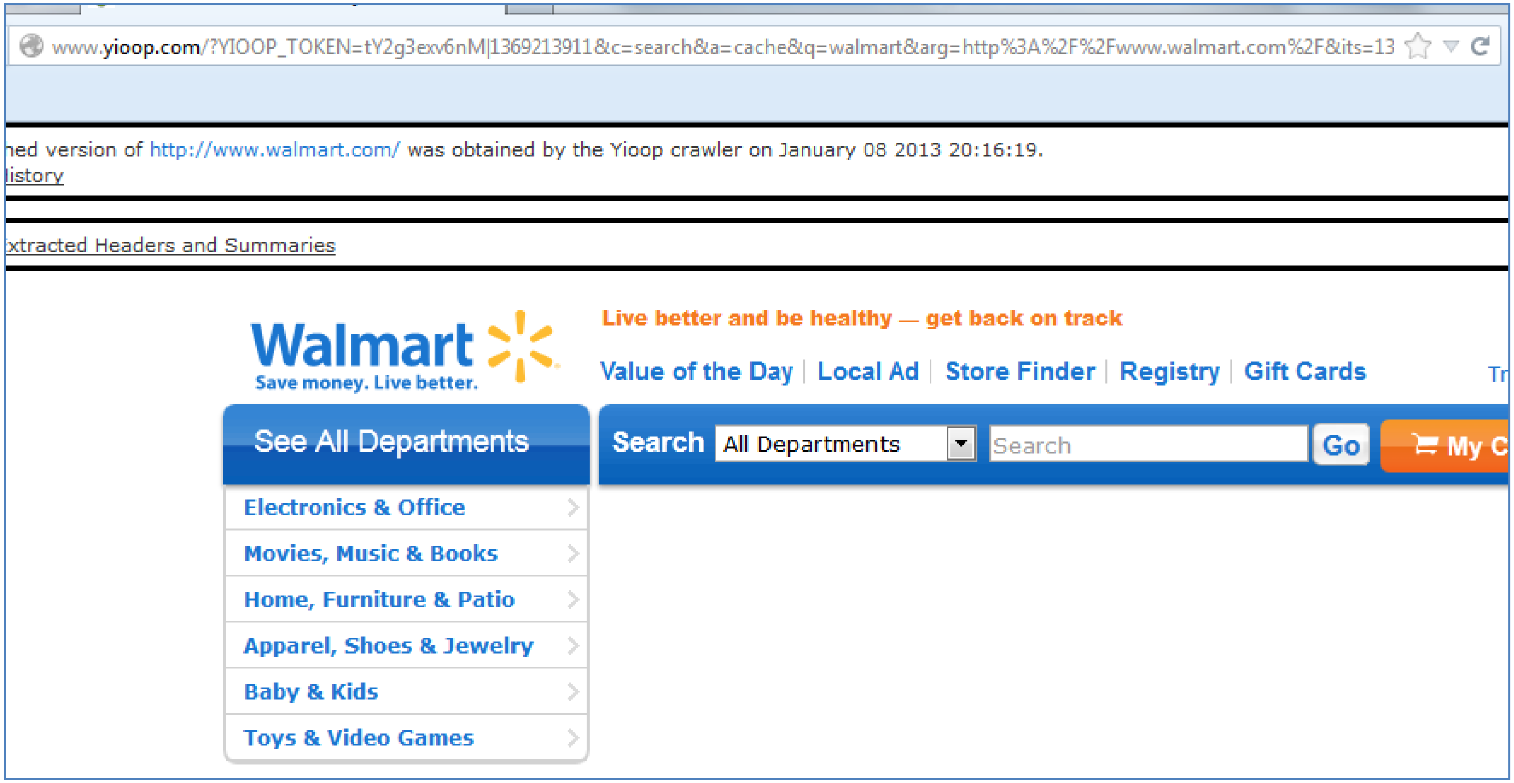
History Feature

- Search engines often maintain caches of web pages
- Link to cached version displayed along with search results
- Only latest version of cached page is accessible
- History feature displays links to all cached pages.

History Feature

- Step 1: Modified Yioop’s cache request and output code
  - If a cached page is not present for a given timestamp, find the nearest timestamp that has a cache.

History Feature

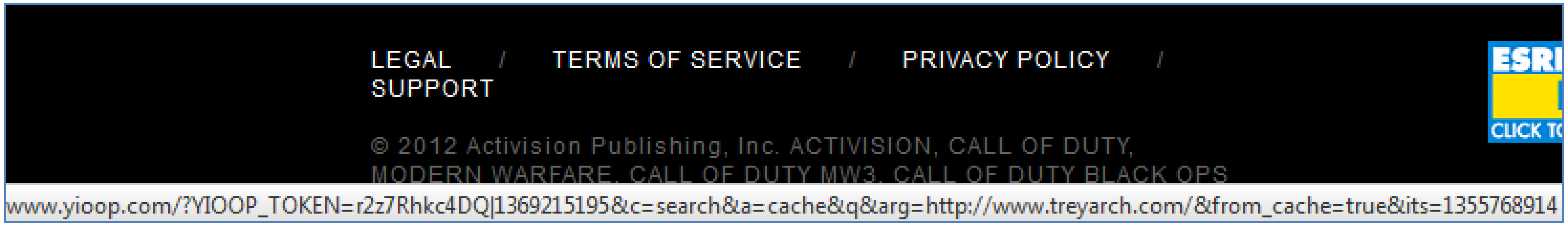


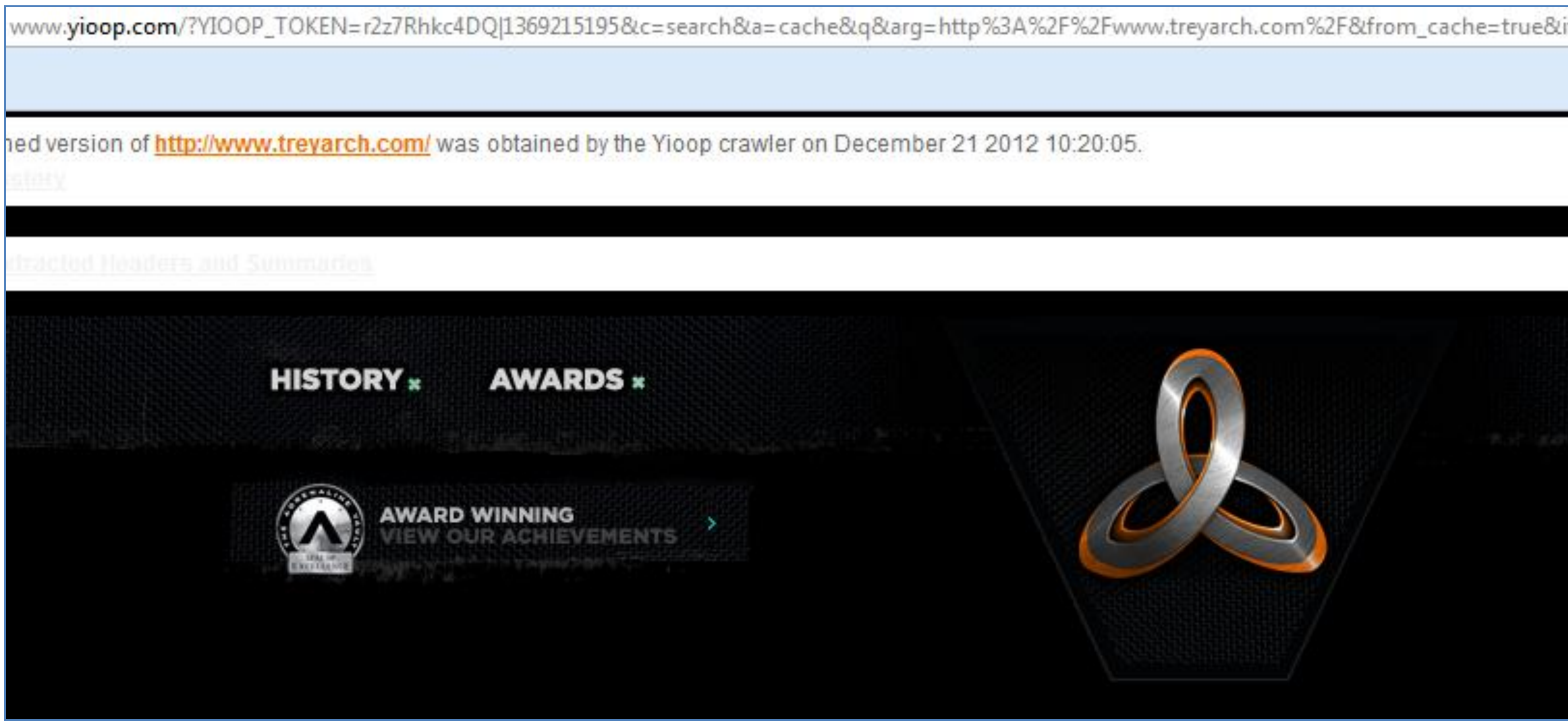
http://www.yioop.com/?YIOOP_TOKEN=tY2g3exv6nM|1369213911&c=search&a=cache&q=walmart&arg=http%3A%2F%2Fwww.walmart.com%2F&its=1355768914

History Feature

- Step 2: Modified links within cached web pages so that they follow Step 1
  - Modification done is similar to that done by the WayBack Machine.
  - WayBack Machine uses JavaScript.
  - History Feature modifies links during link canonicalization

History Feature

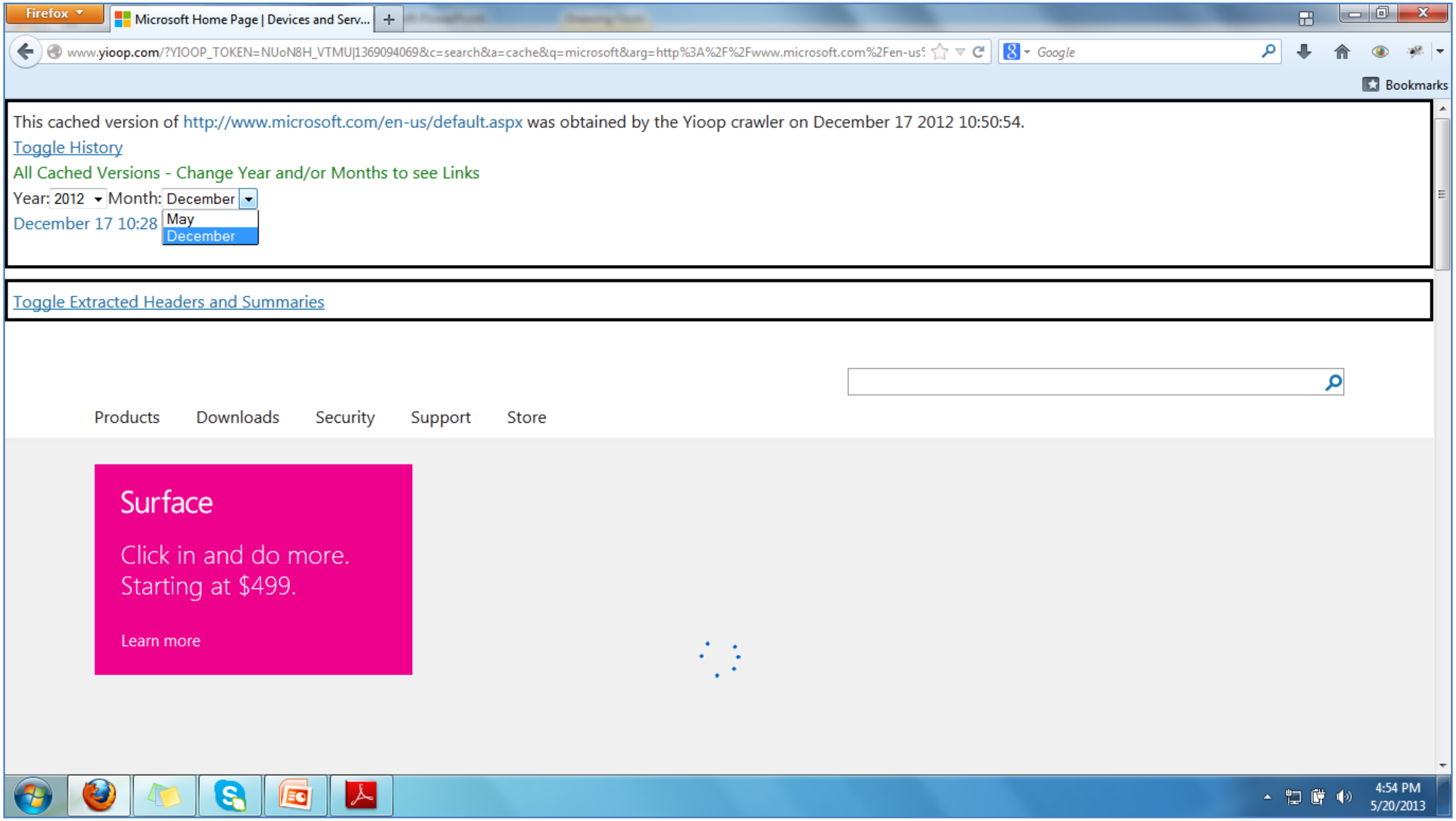




History Feature

- Step 3: Implemented History UI
  - Displays links to all versions of cached pages with Day and Time
  - User can change Year and Month to view links

History Feature



Cache Page Validation Feature

- ETags: Unique identifiers associated with web resources
  - Part of HTTP
  - When a web resource is modified, the ETag is changed

```
HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Mon, 24 Dec 2012 08:54:24 GMT
Accept-Ranges: bytes
ETag: "4455a945b4e1cd1:0"
Date: Wed, 22 May 2013 10:25:10 GMT Content-
Length: 2292
```

Cache Page Validation Feature

- ETag headers: Tacked on to HTTP header when making a request
  - If-Match: “etag”
    - If “etag” matches ETag of requested resource, entire resource is downloaded
    - Otherwise, Status **412 Precondition** failed is returned
  - If-None-Match: “etag”
    - If “etag” matches ETag of resource, resource has not been modified
    - Status **304 Not Modified** is returned by the server

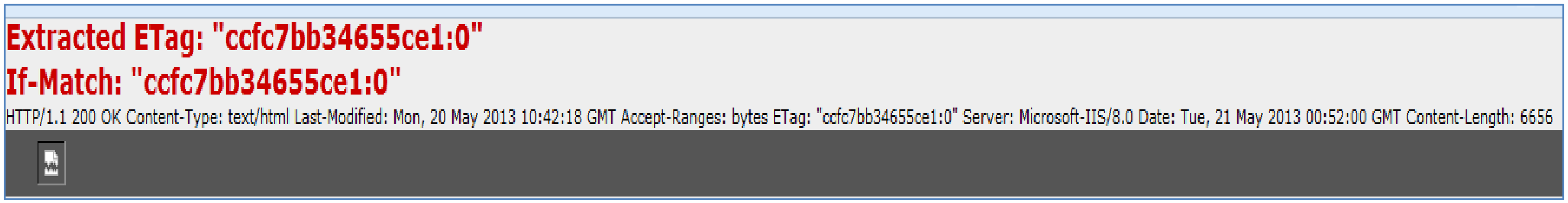
Cache Page Validation Feature

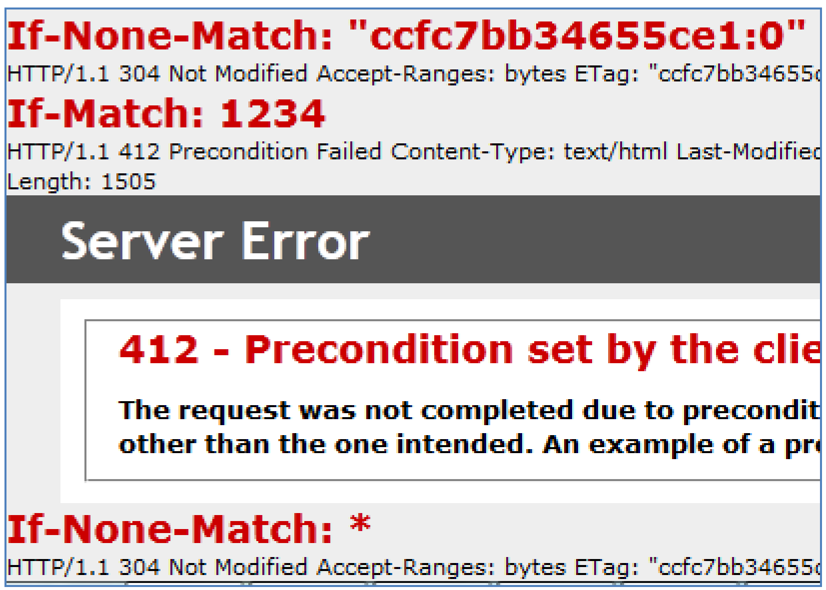
- Expires header: Tells when a web resource will expire.

```
HTTP/1.1 200 OK
Date: Wed, 22 May 2013 10:58:05
GMT Server: Apache
Accept-Ranges: none
Cache-Control: max-age=86400
Expires: Thu, 23 May 2013 10:58:05 GMT
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

Cache Page Validation Feature

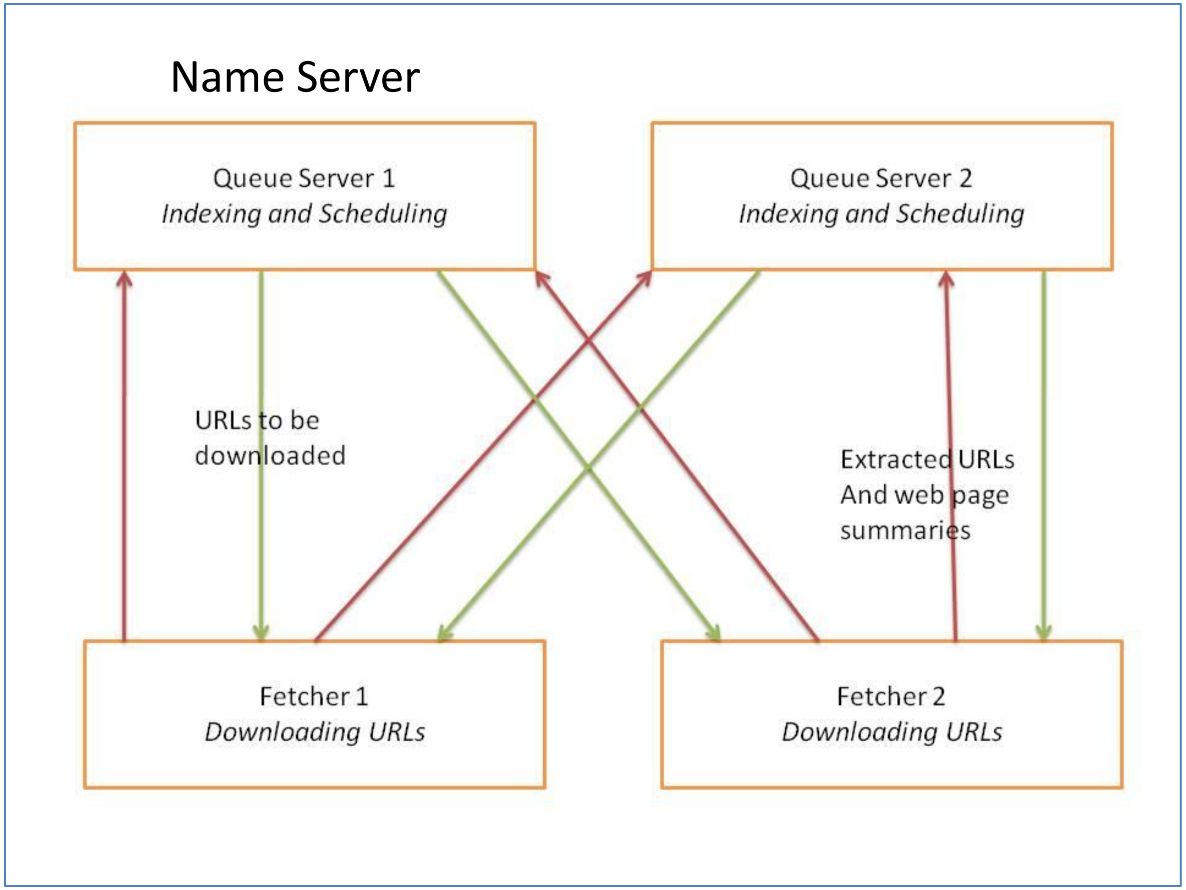
- ETag experiment with PHP, cURL, and ETag headers





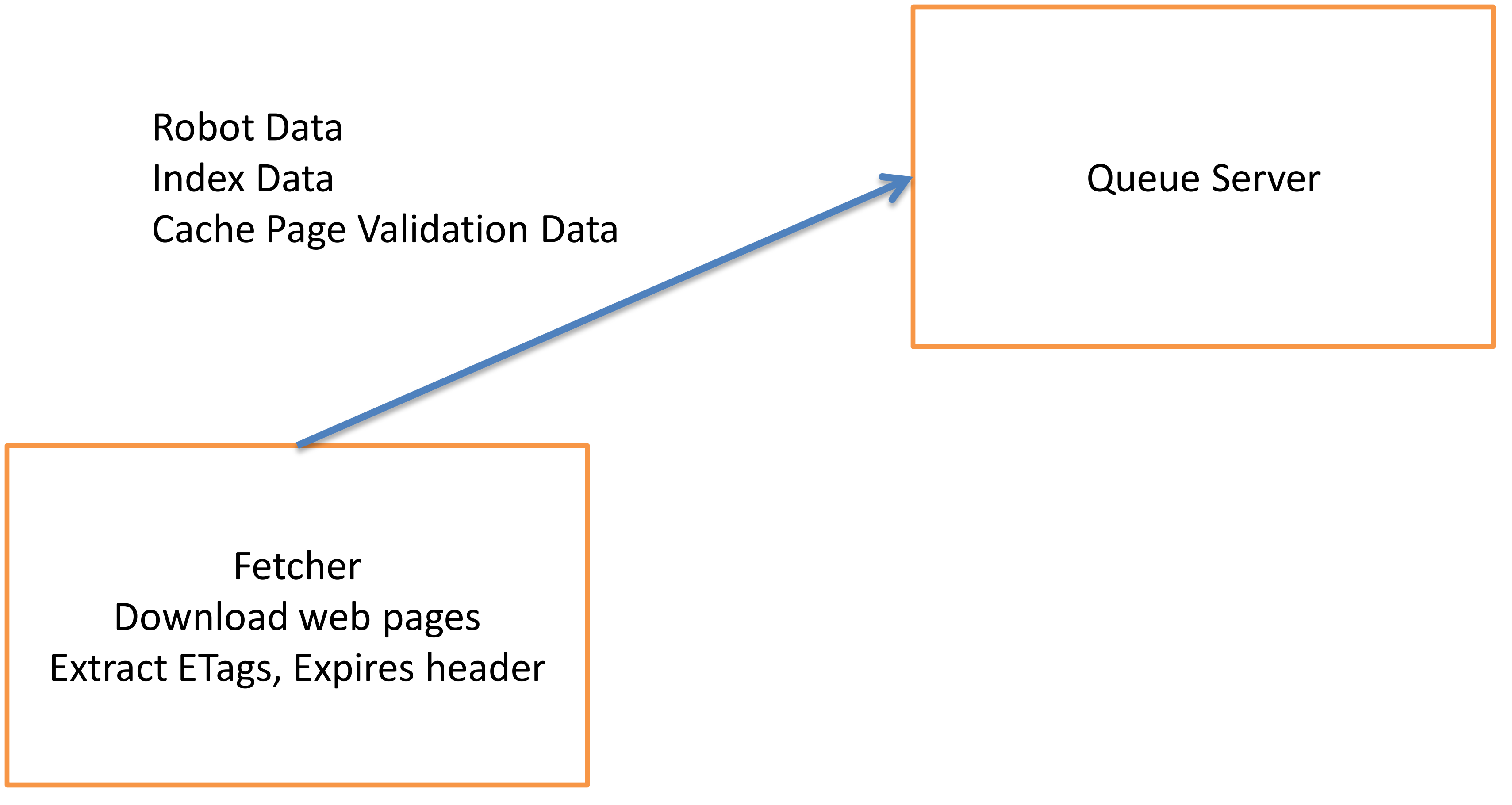
Cache Page Validation Feature

- Yioop's components



Cache Page Validation Feature

- Step 1: Modified Fetcher code

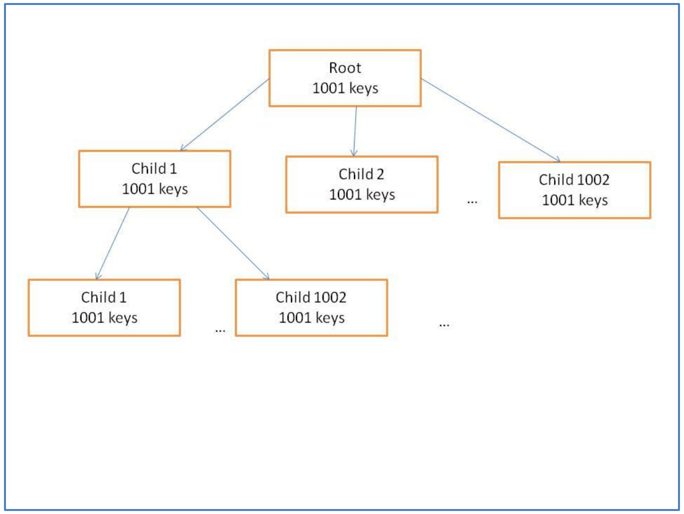


Cache Page Validation Feature

- Disk access experiment
  - Queue Server URL fetch batch size = 5000
  - Data structure for cache page validators should be fast to lookup 5000 entries among millions of URLs
  - Storage issues
    - No limit on ETag length
    - Limit on maximum file size depends on file system
  - Performed experiment with 5000 lookups on a 2GB file with 4 byte offsets.
    - Total time taken = 0.22 seconds (lower bound)

Cache Page Validation Feature

- Data structure for storing ETags and Expires
  - B-Tree
    - High branching factor reduces tree height
    - Reduced height means reduced number of disk lookups
    - Scalable with large number of keys



Cache Page Validation Feature

- B-Tree implementation for storing ETags and Expires
  - ETags and Expires headers stored as key-value pairs
  - Key = hash(URL) using Yioop's hash function
  - Value = ETag and Expires timestamp
  - Each node can have up to 1000 key-value pairs

Cache Page Validation Feature

Node Id

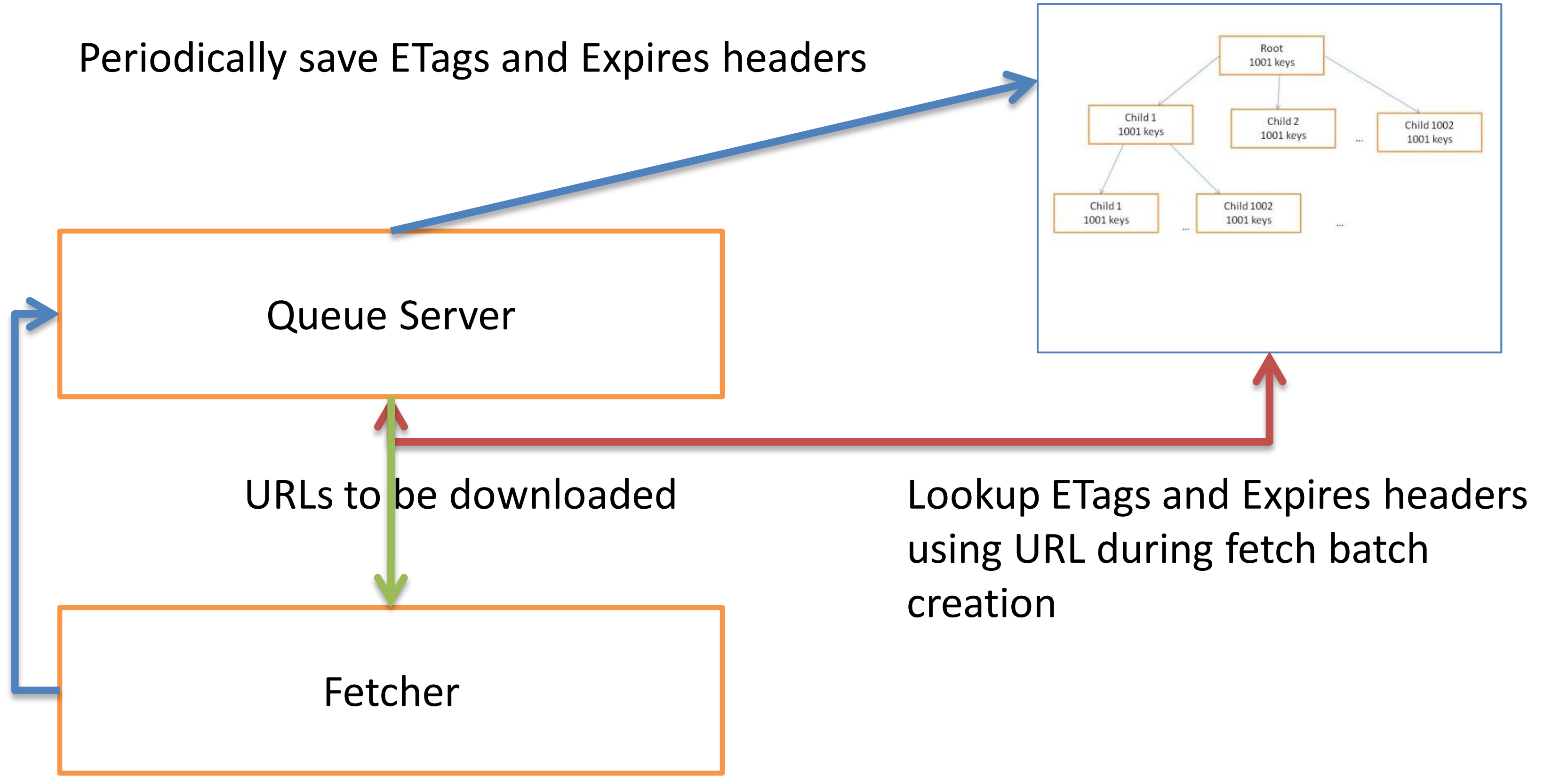
keys = array(array(key1, array(ETag, Expires)), array(key2, array(ETag, Expires)), …)

Links = array(child1_id, child2_id, …)

B-Tree node for cache page validation feature

Cache Page Validation Feature

- Step 2: Modified Queue Server code



Cache Page Validation Feature

- Queue Server pseudo-code

```
Lookup cachePageValidationData in Work Directory
if cachePageValidationData is found
    data = read(cachePageValidationData)
    array(url, array(etag, expires)) = data
    url_hash = hash(url)
    btree->insert(array(url_hash, array(etag, expires)))
```

Cache Page Validation Feature

- During Fetch batch creation

```
While creating fetch batch of URLs to be downloaded
    for each URL selected from priority queue
        url_hash = hash(url)
        array(url_hash, array(etag, expires)) = btree->lookup(url_hash)
        if etag found and expires found
            if current timestamp < expires
                continue
            else
                append etag to url
        else if only etag found
            append etag to url
        else if only expires found
            if current timestamp < expires
                continue
        add url to fetch batch
```

Cache Page Validation Feature

- Fetcher pseudo-code

```
Lookup scheduledata
if scheduledata is found
    urls = read(scheduledata)
    for each url in urls
        if url has etag appended to it
            url_header = concatenate("If-None-Match:",etag)
        downloaded_page = download(url, url_header)
        if downloaded_page has etag and expires
            add array(url, array(etag, expires)) to list of found sites
if sending Index and Robot Data
    Check if found sites have etag and expires
    CachePageValidationData = array(array(url1, array(etag1, expires1)),
        array(url2, array(etag2, expires2),...))
    send IndexData, RobotData, and CachePageValidationData to Queue Server
```

Cache Page Validation Feature

```
function download(url, header)
1.  Initialize cURL with url
2.  add headers to cURL HTTP header
3.  downloaded_page = execute cURL request
4.  return downloaded_page
```

Cache Page Validation Feature

- Experiment to see if the cache page validation feature is feasible
  - Performed web crawl
  - Number of pages crawled = 100, 000
  - Using Yioop's default set of seed sites
  - Page re-crawl frequency = 3 hours
  - Two Queue Servers and two Fetchers on a single machine
- One crawl each for Yioop without cache page validation, and Yioop with cache page validation

Cache Page Validation Feature

- Experiment 1: Comparison of average time taken by Queue Server to create fetch batch
  - Noted down the time taken by the cache page validation feature and compared with time taken by Yioop without cache page validation

Without cache page validation	With cache page validation
0.8 seconds	14 seconds

  - Conclusion: B-Tree lookup took 14 seconds on average
    - Serialization/de-serialization in PHP

Cache Page Validation Feature

- Experiment 2: Determining savings in bandwidth
  - Noted down URLs that weren't scheduled by Queue Server
  - Noted down URLs that returned Status 304 on being requested by Fetcher
  - Results:
  - Total URLs stored in B-Tree = 54,939

Cache Page Validation Feature

- Results contd…
  - Total URLs re-crawled with cache page validators = 412 (0.7% of total URLs in B-Tree)
  - Total savings in bandwidth = 15 MB
  - Savings due to Expires: 12 MB
  - Savings due to ETags: 3 MB
  - Savings in images
    - Savings for both Yioop and the source

Conclusion

- History feature enables users to view entire history of web pages cached by Yioop.
  - Enables full text search on all cached versions
- Cache page validation feature is promising
- Improvements and Future Work
  - Use methods other than serialization for storing nodes
  - Experiment with other disk-based data structures. For example B+ Trees.
  - Test on a larger crawl with multiple machines