Distributed Cache Management

Paolo Romano
Cache Systems

• Caching is based on the idea of replicating frequently accessed data items in lower latency storage units:
  – performance is the main goal here…
  – …but caches can also provide better availability, resource utilization…

• Caches are widely employed at a wide variety of levels:
  
  Hardware:
  – *(Multi)* Processor caches:
    • SRAM vs DRAM
    • Local SRAM vs Remote SRAM/DRAM

  – Disk Caches:
    • RAM vs Disk

  Software:
  – *(Distributed)* File Systems:
    • Main Memory vs Disk
    • Local FS vs Remote FS

  – World Wide Web:
    • browser vs forward/reverse proxy vs Web Server
    • RAM vs Web Server Disk

  – *(Distributed)* Database Systems:
    • RAM vs Disk
    • Local DBMS vs Remote DBMS
Distributed Cache Systems

• Locality principles still drive the design process, just like in the non distributed case…

• …but now distribution raises a number of additional issues…
  – high/unpredictable communication latency:
    • we don’t want highly/unpredictable inconsistent caches!
  – possibility for cooperation among caches, but…
  – …need for system autonomy despite single caches failures
  – mutual effect of multiple caching tiers on actual miss rates:
    • trickle-down effect
  – trackability:
    • how many hits on my web page?
  – security, just like in any distributed data replication scheme
  – …
Focus of this course

• The consistency requirements play a fundamental role in the design of a distributed cache management scheme…

• Discussing the manifold formal consistency models presented in literature is out of the scope of this course - you’re already seeing them in the Distributed Systems course.

• We’ll rather take a pragmatical approach and analyze two case studies representative of weak and strong consistency constraints:
  – WWW Caching
  – Transactional Caching
Web Caching

Simplest model:
- clients are read-only, only server updates data
- content staleness is tolerated
Why Web Caching?

- **Cost**
  - Original motivation for adopting caches (esp. internationally)
  - Caching saves bandwidth (bandwidth is expensive)
  - 50% byte hit rate cuts bandwidth costs in half

- **Performance**
  - User: Reduces latency
    - RTT to cache lower than to server
  - Server: Reduces load
    - Caches filter requests to server
  - Network: Reduces load
    - Requests that hit in the cache do not travel all the way to server
Proxy Caching

- Proxy caching is one of the most common methods used to improve Web performance
  - Duplicate requests to the same document served from cache
  - Hits reduce latency, b/w, network utilization, server load
  - Misses increase latency (extra hops)
Where to cache?
Where to cache?

EVERYWHERE!
Where to cache?

EVERYWHERE!

Browser (user)
- Small: 1MB memory, 7MB disk (Netscape)
  » Note recursive caching (memory vs. disk)!
- 20% hit rate
Where to cache?

Organization (client-side proxy)
- Large: Gigabytes (with disk)
- 50% hit rate (for large client populations)
- Cache popular contents for a user community
Where to cache?

Web’s site hosting provider
- Large: Gigabytes (with disk)
- Improve access to a logical set of contents
Where to cache?

EVERYWHERE!
Web Traffic Characterization

• **Research question**: how do goals and traffic behavior shape strategies for deploying and managing proxy caches?
  – Replacement policy: what objects to retain in cache?
    • Large vs. small, relative importance of popularity and stability
  – Deployment: where to place the cache?
    • Close to server or client?
  – How many users per cache?
  – Prefetching?

• **Since the Web is in active deployment on a large-scale, Web traffic characterization is an empirical science.**
  – Science of mass behavior: observe and test hypotheses.
Zipf

• A number of studies observed that Web accesses can be modeled using Zipf-like probability distributions.
  – Rank objects by popularity: lower rank \( i \) \( \implies \) more popular.
  – The probability that any given reference is to the \( i \)th most popular object is \( p_i \)
    • Not to be confused with \( p_c \), the percentage of cacheable objects.

• Zipf says: “\( p_i \) is proportional to \( 1/i^\alpha \), for some \( \alpha \) with \( 0 < \alpha < 1 \)”.
  – Higher \( \alpha \) gives more skew: popular objects are way popular.
  – Lower \( \alpha \) gives a more heavy-tailed distribution.
  – In the Web, \( \alpha \) ranges from 0.6 to 0.8.
  – With \( \alpha=0.8 \), 0.3% of the objects get 40% of requests.
Zipf-like Reference Distributions

Probability of access to the object with popularity rank $i$:

$$p_i \sim \frac{1}{i^\alpha}$$

such that:

$$\sum p_i = 1$$

(This is equivalent to a power-law or Pareto distribution.)
Importance of Traffic Models

• Analytical models like this help us to predict cache hit ratios (object hit ratio or byte hit ratio).
  – E.g., get object hit ratio as a function of size by integrating under segments of the Zipf curve
    • …assuming perfect LFU replacement
  – Must consider update rate
    • Do object update rates correlate with popularity?
  – Must consider object size
    • How does size correlate with popularity?
  – Must consider proxy cache population
    • What is the probability of object sharing?
  – Enables construction of synthetic load generators
    • SURGE
Object Popularity Implications

- The implications of the object popularity distribution are interesting
- Cache hit rate grows logarithmically with
  - Cache size
  - Number of users
  - Time
- Easy to get most of the benefit of caching
  - Beginning of the distribution
- Hard to get all
  - Tail of the distribution
Cache Misses

• There are a number of reasons why requests miss:
  – Compulsory (50%)
    • Object uncacheable (20%)
    • First access to an object (30%)
  – Capacity (<5%)
    • Finite resources (objects evicted, then referenced again)
  – Consistency (10%)
    • Objects change (‘../today’) or die (deleted)
Uncacheable Objects

- Caches cannot handle all types of objects
  - Pages constructed from server-side programs
    - My Yahoo, E-commerce
  - Changing data
    - Stock quotes, sports scores, page counters
  - Queries
    - Web searches
  - Marked uncacheable
    - Server wants to see requests (e.g., hit counting)
  - Challenges
    - Difficult to solve, not one culprit
Caching More Caching More

• Approaches to caching more types of web content
  – Caching active data: Data sources may be dynamic, but not continuously (e.g., sports scores (Olympic web sites))
    • Snapshots generated from databases
    • Requires cooperation of server and database
  – Cache server-side program inputs and outputs
    • Need to recognize program+inputs
  – “Active caches”: Run programs (e.g., Java) at caches to produce data
    • Can handle almost anything dynamic
    • Need data sources, though&starts to become distributed server
  – Consistency mechanisms (more later)
Web Cache Consistency

“Requirements of performance, availability, and disconnected operation require us to relax the goal of semantic transparency.”
- HTTP 1.1 specification

- Any caching/replication framework must take steps to ensure that the cache does not deliver old copies of modified objects.

- Issues for cache consistency in the Web:
  - large number of clients/proxies
  - most static objects don’t change very often
  - weaker consistency requirements
    - Stale information might be OK, as long as it is “not too stale”.

Consistency Issues

• Web pages tend to be updated over time
  – Some objects are static, others are dynamic
  – Different update frequencies (few minutes to few weeks)

• How can a proxy cache maintain consistency of cached data?
  – Send invalidate or update
  – Push versus pull
Push-based Approach

- Server tracks all proxies that have requested objects
- If a web page is modified, notify each proxy
- Notification types
  - Indicate object has changed [invalidate]
  - Send new version of object [update]
- How to decide between invalidate and updates?
  - Pros and cons?
  - One approach: send updates for more frequent objects, invalidate for rest
Push-based Approaches

• Advantages
  – Provide tight consistency [minimal stale data]
  – Proxies can be passive

• Disadvantages
  – Need to maintain state at the server
    • Recall that HTTP is stateless
    • Need mechanisms beyond HTTP
  – State may need to be maintained indefinitely
    • Not resilient to server crashes
Pull-based Approaches

- Proxy is entirely responsible for maintaining consistency
- Proxy periodically polls the server to see if object has changed
  - Use if-modified-since HTTP messages
- Key question: when should a proxy poll?
  - Server-assigned *Time-to-Live (TTL)* values
    - No guarantee if the object will change in the interim
Pull-based Approach: Intelligent Polling

• Proxy can dynamically determine the refresh interval
  – Compute based on past observations
    • Start with a conservative refresh interval
    • Increase interval if object has not changed between two successive polls
    • Decrease interval if object is updated between two polls
    • Adaptive: No prior knowledge of object characteristics needed
Pull-based Approach

• Advantages
  – Implementation using HTTP (If-modified-Since)
  – Server remains stateless
  – Resilient to both server and proxy failures

• Disadvantages
  – Weaker consistency guarantees (objects can change between two polls and proxy will contain stale data until next poll)
    • Strong consistency only if poll before every HTTP response
  – More sophisticated proxies required
  – High message overhead
A Hybrid Approach: Leases

- Lease: duration of time for which server agrees to notify proxy of modification
- Issue lease on first request, send notification until expiry
  - Need to renew lease upon expiry
- Smooth tradeoff between state and messages exchanged
  - Zero duration => polling, Infinite leases => server-push
- Efficiency depends on the *lease duration*

\[
\begin{array}{c}
\text{Client} \quad \text{read} \quad \text{Proxy} \quad \text{Server} \\
\text{Get + lease req} \quad \text{Reply + lease} \quad \text{Invalidate/update}
\end{array}
\]
Policies for Leases Duration

• Age-based lease
  – Based on bi-modal nature of object lifetimes
  – Larger the expected lifetime longer the lease

• Renewal-frequency based
  – Based on skewed popularity
  – Proxy at which objects is popular gets longer lease

• Server load based
  – Based on adaptively controlling the state space
  – Shorter leases during heavy load