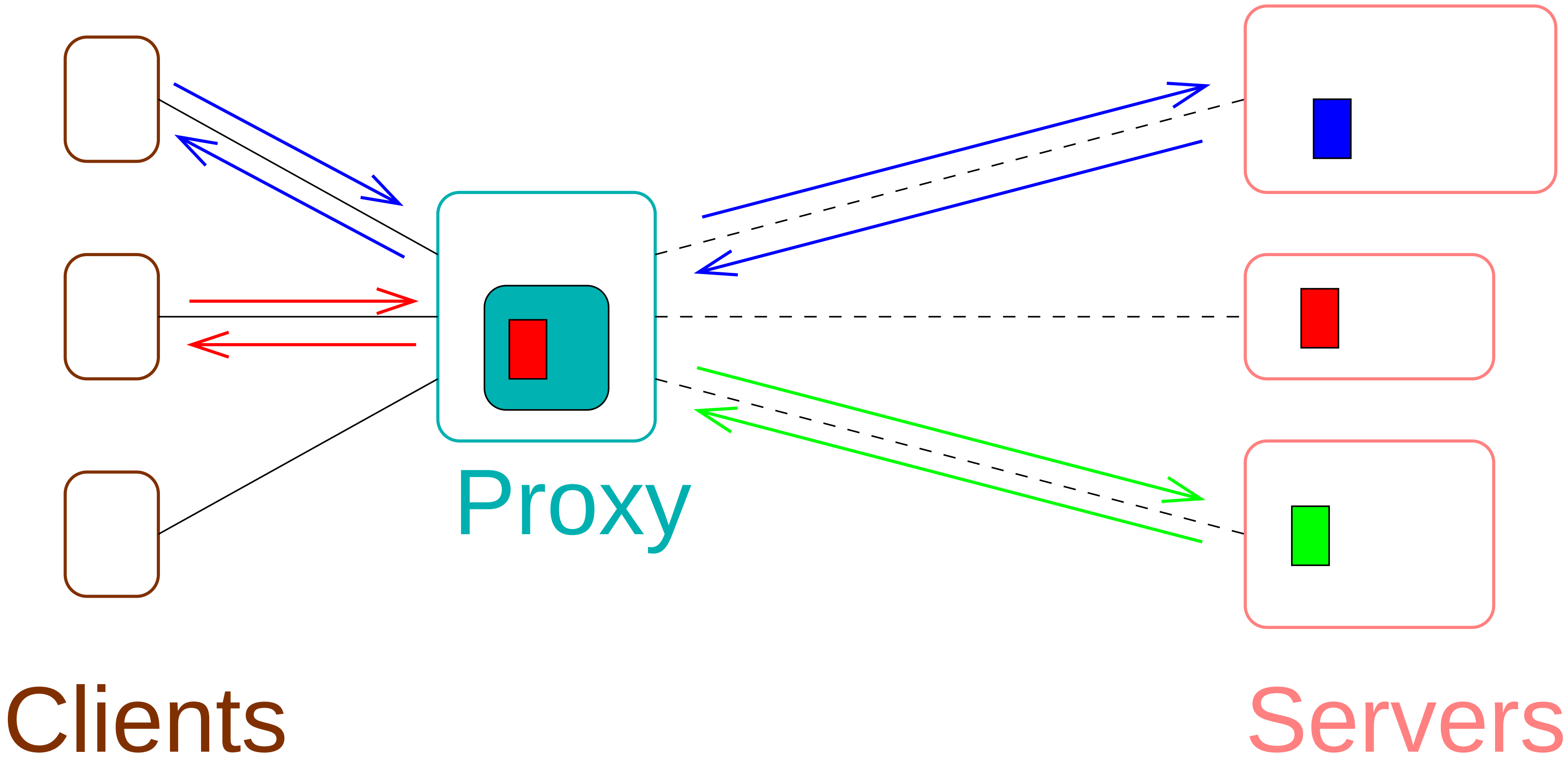
Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters

Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford
AT&T Labs–Research
180 Park Avenue, Florham Park, NJ 07932 USA
{edith,bala,jrex}@research.att.com

World Wide Web

- Very popular, rapid climb in traffic
- Few servers attract most of traffic
- Few resources on a site attract most of the accesses
- User-perceived latency, bandwidth, and server load increasing

Current model of the Web



Improving web performance

- Caching (Williams+96, Cao/Irani97)
- Cache coherency (Dingle/Partl 96, Krishnamurthy/Wills 97 and 98)
- Prefetching (Bestavros95, Padmanabhan/Mogul96, Kroeger+97)
- Persistent connection/Pipelining (Padmanabhan/Mogul94, HTTP 1.1)
- Delta/Compression (SIGCOMM97)

Performance can be further improved by having proxies and servers exchange hints about future accesses

Subproblems to solve

- What hints should server send

  Resources likely to be accessed in the future
  Last-Modified time, size, type and other attributes

- How server should precompute hints (*volumes*)

  Groups of resources in the same (sub)directory
  Resource pairs that are typically accessed together

- What information should proxy send (*filters*)

  Signature of cache (resource types, sizes, resources)
  Limiting number of hints (maximum size, recent volumes)

- How to exchange information: piggyback on request/response

Exchanging 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 proxy request for resource r

- Applies the proxy filter to the precomputed volume v(r)
- Sends a response for resource r and piggybacks filtered volume

Proxy returns the resource to the client and processes the hints

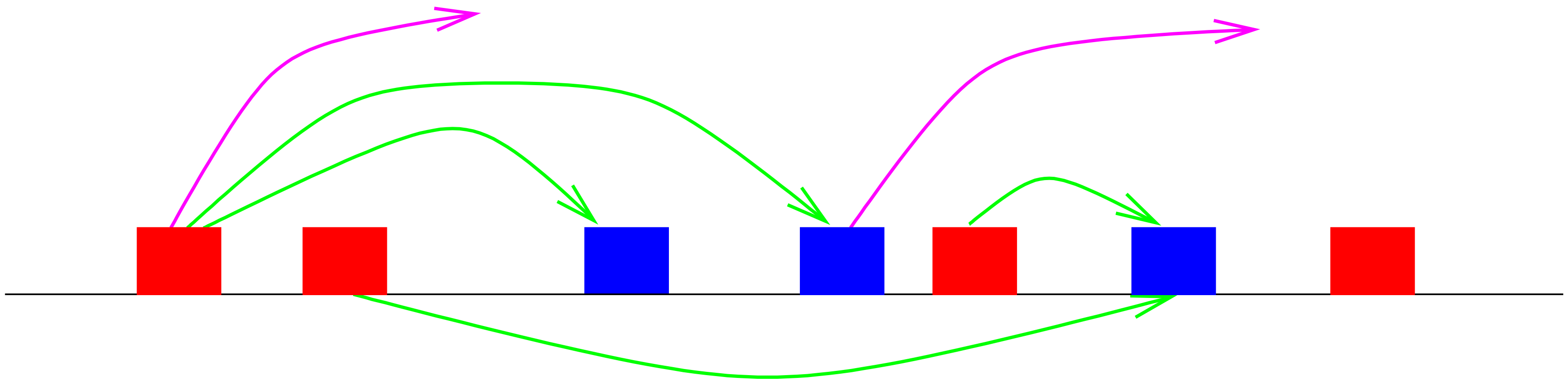
Volume construction techniques

- Volume-set: mapping of each resource r to a set of related resources
- Extract (clientIP, resource, RequestTime) triple from server logs
- Volume items treated as predictions upon access to r

Two techniques:

- Directory structure based (`foo/a, foo/b, foo/bar/a`)
- Probability based (look at pairwise probability of a resource following another)

Performance metrics



Fraction predicted: fraction of requests predicted by recently received volumes in last T seconds

True prediction: fraction of distinct predictions that get fulfilled in next T seconds

Bandwidth cost: Average size of filtered volume

Proxy and server logs

Proxy logs of requests to a wide range of servers:

- Can evaluate performance of directory-based volumes
- Cannot evaluate probability-based volumes or measure volume size
- Experiments with large AT&T and DEC proxy logs

Server logs of accesses from a wide range of proxies:

- Can evaluate performance of both volume-construction schemes
- Cannot determine accesses that are handled directly by proxies
- Experiments with Sun, Apache, AIUSA, and Marimba server logs

Results of volume construction experiments

Directory-based volumes

- Can predict 60-80% of accesses with volume size of 50-100 hints with low true prediction
- Disabling use of recently sent volumes retains fraction predicted, and true prediction while lowering volume size to 20 hints

Probability-based volumes

- Can predict 60-80% of accesses with volume size of 2-10 hints and higher true prediction
- Removing hints that do not typically generate new predictions retains fraction predicted, further lowers volume size, and increases true prediction

Application of results

Cache coherency

- Reduce GET If-Modified-Since requests, update resources (via deltas)
- Need high prediction fraction to in/validate most resources

Cache replacement

- Avoid replacing unchanged resources, and delete changed resources
- Need relatively high true predictions for good replacement decisions

Prefetching

- Prefetch resources in the piggyback list
- Need very high true predictions to avoid wasted prefetching

Practical limitations

Pragmatic constraints

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

Performance constraints

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

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 allows sending pointer to volumes

Pragmatics: size of piggyback message

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

Example

Request:

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

Response:

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

Conclusion

- Proxy-server information exchange framework
- Volume construction and filtering techniques
- Performance evaluation on proxy and server logs
- Mechanisms for piggybacking within HTTP 1.1

Ongoing work

- Efficient algorithms for constructing server volumes
- Proxy cache replacement policies that use server hints
- Persistent TCP connection policies that use server hints
- Volume center to construct volumes and generate piggyback messages
- Proxy filter specification language and efficient data structures for specifying and applying filters

Thanks to all proxy/server log suppliers!