The Cache Sketch: Revisiting Expiration-based Caching in the Age of Cloud Data Management

Felix Gessert, Michael Schaarschmidt, Wolfram Wingerath, Steffen Friedrich, Norbert Ritter
gessert@informatik.uni-hamburg.de
Presentation is loading
The Latency Problem

100 ms

Loading…

Average: 9.3s

-1% Revenue
The Latency Problem

400 ms

-9% Visitors

-1% Revenue
The Latency Problem

500 ms

Loading...

Average: 9.3s

-20% Traffic

-9% Visitors

-1% Revenue
The Latency Problem

1s  Loading…  Average: 9,3s

-7% Conversions
-20% Traffic
-9% Visitors
-1% Revenue
If perceived speed is such an import factor

...what causes slow page load times?
State of the art

Two bottlenecks: latency and processing

High Latency

Processing Time
Network Latency

The underlying problem of high page load times

The low-latency vision

Data is served by ubiquitous web-caches
The web’s caching model
Staleness as a consequence of scalability

Expiration-based
Every object has a defined Time-To-Live (TTL)

Revalidations
Allow clients and caches to check freshness at the server
The web’s caching model
Staleness as a consequence of scalability

Expiration-based
Every object has a defined Time-To-Live (TTL)

Revalidations
Allow clients and caches to check freshness at the server

Stale Data

Diagram illustrating the web’s caching model with expiration-based and revalidations mechanisms.
The web’s caching model
Staleness as a consequence of scalability

Expiration-based
Every object has a defined Time-To-Live (TTL)

Revalidations
Allow clients and caches to check freshness at the server

Research Question:
Can database services leverage the web caching infrastructure for low latency with rich consistency guarantees?
Web Caching Concepts

Invalidation- and expiration-based caches

Expiration-based Caches:
- An object $x$ is considered fresh for $TTL_x$ seconds
- The server assigns TTLs for each object

Invalidation-based Caches:
- Expose object eviction operation to the server
The Cache Sketch approach
Letting the client handle cache coherence

Client
Expiration-based Caches
Invalidation-based Caches
Request Path
Server/DB
Cache Hits
Invalidation-based Caches
Invalidations, Records

Browser Caches, Forward Proxies, ISP Caches
Content Delivery Networks, Reverse Proxies

Client Cache Sketch
Needs Revalidation?

Bloom filter
10101010

Client Cache Sketch
Periodic every Δ seconds
at connect

Server Cache Sketch
Non-expired Record Keys
Report Expirations and Writes
Needs Invalidation?

Counting Bloom Filter
10101010
10201040

Server Cache Sketch
Transaction begin

Staleness-Minimization
Invalidation-Minimization

Client
Expiration-based Caches
Invalidation-based Caches
Request Path
Server/DB
Cache Hits
Invalidation-based Caches
Invalidations, Records

Browser Caches, Forward Proxies, ISP Caches
Content Delivery Networks, Reverse Proxies

Client Cache Sketch
Needs Revalidation?

Bloom filter
10101010

Client Cache Sketch
Periodic every Δ seconds
at connect

Server Cache Sketch
Non-expired Record Keys
Report Expirations and Writes
Needs Invalidation?

Counting Bloom Filter
10101010
10201040

Server Cache Sketch
Transaction begin

Staleness-Minimization
Invalidation-Minimization
The End to End Path of Request
The Caching Hierarchy

DB.posts.get(id) Javascript

Dynamic Web App

Client-(Browser-) Cache
Proxy Caches
ISP Caches
CDN Caches
Reverse-Proxy Cache
Orestes

Hit

Updated by Cache Sketch

Miss
Miss
Miss
Miss
Miss
Miss
Miss
Miss

Updated by the server
The End to End Path of Request

The Caching Hierarchy

GET /db/posts/{id} HTTP

Dynamic Web App

Updated by Cache Sketch

Updated by the server

Client- (Browser-) Cache
Proxy Caches
ISP Caches
CDN Caches
Reverse-Proxy Cache
Orestes
The End to End Path of Request

The Caching Hierarchy

**Cache-Hit:** Return Object

**Cache-Miss or Revalidation:** Forward Request

- Client-(Browser-)Cache
- Proxy Caches
- ISP Caches
- CDN Caches
- Reverse-Proxy Cache
- Orestes

*Updated by Cache Sketch*

*Updated by the server*
The End to End Path of Request

The Caching Hierarchy

- Client- (Browser-) Cache
- Proxy Caches
- ISP Caches
- CDN Caches
- Reverse-Proxy Cache

Miss
Miss
Miss
Miss
Miss

Return record from DB with caching TTL

Hit

Updated by Cache Sketch

Updated by the server
The End to End Path of Request

The Caching Hierarchy

- Low Latency
- Reduced Database Load
- Flash-Crowd Protection
- Higher Availability

- Client- (Browser-) Cache
- Proxy Caches
- ISP Caches
- CDN Caches
- Reverse-Proxy Cache

- Updated by Cache Sketch
- Updated by the server

- Hit
- Miss

Dynamic Web App

Low Latency
Reduced Database Load
Flash-Crowd Protection
Higher Availability
Let $c_t$ be the client Cache Sketch generated at time $t$, containing the key $key_x$ of every record $x$ that was written before it expired in all caches, i.e. every $x$ for which holds:

$$\exists r(x, t_r, TTL), w(x, t_w) : t_r + TTL > t > t_w > t_r$$
1 Slow initial page loads

- Solution: **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency
1. Slow initial page loads

- **Solution: Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency

\[
f \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad \text{Hash-Functions:} \quad k = \left\lceil \ln(2) \cdot \left(\frac{n}{m}\right) \right\rceil
\]

With 20,000 distinct updates and 5% error rate: **11 KByte**
2. Slow CRUD performance

- Solution: **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds Δ
  → *Consistency guarantee*: Δ-atomicity
Solution: **Conflict-Avoidant Optimistic Transactions**

- Cache Sketch fetched with transaction begin
- **Cached reads → Shorter transaction duration → less aborts**

Diagram:

1. **Begin Transaction**
2. **Reads** from Cache
3. **Writes** to Cache
4. **Commit**: read- & write-set versions
5. **Read all** from **Write (Public)**
4 Costly Invalidations

Solution: Invalidation Minimization
- The server Cache Sketch tracks TTLs
- Invalidation only necessary, if there are unexpired records
End-to-End Example

Client Cache Sketch
- \( b = \emptyset \)
- \( b = \{x_2\} \)

Client
- INITIALIZE
- READ \( x_3 \)
- RESPONSE false
- READ \( x_2 \)
- RESPONSE true

Expiration-based Cache
- GET \( x_3 \)
- RESPONSE false

Invalidation-based Cache
- REVALIDATE
- RESPONSE \( x_2 \)

Server
- CONNECT
- \( b_{t0} = \{x_2\} \)

Server Cache Sketch
- \( b = \{x_2\} \)
- \( t = \{(x_2, t_2), (x_3, t_3), (x_1, t_1)\} \)

Client
- QUERY \( x_3 \)
- RESPONSE false
- QUERY \( x_2 \)
- RESPONSE true

Server
- REPORT READ
- \( x_2, t_4 \)

Client
- WRITE \( x_1 \)
- PUT \( x_1 = v \)

Server
- REPORT WRITE
- \( b = \{x_1, x_2\} \)
- \( t = \{(x_2, t_4), (x_3, t_3), (x_1, t_1)\} \)

Client
- RESPONSE ok

Server
- INVALIDATE
- RESPONSE \( x_1 \)
- \( \text{inv} = \text{true} \)
**Problem**: if TTL $\gg$ time to next write, then it is contained in Cache Sketch unnecessarily long

**TTL Estimator**: finds „best“ TTL

**Trade-Off:**

**Shorter TTLs**
- less invalidations
- less stale reads

**Longer TTLs**
- Higher cache-hit rates
- more invalidations
**TTL Estimation**

**Determining the best TTL**

**Idea:**
1. Estimate average time to next write $E[T_w]$ for each record
2. Weight $E[T_w]$ using the cache miss rate

![Diagram](image)

**TTL Estimator**

Objective:
- maximize Cache Hits
- minimize Purges
- minimize Stale Reads
- bound Cache Sketch false positive rate

Client

```
Writes ~ Poisson
```

Caches

```
Writes ~ Poisson
```

Server

```
λ_m: Miss Rate
λ_w: Write Rate per record
```

```
l_m l_w
```

Reads

```
λ_m λ_w
```

```
collect TTL
```

Graph:

- No Caching
- Maximum TTL

Axes:

- write rate [ops/time unit]
- miss rate [ops/time unit]

Colors:

- Blue to yellow gradient

Legend:

- Different rates and configurations
Goal: Analysis of arbitrary caching architectures using the standard **YCSB** benchmark

- **Metrics to evaluate:** *Latency, Throughput, Cache Hits, Stale Reads, Invalidations*
Results: Simulation & real-world

Setup:

Page load times with **cached initialization** (YMCA):

- Average Latency for YCSB Workloads A and B (real):

![Graph showing load time vs hit ratios with p = 5% and p = 30%]

![Graph showing average latency with Orestes and MongoDB workloads]

Client → CDN → Orestes → MongoDB

Northern California

Ireland
Results: Simulation & real-world

Setup:

Page load times with **cached initialization** (YMCA):

![Graph showing load times with different initializations](image)

Average Throughput for YCSB Workloads A and B (real):

![Graph showing throughput](image)
The Server Cache Sketch
Scalable Implementation

- Goal: Efficient *Generation of Cache Sketch* and *Invalidation Minimization*
- **Counting Bloom Filter** and key → expiration mapping

Add key\(x\) if \(x\) unexpired

Cache Sketch for Table A
Redis

... B
Redis

... C
Redis

Get Cache Sketch:
Union
(Bitwise OR)

https://github.com/Baqend/Orestes-Bloomfilter
The Big Picture
Implementation in ORESTES

- Cache Sketch is part of ORESTES, a database-independent Backend-as-a-Service
The Big Picture
Implementation in ORESTES

Polyglot Storage

Desktop
Mobile
Tablet

Internet
Content-Delivery-Network

Reverse-Proxy Caches
Orestes Servers
Cache Sketch

mongodb
elasticsearch
The Big Picture
Implementation in ORESTES

Desktop
Mobile
Tablet

Internet
Content-Delivery-Network

Database-as-a-Service Middleware:
Caching, Transactions, Schemas, Authorization, Multi-Tenancy

Cache Sketch
Reverse-Proxy Caches
Orestes Servers

MongoDB
Redis
Elasticsearch
The Big Picture
Implementation in ORESTES

Standard HTTP Caching

Desktop
Mobile
Tablet

Content-Delivery-Network

Internet

Reverse-Proxy Caches

Orestes Servers

Cache Sketch

mongoDB

elasticsearch

Dynamic Web App

Dynamic Web App

Dynamic Web App

Dynamic Web App
The Big Picture
Implementation in ORESTES

Unified REST API

Desktop
Mobile
Tablet

Internet

Content-Delivery-Network

Reverse-Proxy Caches
Orestes Servers

Cache Sketch

mongoDB

elasticsearch
Future Work
Query-Result-Caching

Decision Model:
When is it better to cache lists of ids vs. full results and for which TTL – or not at all?

Cached Query

Caches

Create, Update, Delete

ORESTES

operation & after-image

Pub-Sub

Stream-Processing:
Which query result sets changed?

Cache Sketch & Invalidator

Changed queries

invalidate

Cache Sketch of queries

Pub-Sub
Page-Load Times

What impact does the Cache Sketch have?

<table>
<thead>
<tr>
<th>Region</th>
<th>Load Time 0.7s</th>
<th>Load Time 1.8s</th>
<th>Load Time 2.8s</th>
<th>Load Time 3.4s</th>
<th>Load Time 3.6s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALIFORNIA</td>
<td>0.7s</td>
<td>1.8s</td>
<td>2.8s</td>
<td>3.6s</td>
<td>3.4s</td>
</tr>
<tr>
<td>FRANKFURT</td>
<td>0.5s</td>
<td>1.8s</td>
<td>2.9s</td>
<td>1.5s</td>
<td>1.3s</td>
</tr>
<tr>
<td>TOKYO</td>
<td>0.5s</td>
<td>2.4s</td>
<td>4.0s</td>
<td>4.7s</td>
<td>7.2s</td>
</tr>
<tr>
<td>SYDNEY</td>
<td>0.6s</td>
<td>3.0s</td>
<td>7.2s</td>
<td>5.7s</td>
<td>5.0s</td>
</tr>
</tbody>
</table>

**Politik**
- 11. November 2014 15:42 Uhr
  - Renten könnten 2015 um zwei Prozent steigen
- 11. November 2014 10:09 Uhr
  - Europäischer Gerichtshof verweigert Deutschland darf EU-Ausländern Hartz IV verweigern
- 11. November 2014 08:45 Uhr
  - APPEL, STEFFEN
  - Obama besänftigt China

**Wirtschaft**
- 11. November 2014 07:19 Uhr
  - Guter Rat zur Geldanlage ist selten
  - Der berühmteste Wohtälter Chinas – nach eigenen Angaben
- 11. November 2014 14:30 Uhr
  - China steckt in der Wachstumsfalle

**Kultur**
- 11. November 2014 10:16 Uhr
  - Mozarts Triptychon
- 11. November 2014 08:30 Uhr
  - Der Unerschütterliche
- 11. November 2014 08:30 Uhr
  - Sandmännchen und Slasi-Mikrofone
- 13. November 2014 15:05 Uhr
  - Klare Ansage aus Harlen

*German text*
Summary

- **Cache Sketch**: dual approach to web caching for database services
  - Consistent (Δ-atomic) *expiration-based* caching
  - *Invalidation-based* caching with minimal purges

- **Keys Ideas:**
  - Maintain *Bloom filter* of potentially stale objects
  - Let clients handle cache coherence through *revalidations* when an object is contained in the filter
  - Estimate the best *TTL* based on access statistics
Thank you

gessert@informatik.uni-hamburg.de
Orestes.info
Baqend.com