Slide 0

Improving Web Performance Through Server Volumes and Proxy Filters

Jennifer Rexford

`http://www.research.att.com/~jrex`

Work with Edith Cohen and Balachander Krishnamurthy.

Slide 1

Outline

- Web performance problems, and solutions
- Hint exchange between proxies and servers
- Efficient computation of server hints
- Evaluation using proxy and server logs
- Practical issues (protocol and overheads)
- Conclusions and ongoing work

Slide 2

Web Performance Problems

Opening a TCP connection to the server

- One-way delay for TCP-open request to reach server
- Server delay to open the TCP connection
- One-way delay for acknowledgment to reach client

Requesting resource from the server

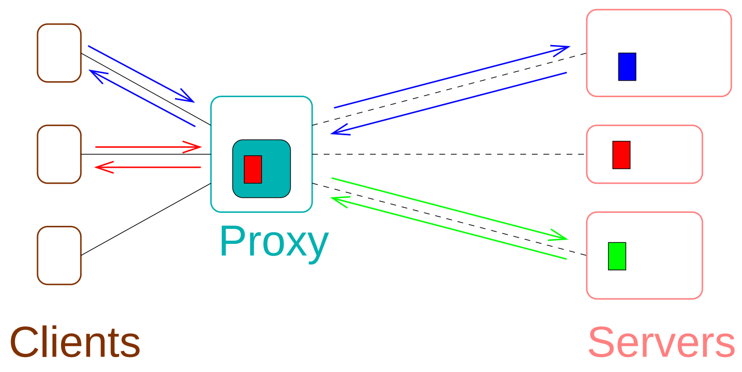
- One-way delay for HTTP request to reach server
- Server delay to generate HTTP response

Transmitting response to client

- One-way delay for the first byte to reach client
- Delay-cum-throughput to receive remaining bytes

Slide 3

Web Clients, Proxies, and Servers



Slide 4

Cache Replacement

Problem formulation

- Decide which resource to evict when cache is full
- Increase hit rate (object hit rate and byte hit rate)
- Reduce user latency and network bandwidth

Proposed policies

- Replace least recently/frequently used resource
- Weight decision by resource size or fetching cost
- Weight decision by staleness of the cached resources

Slide 5

Cache Coherency

Problem formulation

- Validate resource with GET If-Modified-Since request
- Receive Not-Modified response or the newest version
- Reduce latency and bandwidth, and control staleness

Proposed policies

- Static time-to-live to evict or refresh resources
- Adaptive time-to-live based on age of the resource
- Piggybacking of cache validation on regular messages

Slide 6

Other Performance Improvements

Prefetching

- Fetch resources in advance of client request
- Identify resources likely to be requested in future

⇒ reduces user latency, but consumes extra bandwidth

Persistent connections and pipelining

- Maintain a TCP connection to the server
- Fetch successive resources on same TCP connection
- Pipeline requests to avoid waiting for response

⇒ reduces user latency, but increase # active connections

Slide 7

Observations

Better performance with an end-to-end solution

- Proxy decisions have a big impact on performance
- Good decisions require estimates of future accesses
- Server has extensive info about access patterns

⇒ proxy-server exchange of hints about future accesses

Heterogeneity of access characteristics

- Small number of servers attract most of traffic
- Small number of resources attract most of the accesses

⇒ don't need to change the entire Web to see benefits

Slide 8

Practical Limitations

Pragmatic constraints

- No changes to HTTP 1.1
- No forced compliance at all proxies/servers
- Avoid per-proxy/per-server state
- Avoid generating false hints

Performance constraints

- No new TCP connections
- Avoid wasting bandwidth
- Avoid increasing user-perceived latency
- Avoid increasing load on the server and proxy

Slide 9

Subproblems

- What hints should server send
- How to exchange information (piggybacking)
- How to precompute server hints (volumes)
- How should the proxy limit hint size (filters)

Slide 10

What Hints to Send?

- Resources likely to be accessed in the near future
- Likelihood of access to resource
- Last-Modified time of resource
- Resource size
- Resource content type
- ...

Slide 11

How to Exchange Hints?

- Proxy receives a client request for resource r
  - Checks if the cache has fresh copy of r
  - On cache miss, sends a request for r to server and piggybacks the filter
- Server receives a client/proxy request for r
  - Applies the proxy filter to volume v(r)
  - Piggybacks filtered volume on response
- Proxy replies to client and processes hints

Slide 12

How to Filter Hints?

Tailor hints to requesting proxy

- Types and sizes of resources in cache
- Cache policies (e.g., no images; no large resources)
- User needs (e.g., no staleness; no large resources)

Limit number of hints

- Specify maximum number of piggybacked hints
- Disable hints from recently piggybacked volumes

Slide 13

Volume Construction

Problem formulation

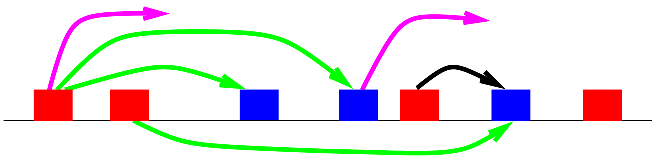
- **Input**: triple (srcId, resource, ReqTime)
- **Output**: map r to a set of resources $v(r)$
- **Evaluate**: predict $v(r)$ after request for r

Performance metrics

- **Fraction predicted**: fraction of requests predicted by in the last T seconds
- **True prediction**: fraction of "distinct" predictions that get fulfilled in next T sec
- **Hint size**: Average # resources in filtered volume

Slide 14

Predictions



- **Fraction predicted**: $3/(3+4) = 3/7$
- **True prediction**: $3/(3+2) = 3/5$
- **Hint size**: $(3+3)/(3+4) = 6/7$

Slide 15

Volume Construction Techniques

- Optimal volume-construction
  - Optimize one metric subject to bound on others
  - Problem is NP complete, so need heuristics
- Directory structure in URL
  - All resources in same k-level subdirectory
  - E.g., `cnn.com/a/b.html` and `cnn.com/a/c.html`
- Pairwise probabilities with thresholding
  - Resources s accessed after r with high probability
  - E.g., probability threshold of 0.7 or 0.9

Slide 16

Probability-Based Volumes

Compute pairwise probabilities from counters

- $c(r)$: number of accesses to each resource r
- $c(s|r)$: accesses to r followed by s within T seconds
- $p(s|r)$: access probability $c(s|r)/c(r)$

$\Rightarrow$ include s in $v(r)$ if probability $p(s|r)$ exceeds threshold

Example computation of $p(b|a)$

- Accesses $a, b, c, \ldots, b, d, \ldots a, d, \ldots a, e, b, \ldots a, f$
- Number of accesses: $c(a) = 4$ and $c(b) = 3$
- Counter $c(b|a) = 2$ and probability $p(b|a) = 2/4$

Slide 17

Number of Counters

Scalability challenges in maintaining counters

- Servers often host a large number of resources
- Volume construction requires n^2 pairwise counters
- Storing/updating counters introduces overhead

Reducing the number of counters

- Omit resources that do not occur very often
- Resources in the same (sub)directory or chain of links
- Random sampling of resources that occur often

$\Rightarrow$ significant reduction in the number of counters

Slide 18

Ineffective Predictions

Limitation of probability-based volumes

- Reduce complexity by considering pairs independently
- Ignores predictions generated by other volumes
- Redundant and false predictions, and larger hints

Example of ineffective prediction $p(s|r)$

- Accesses $w, x, r, s, \ldots, w, x, s, \ldots, r, t, u, \ldots, w, r, s$
- r never generates a new, true prediction for s
- r sometimes generates a false prediction for s

$\Rightarrow$ $v(w)$ should include s, and $v(r)$ should not

Slide 19

Reducing False Predictions

Thin volumes based on effectiveness metric

- Construct volumes using pairwise probabilities
- Measure how often predictions are both new and true
- Apply a threshold to detect poor predictors
- Remove ineffective elements from the volumes

Potential performance benefits

- Reduce hint size
- Increase true predictions
- Limited impact on prediction fraction
- Just one extra pass through the log

Slide 20

Proxy and Server Logs

Proxies generate requests to diverse servers:

- Large AT&T and DEC proxy logs
- Can evaluate directory-based volumes
- Cannot evaluate probability-based volumes or measure volume size

Servers receive requests from diverse clients:

- Sun, Apache, AIUSA, and Marimba server logs
- Can evaluate both volume-construction schemes
- Cannot determine accesses handled directly by proxies

Slide 21

Web Server Logs

Server Log (days)	Number of Requests	Number of Clients	Unique Resources
Sun (9)	13,037,895	218,518	29436
Apache (49)	2,916,549	271,687	788
Marimba (21)	222,393	24,103	94
AIUSA (28)	180,324	7,627	1102
EW3_4 (94)	6,273,792	79,623	2638
EW3_6 (94)	6,281,303	102,454	784
EW3_18 (94)	3,902,302	58,839	279
EW3_60 (94)	45,903,646	481,628	2,031
Nagano (1)	11,665,713	61,707	33,875

Slide 22

Directory-Based Volumes

Basic performance properties

- Predict 60–80% of accesses
- Average piggyback size of 50–100 elements
- Low true-prediction fraction

Disabling piggybacking of recent volumes

- Retains the $60 - 80\%$ prediction
- Lowers average piggyback size to 20
- Little change in true-prediction fraction

Slide 23

Probability-Based Volumes

Basic performance properties

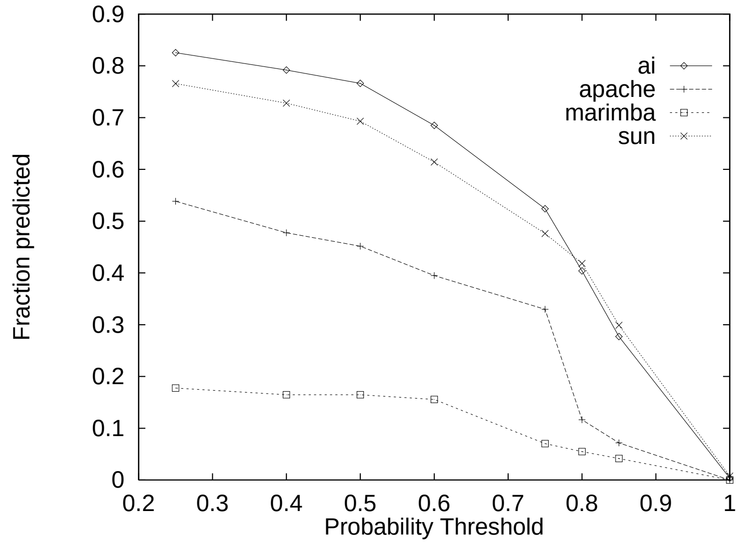
- Predict 60–80% of accesses
- Average piggyback size of 5–15
- Higher true-prediction fraction

Removing ineffective hints

- Still predict 60-80% of accesses
- Average piggyback size of 2–10
- Even higher true-prediction fraction

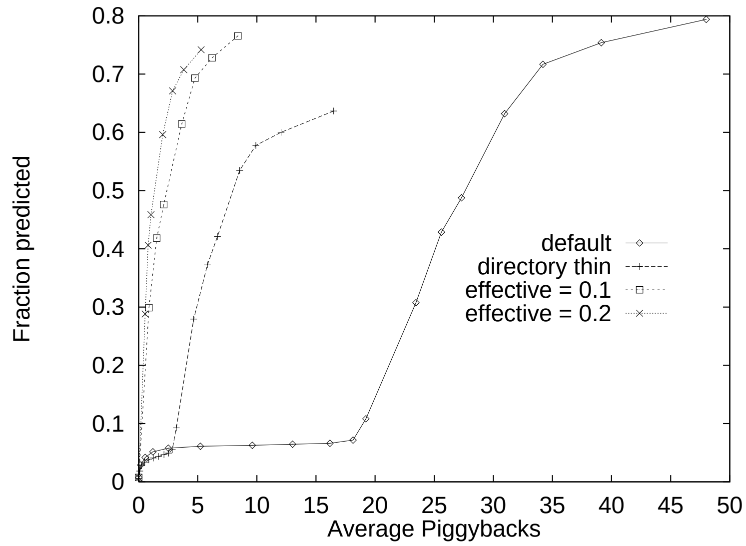
Slide 24

Predictions vs. Probability Threshold



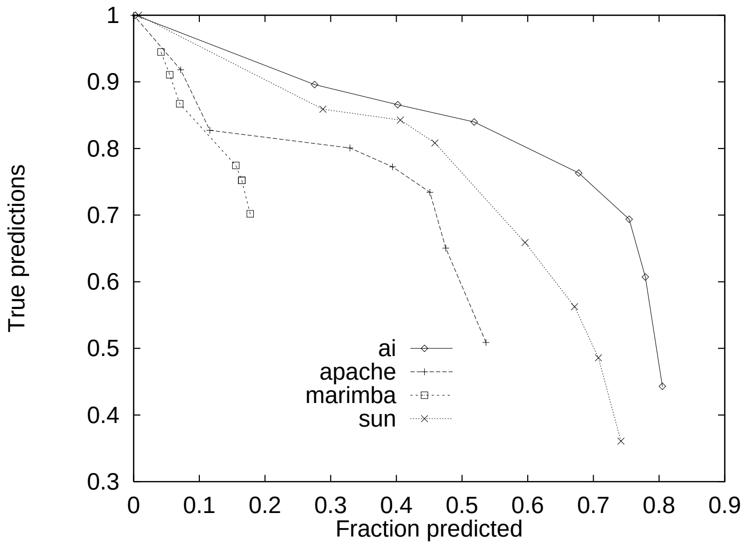
Slide 25

Predictions vs. Piggyback Size (Sun)



Slide 26

Predictions vs. True Predictions



Slide 27

Application of Results

Cache coherency

- Reduce GET-IMS requests, update changed resources
- High prediction fraction to in/validate most resources

Cache replacement

- Avoid replacing unchanged resources, and delete changed resources
- Relatively high true predictions for good decisions

Prefetching

- Prefetch resources in the piggyback list
- High true predictions to avoid wasted prefetching

Slide 28

Pragmatics: Transmitting Volumes

Add to response header

- Pro: works in both HTTP 1.0 and HTTP 1.1
- Con: response body delayed

Use chunked encoding and add to trailer

- Pro: trailer avoids delaying response body
- Con: works only in HTTP 1.1

... proposed mandatory HTTP extension to send pointer to volumes

Slide 29

Pragmatics: Piggyback Messages

- Volume identifier (2 bytes per message)
- URL (50), Last-Modified Time (8), Size (8) bytes per piggyback element
- Small number of piggybacks (e.g., ten, at 662 bytes)
- Adds at most one extra packet (5% increase in bytes)
- Often avoids future server requests and reduce latency

Slide 30

Example

Request:

```
GET /mafia.html HTTP/1.1
host: sig.com
TE: chunked
Piggy-filter: maxpiggy=10; rpv="3,4";
```

Response:

```
HTTP/1.1 200 OK
Trailer: P-vol
Transfer-Encoding: chunked
 < Size-of-chunk >
      <data>
   ...
 0
P-vol: vol=7; pe="u4,895527629,5465"; pe="u3,891527021,1290";
CRLF
```

Slide 31

Conclusion

- Efficient proxy-server information exchange
- Volume construction and filtering techniques
- Experimentation on proxy and server logs
- Piggybacking mechanisms within HTTP 1.1

Slide 32

Ongoing Work

- Efficient algorithms for volume construction
- Using hints in cache replacement policies
- Using hints in persistent TCP connection
- Transparent volume center to offload server
- Proxy filter specification language
- Efficient data structures for applying filters