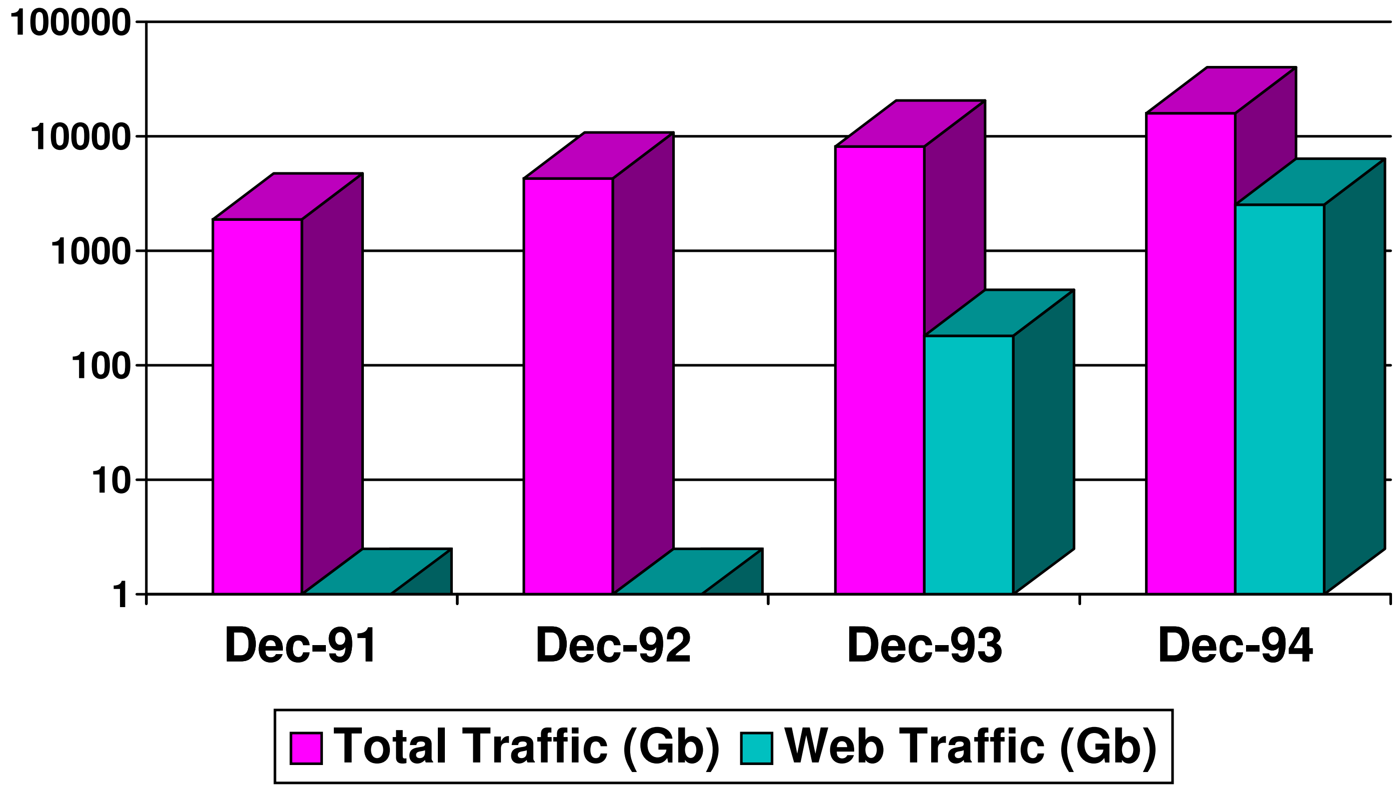
Autonomous Replication in Wide-Area Internetworks

James Gwertzman

Harvard University
gwertzma@das.harvard.edu
http://das-www.harvard.edu/research/vino/

Internet on verge of collapse?



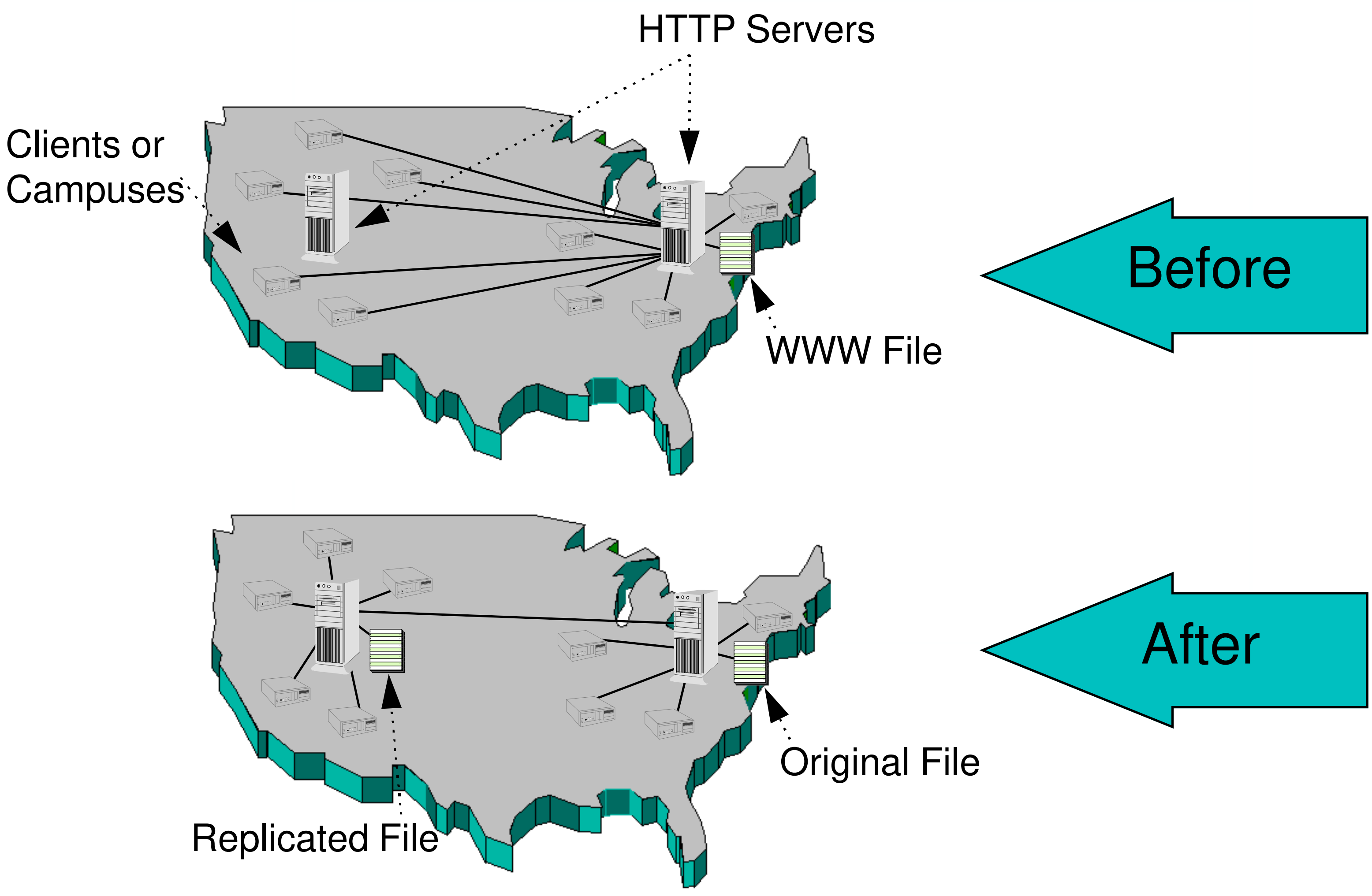
Solution: Cache Data

- Client-Caching
  - Cache space limited
- Proxy-Caching
  - Many clients share one cache space
  - 3 Gb cache = 65% request hit rate
  - Inefficient use of global disk-space
  - Special case of hierarchical-caching
- Hierarchical-Caching
  - Requests are serviced through hierarchy
  - Hard to deploy

Push-Caching

- Server decides where to cache objects
  - Can use network topology
  - Very efficient
  - Very scaleable / easy to deploy
- Client uses nearest replicated object
  - Added overhead offset by server load savings
  - Primary host redirects to nearest replicated data.
  - Inefficient: more efficient resource location schemes will only improve performance.

Push-Caching Illustrated



Presentation Roadmap

- The Case for push-caching
- Experimentation
  - Push-cache simulations
- Results
- Conclusion
  - Future work
- Cache Consistency

Web Traces

Server	*Traffic (Mb) / day*	*Requests / day*	*% Cacheable*
NCSA	6,492	605,454	66%
FAS	9.3	2,760	73%
HCS	8.1	1,750	72%
DAS	15	1,390	73%

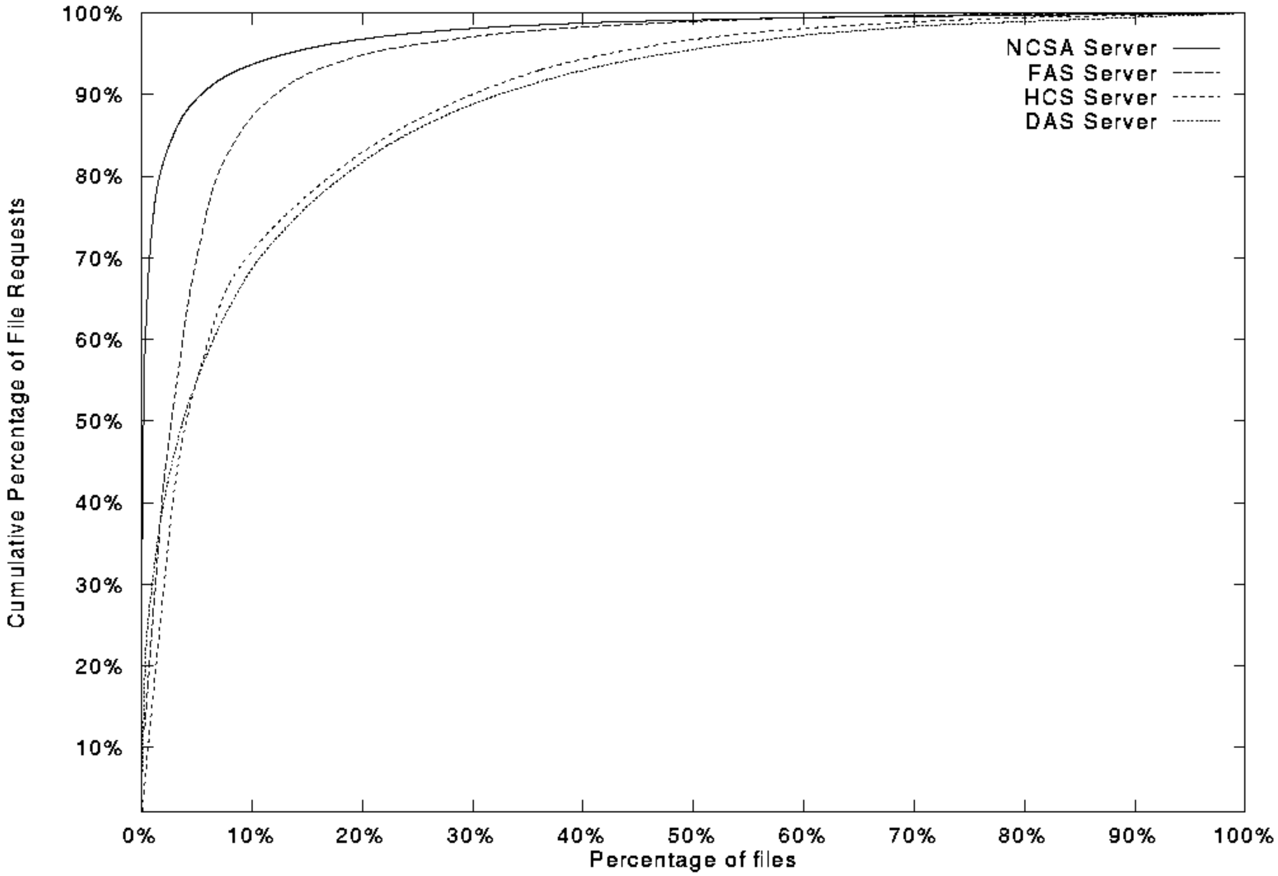
NCSA: Globally popular

FAS: Campus Information Server

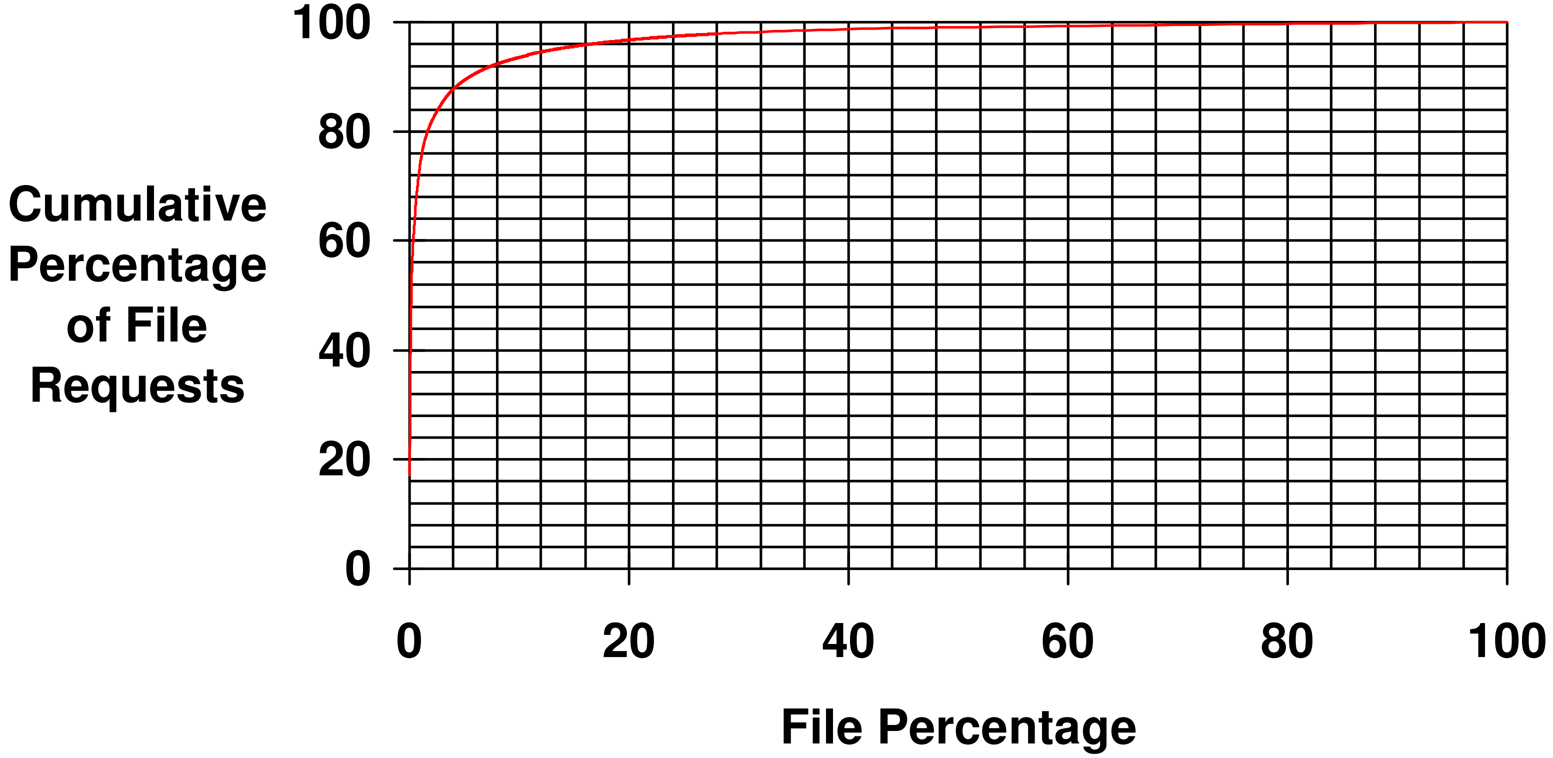
HCS: Harvard Computer Society

DAS: CS Department Server

File Popularity



NCSA File Popularity



Network Topology

- Network topology yields optimal caching
  - Not currently available for Internet
  - Approximate with geographical information?
- Moderate correlation between miles, network hops, and latency ($r^2 \sim .40$)
  - `traceroute` provides hops and latency
  - `nic.merit.edu` provides miles
- Higher correlation across same backbone ($r^2 \sim .60$)
  - Backbone-backbone connections often congested

Push-Cache Simulator

- Evaluate push-caching with simulator
  - » Trace-driven
  - » Simulates Internet
    - – Clients request files from servers
    - – Uses sample topology to measure traffic (from Guyton and Schwartz)
  - » Records:
    - – Server load (bytes/hour, requests/hour)
    - – Bandwidth (bytes x hops)
    - – Disk consumption

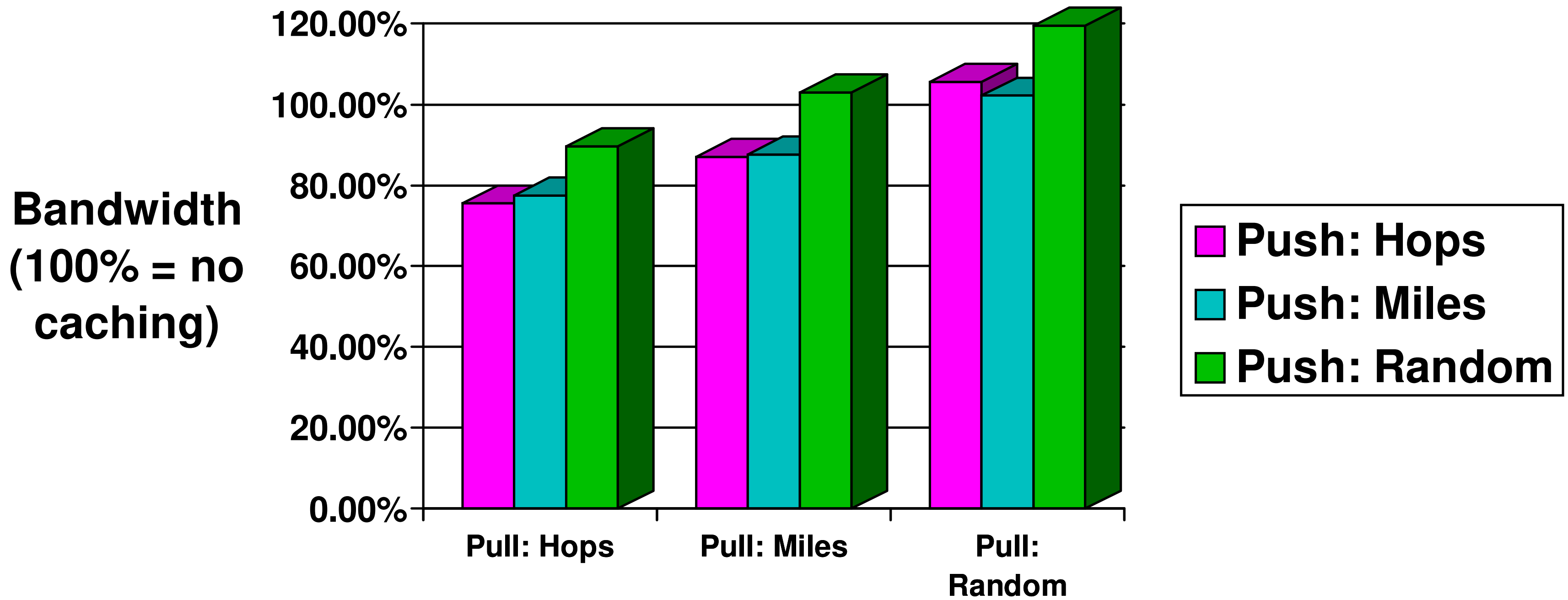
Parameters

Parameter	Description	Settings
Caching-Strategy	Where to push?	{random, miles, hops}
Lookup-Strategy	Where to pull?	{random, miles, hops}
Push-Threshold	When to push?	{0, ..., 100,000} req/hour
Pages-to-Push	How many to push?	{0, ..., 100} pages
Server-Number	How many servers?	{0, ..., 100} servers

Caching-Strategy used *friends-of-friends* algorithm

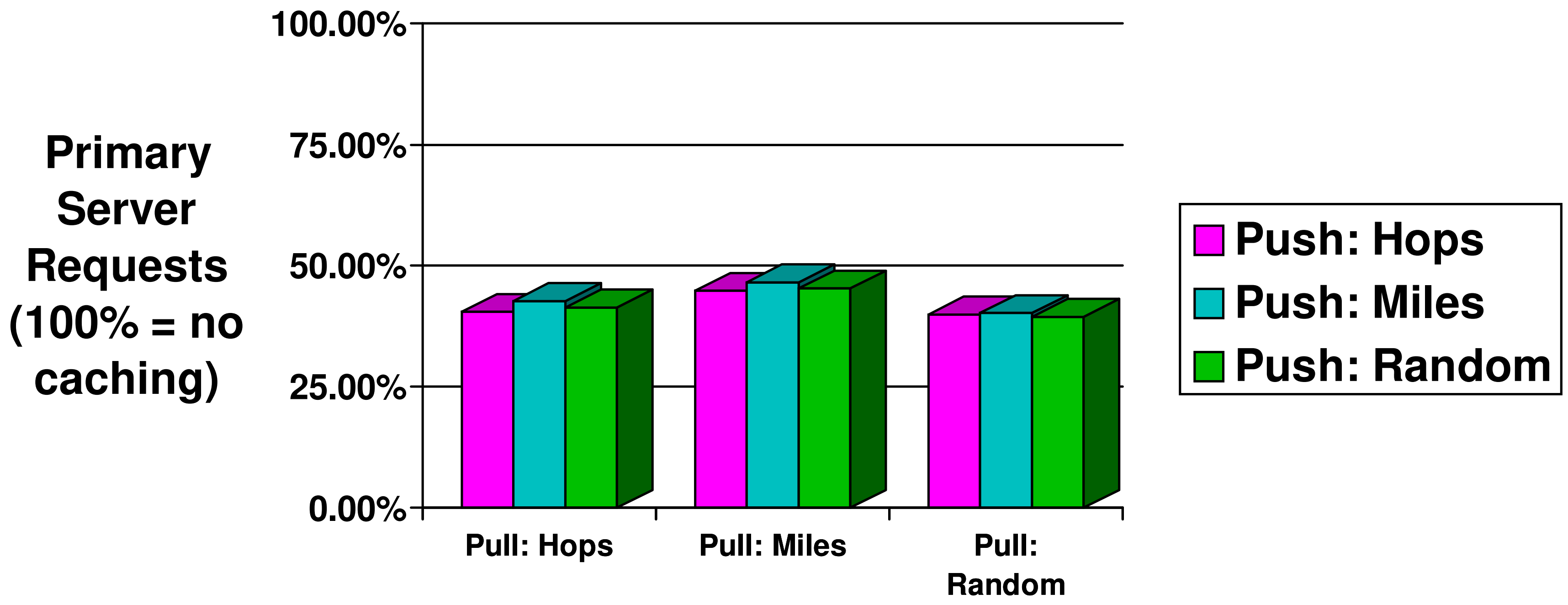
Random: Choose randomly
Miles: Use geography to calculate closest server
Hops: Use topology to calculate closest server

Strategy and Bandwidth



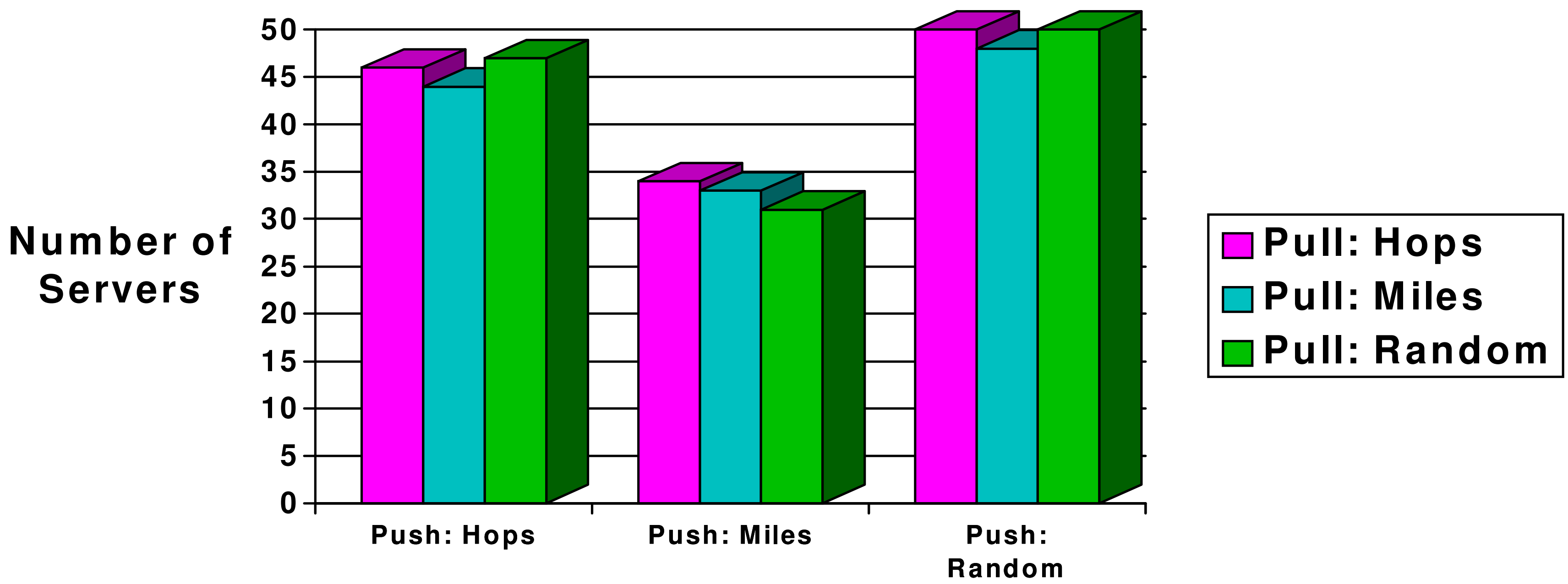
- Pulling strategy more important than pushing strategy
- Random uses more bandwidth than no caching

Strategy and Server Traffic



•Random is the most efficient at distributing load, but all perform comparably

Strategy and Number of Push-Cache Servers



- Random pulling/pushing can easily saturate servers
- Miles has limited resolution so it doesn't distinguish between servers as well

Summary of Other Results

- Threshold affects number of active servers
  - » Global threshold needed to regulate push-caching
- Servers available are not all used
  - » greater variety = closer to optimal
  - » Too many = takes too long to compute
- Amount to push depends on server
  - » Pushing more: reduces server load, increases overhead, decreases bandwidth costs, increases cache space consumed

Client Caching Vs. Push Caching

Caching Method	*Net Traffic*	*Primary Server Traffic*	*Average Server Traffic*	*Active Servers*
None	100%	100%	100%	1
Push-Caching (Miles)	85.6%	46.8%	2.8%	35
Push-Caching (Random)	76.4%	40.9%	1.9%	51
Push-Caching (Hops)	71.8%	39.6%	2.1%	47
Client-Caching	71.7%	71.6%	71.6%	1
Client & Push (miles)	65.4%	40.2%	2.1%	34
Client & Push (hops)	56.7%	33.7%	1.5%	47

Proxy-Caching Vs. Push-Caching

Caching Method	*Net Traffic*	*Primary Server Traffic*	*Total Cache Space (Mb)*	*Efficiency (savings/cache)*
None	100%	100%	0	-
Push-Caching (Miles)	86%	51%	8	1268
Push-Caching (Hops)	74%	41%	27	746
Client-Caching	71%	71%	n/a	n/a
Proxy-Caching	37%	36%	2219	21
Proxy & Push (miles)	36%	31%	2235	21
Proxy & Push (hops)	35%	29%	2256	21

Conclusion

- Push-cache compliments client, proxy cache
  - » Ideal for distributing server-only files
    - – audio streams, scripts for dynamic web files, data bases
  - » Autonomous mirroring of popular FTP files
- Especially well suited for video-on-demand
  - » Dynamically replicating very large files across a well known network topology
- Response time faster than proxy caching

Future Work

- Still need to explore:
  - server conflicts, server management
  - more efficient resource discovery
    - unavailable primary host should not limit access
  - Relationship between proxy caching, client caching, proxy caching
- Need to build working implementation
  - Modify web server to push, proxy cache to pull
  - Build distributed registry to manage servers

Cache Consistency

- Invalidation Protocols
  - Atomic consistency / Hard to implement
- Time-to-live (TTL) fields
  - Weak consistency / Easy to implement
  - Hard to predict
- Age-based Polling (Alex)
  - Weak consistency / Easy to implement
  - Imposes less load than TTL fields

Consistency Simulator

- Popular Web files rarely change
  - » Only 5% of files change frequently
  - » Few stale hits for weak consistency protocols
- Trace-driven simulation shows:
  - » Comparable performance for all three protocols
  - » Alex slightly more efficient than TTL
    - – Both more efficient than Invalidation protocol