Design Considerations for Distributed Caching on the Internet

Renu Tewari

IBM T.J Watson
Research Center

Mike Dahlin, Harrick Vin

Department of Computer Sciences
University of Texas

Jonathan Kay

pushcache.com

June 3, 1999

Goal: True Interactive Performance

"Entire Internet on my desk"

- Human factors studies
  - ➔Interactive: <.5seconds, low variance
  - ➔Productivity improves dramatically

- We're not there
  - The Internet doesn't feel like ,right?
  - E.g. keynote.com Internet measurements
    60 metropolitan areas to 40 top commercial servers
    May 17-21, 1999: Avg 7.02 seconds

Bandwidth is improving

- But applications becoming more demanding:
  more graphics, applets, sound, video, ...

Large-Scale Web Caching

Simple idea:

- Get data from nearby copy rather than distant server

Scalable cache infrastructure

- 10’s-1000’s of caches
- 100K’s-1M’s of clients
- Deployed by ISP, 3rd party cache service, etc.

Challenge: current web caching

- Modest hit rates
- Slow hit times
- Bad consistency
- Hassle to set up

Outline

Goal: Scalable data infrastructure

- Motivation
- Design principles for scalable caching
- Data-location metadata abstraction for scalable caching
  - Scalable data-location metadata hierarchy
  - Application 1: Hint cache
  - Application 2: Push
- Conclusions

Other important problems (not addressed here)

- Consistency, caching the “uncachable,” scalable servers, faster network protocols, server selection, ...

Where Does Time Go?

Approach: study current systems

- Example hierarchies

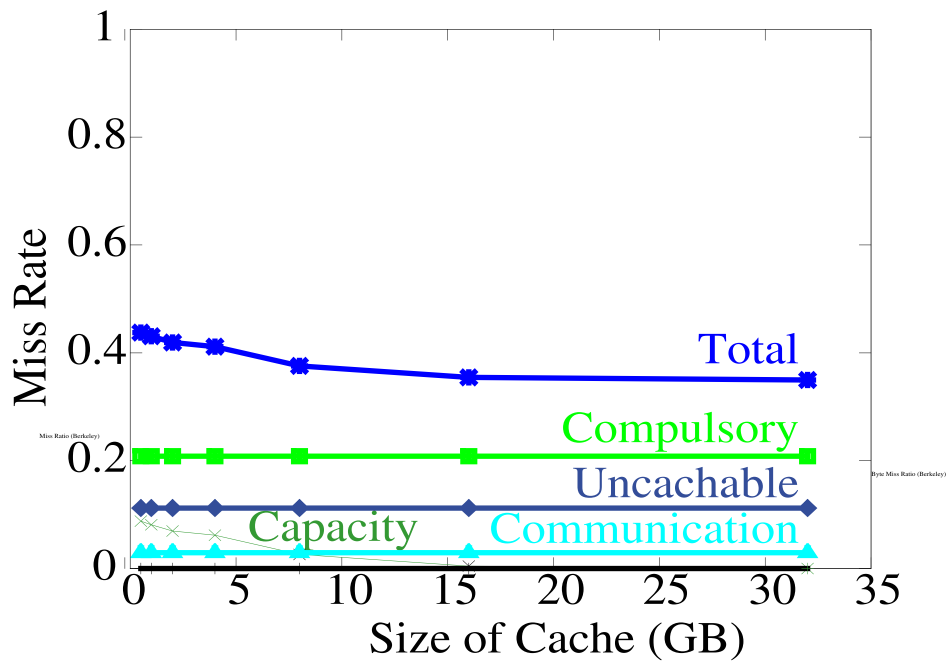
  Rousskov’s measurements of squid caches
  (*Rousskov-best, Rousskov-worst*)

  Our Squid testbed (*testbed*)

- Trace workloads
  - Berkeley -- 8K clients over 19 days
  - DEC -- 16K clients over 21 days
  - Prodigy -- 35K clients over 3 days

  Look at “cachable” requests only

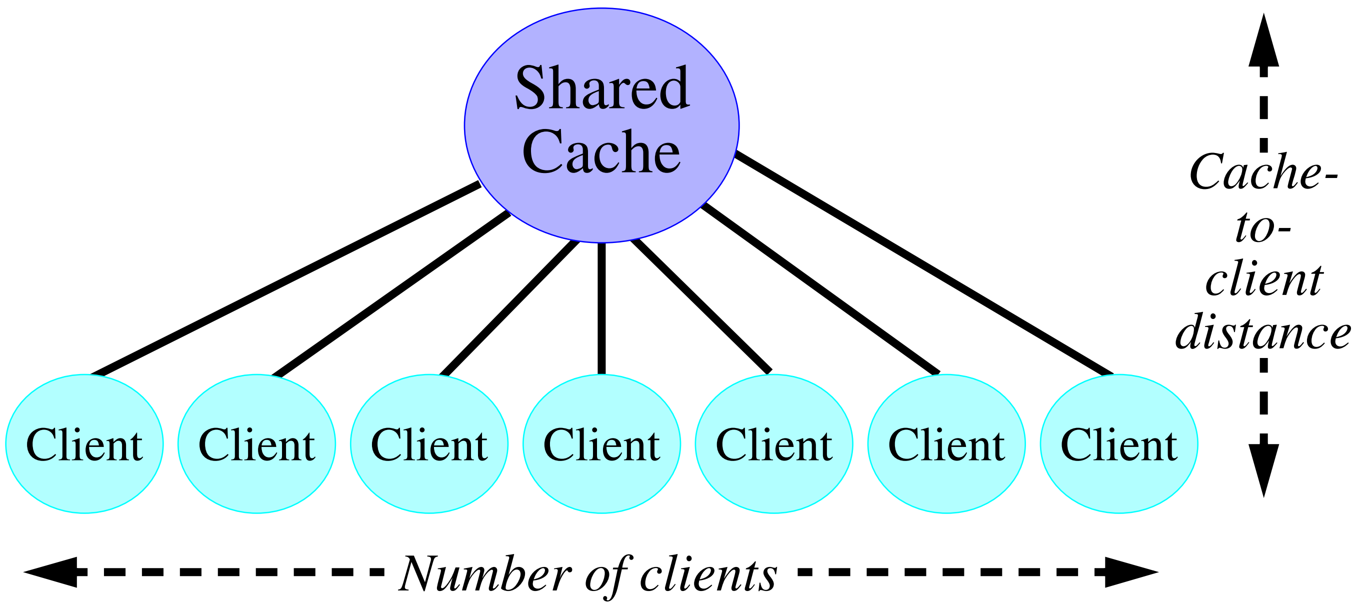
Sources of cache misses



Berkeley trace results (8K clients; 24 days)

- Compulsory: *increase sharing*
- Capacity: *not major issue (aggressive replication OK)*
- Uncachable, communication: *different study*

Sharing v. Hit Time

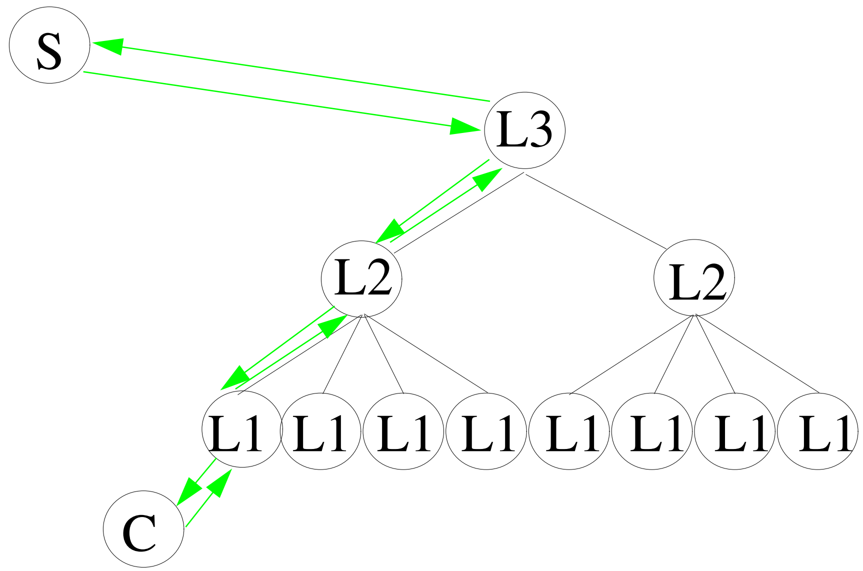


Sharing improves hit rate...

...but may hurt hit time

- When is small, fast better than large, slow?
  Can we have both?
- Current strategy: hierarchy

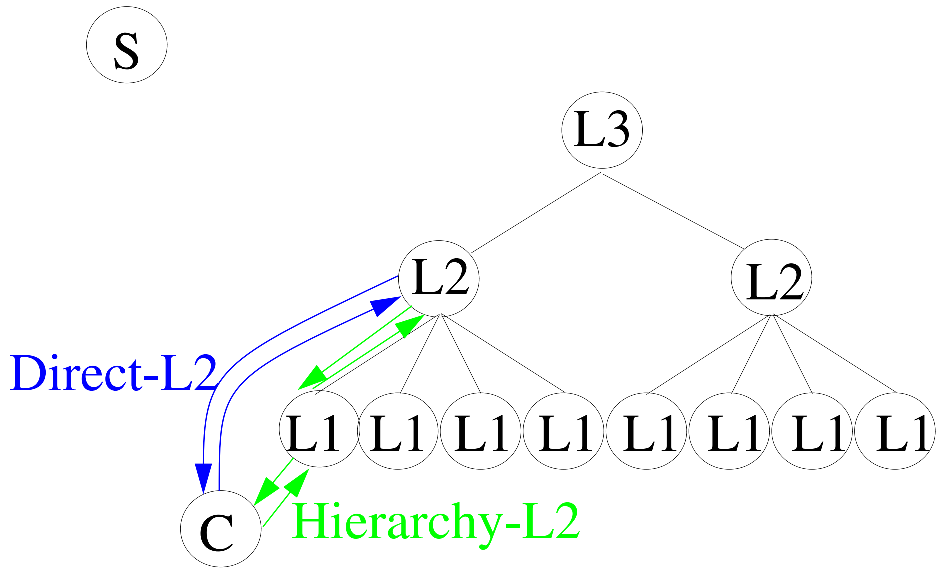
Hierarchy



Hierarchy

- Goal: balance fast access to nearby data with widespread sharing

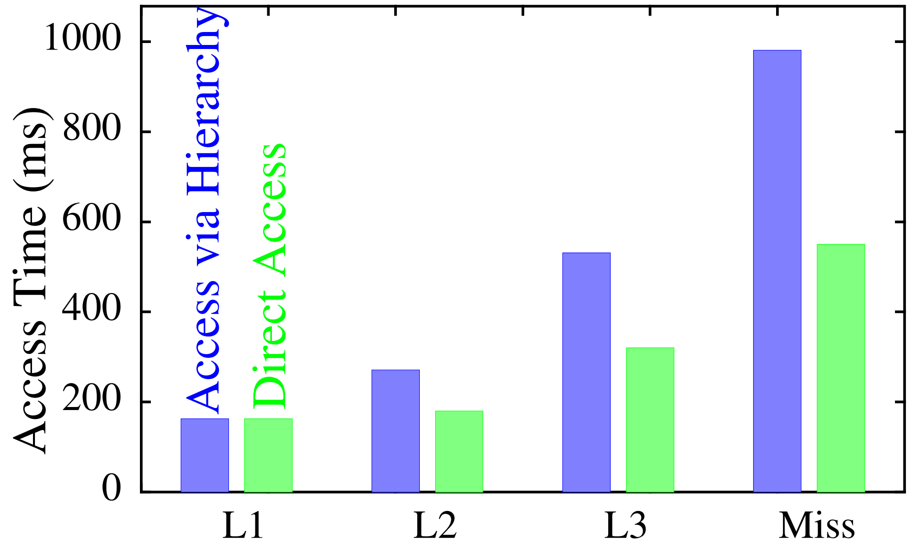
Hierarchy Performance



Compare:

- Hierarchy v. Direct
- L1 v. L2 v. L3 v. Server

Access Times in Squid Hierarchy



- Source: Rousskov-best
- Similar results for testbed hierarchy and Rousskov-worst

Principles:

- Hierarchy v. Direct

➔Reduce number of cache hops to locate and access data

- Near v. Far

➔Cache data close to clients

Design Principles for Large, Distributed Caches

Principles

1) Share data among many clients
2) Minimize the number of hops to locate and access data on hit or miss
3) Cache data close to clients

Is this just "motherhood"?

- Obvious in retrospect
- Violated by hierarchies, ICP, research proposals
    - Tension between goals
- Not just "reduce miss rate"
    - End-to-end approach:
      Also "reduce hit time", "reduce miss time"

Data-location Metadata

Data-location metadata

- Directory of *object* ⟶ *location* mappings
- Core building block for scalable caches

Application 1: Hint Cache

- Problems with hierarchies and ICP:
  - On-line search for data
  - Data returns via same path used to locate it

➔Hint cache uses directory to forward requests to nearest cache

Application 2: Push

- Problem with distributed cache:
  - Distant caches are slow (even with hint cache)

➔Use directory info as hints on when to replicate data

Other uses: consistency, coord. replacement, ...

Related Work: Data-Location Metadata

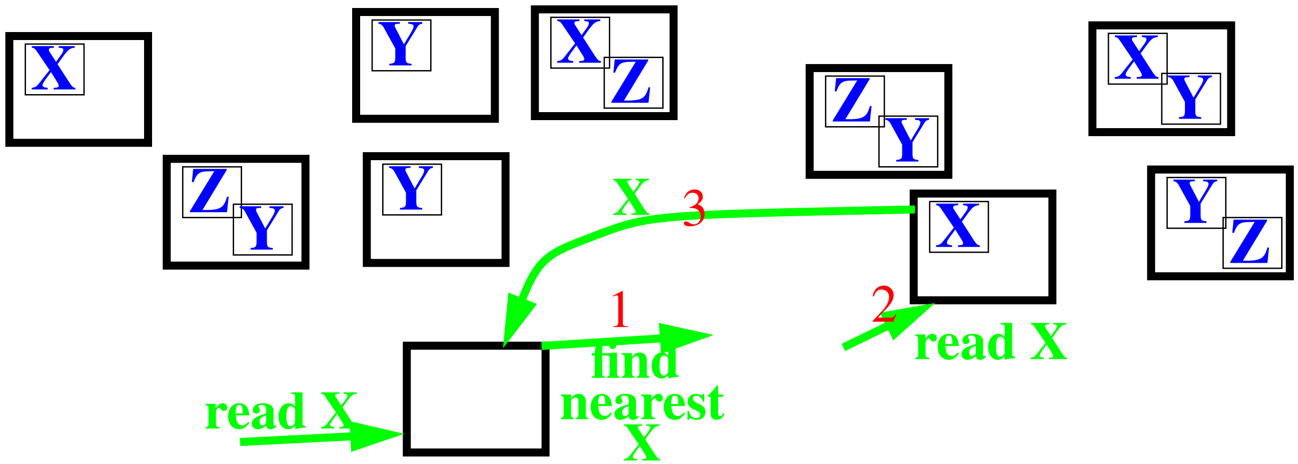
Other request-forwarding efforts

- Replace ICP: Summary Cache (Fan et. al), Cache Digests (Rousskov), ...
- Replace Hierarchy: CRISP (Chase et. al, Adaptive Web Caching (Floyd et. al), ...

Our work: Scalable data-location metadata abstraction

- Replace hierarchy
  - Focus on scalability: self configuring, scalable hierarchy
- Request forwarding + other uses
- Implementation details differ widely

Application 1: Hint Cache (Ideal)



Key ideas: local lookup, direct forwarding

1 Client consults local oracle to find nearest copy of data
2 Client sends request directly to nearest cache
3 Cache sends data directly back to client

Design principles

- Minimize hops to locate and access data on hit or miss
- Widespread sharing
- No centralized bottlenecks -- scalable

Hint Cache: Implementation Details

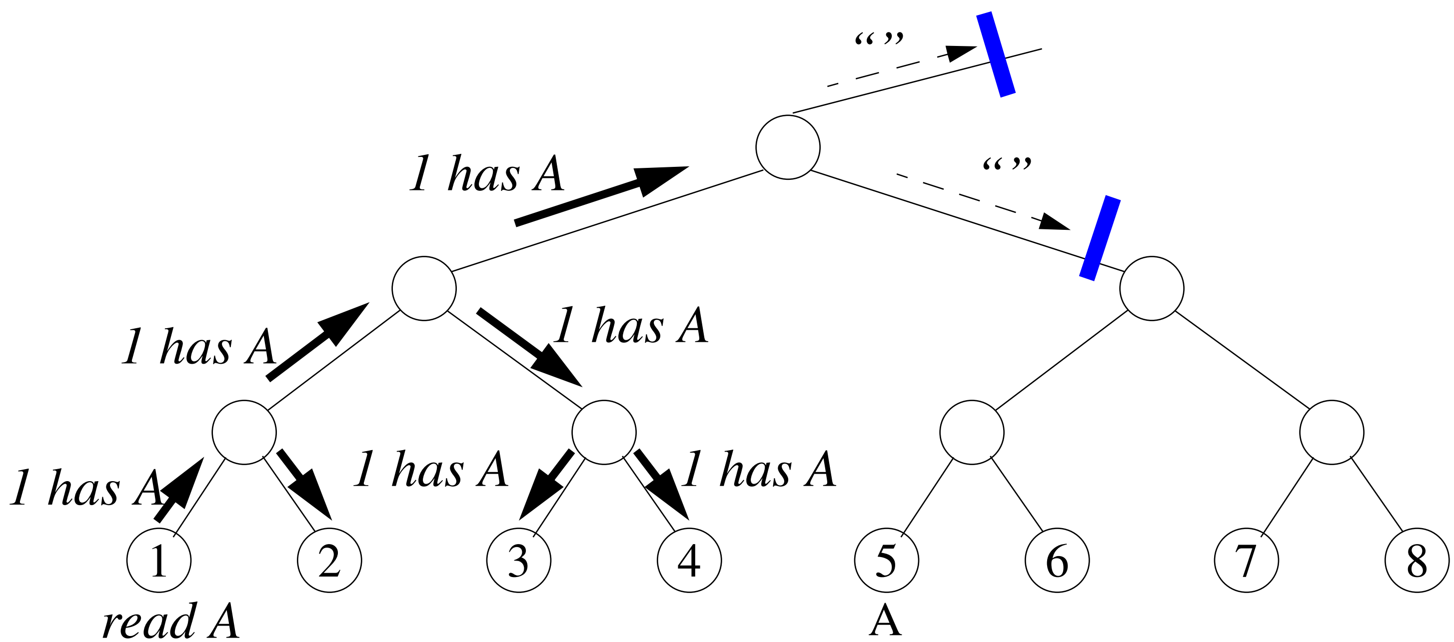
Hint cache: local cache of mappings

objId $\rightarrow$ nearest known cache

- Maximize capacity
  - ObjId = hash(URL)
  - Store mappings in n-way associative table
  - Only track nearest cache
  - Cost:
    4-16 bytes per unique object indexed
- Efficient access to date structure in memory or on disk

Propagate Hints via Metadata Hierarchy

- Asynchronous
  Batch updates for efficiency
- Prune updates
  Only propagate if new copy nearer than old one



Note: You can think of hierarchy as a tree. In reality it is redundant and self-configuring for scalability [Plaxton97]

3 Questions

Can Hint Cache approximate Oracle?

- What is performance of hint-based sharing?
- What is impact of delayed hint propagation?

See paper:

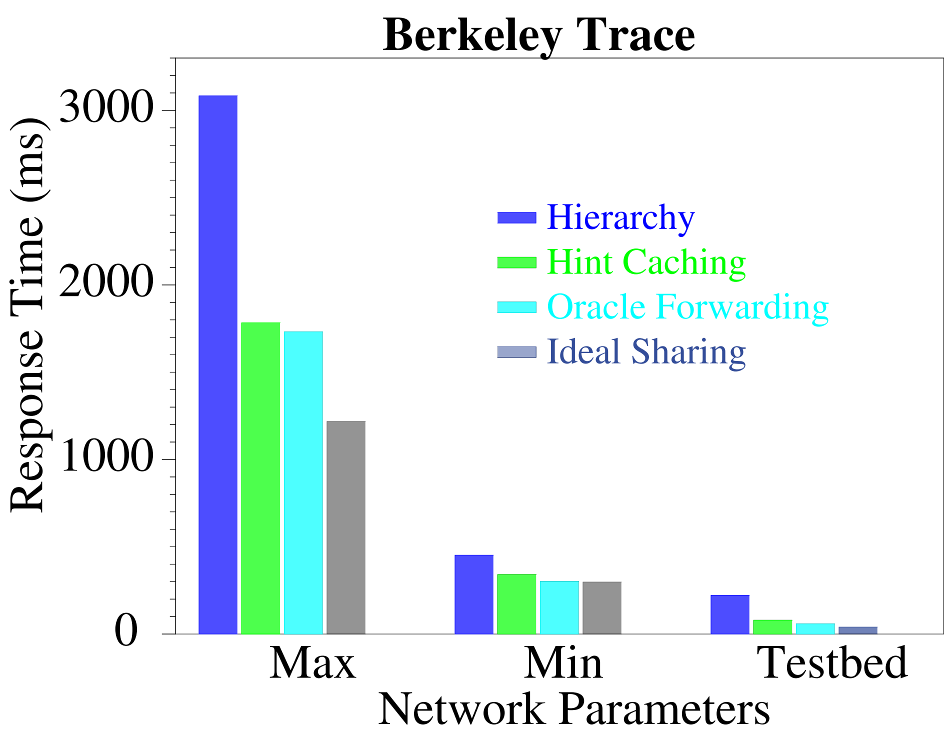
- What is impact of limited hint cache size?

Methodology

Simulate large system

- Use Berkeley, DEC, Prodigy Traces
- Use Rousskov min/max and testbed to parameterize distances to L1, L2, L3 caches
- Group clients into
  - 256 per L1 cache (or L1 metadata node)
  - 2048 per L2 cache (or L2 metadata node)
  - all per L3 cache (or L3 metadata node)
- Compare to *ideal sharing*
  - **Reality**: more clients sharing cache hurts hit time
  - **Ideal**: All clients share a nearby (L1-distance) cache
    - e.g. L3 hit rate at L1 hit time
  - ➔Unrealistic goal puts upper bound on performance

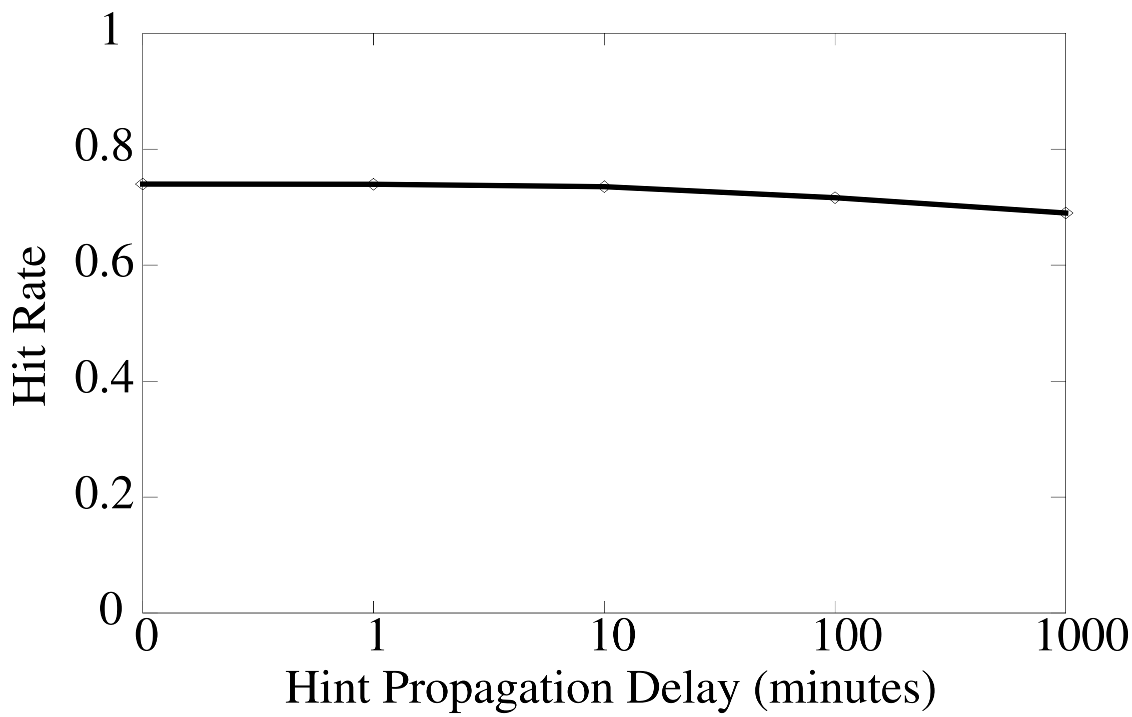
Overall performance



Q: What is performance of hint-based sharing?

- Significant speedups v. hierarchy

Hint Propagation Delay



Not sensitive to propagation delay

Summary of Hint Cache

Direct access

- Minimize hops to locate and access data

Simple, large hint cache

- Facilitate widespread sharing

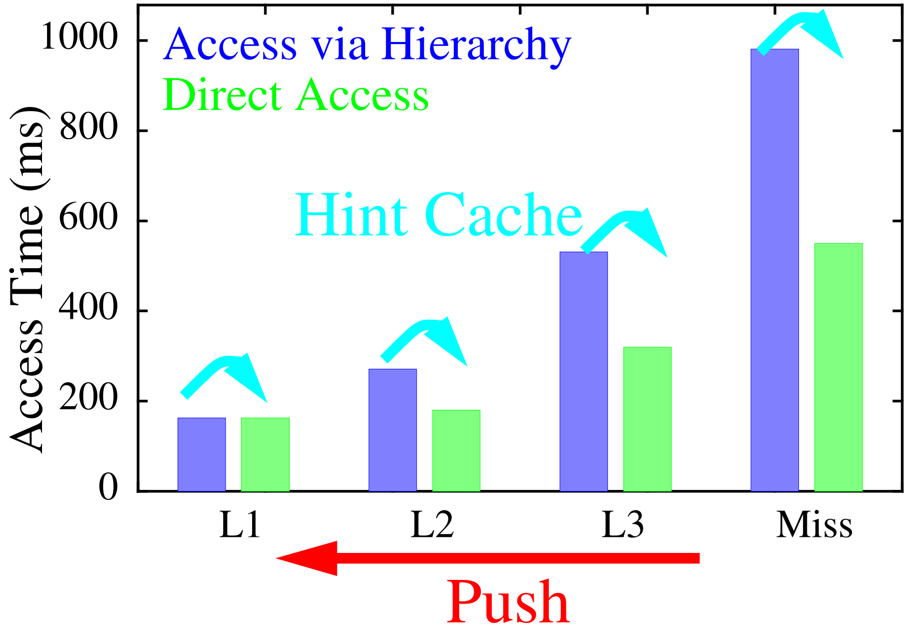
Asynchronous propagation of hints via hierarchy

- Prune updates for scalability

Application 2: Push

Goal: reduce hit time

- Make more hits to nearby caches



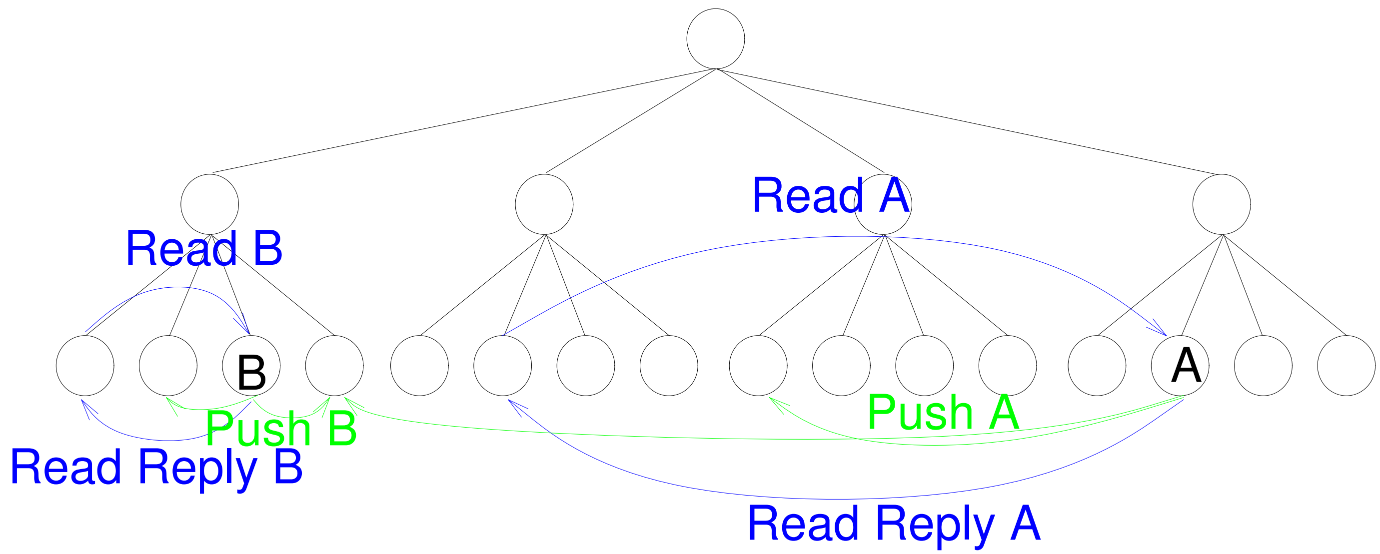
- Hint cache limits hops
- Push reduces distance of remaining hops

Push Algorithms

Wide range of algorithms

- Focus on simple algorithms that illustrate how to use data-location metadata
- Speedups
  simple “push shared”: 1.12 -- 1.25
  ideal: 1.21 -- 1.62

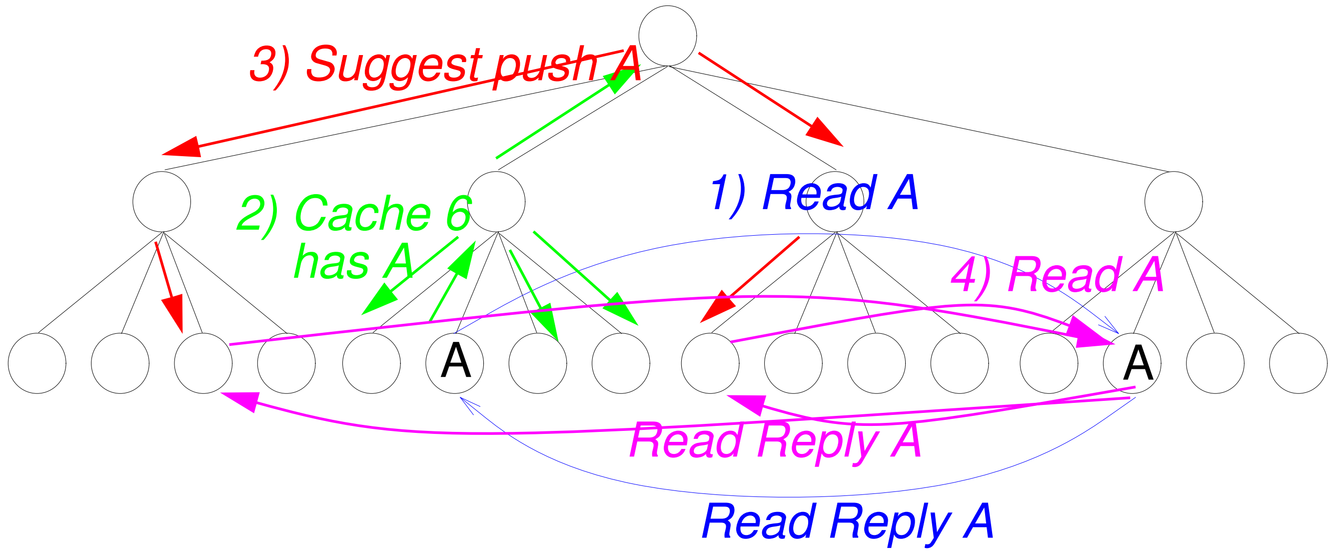
Push Shared



Idea:

- Shared data is likely to be widely shared
- Once data used by 2 subtrees, push it to all subtrees
  Tunable: push *k* copies to each subtree
- Dynamically build multicast tree for shared data

Hint Hierarchy Makes Push Easy



1) Read A
2) Propagate hint information
3) Root of subtree notices object became shared,
suggests push
4) Subtrees fetch suggested data

Conclusions

Optimize hit time, miss time (not just miss rate)

- On-line search is expensive in WAN

➔Hierarchies don't work

Data-location metadata hierarchy

- Application 1: Hint caches
- Application 2: Push
- Total speedups of 1.27 to 2.43 compared to hierarchy

Status

- Prototype based on Squid available at Beyond Browsers project page:

  http://www.cs.utexas.edu/users/bb

  and at

  http://pushcache.com