Low Latency for Cloud Data Management

Felix Gessert
December 18, 2018, Universität Hamburg, DBIS Group
Presentation is loading
Web Performance
An Open Challenge With a Huge Impact

Amazon: 100 ms faster $\rightarrow$ +1% Revenue $\equiv$ $1.7$ Billion

Google: 500 ms faster $\rightarrow$ +20% Ad Sales $\equiv$ $19$ Billion

Greg Linden. Make Data Useful. 2006
Cloud-Based Web Applications

Three Sources of Page Load Time

1. Backend Processing
2. Network Delays
3. Frontend Rendering
Latency is the Problem
Throughput vs. Latency

Latency is the Problem
Throughput vs. Latency

2× Throughput = Same Load Time

½ Latency ≈ ½ Load Time
Problem Statement
Four Challenges

1. Latency of Dynamic Data
2. Direct Client Access
3. Transaction Abort Rates
4. Polyglot Persistence
Problem Statement

Research Question

How can the latency of retrieving dynamic data from cloud services be minimized in an application- and database-independent way while maintaining strict consistency guarantees?
Problem Statement
Four Challenges

1. Latency
2. Direct Access
3. Transactions
4. Polyglot Persistence
Outline

1. End-to-End Latency in Cloud-based Architectures
2. Providing Modern NoSQL Systems as a Low-Latency DBaaS
3. Cache Sketches: Solving Staleness of Reads and Queries
4. The Future of Polyglot Data Management in the Cloud
Why is end-to-end latency an open problem?
Background & Motivation
Background & Motivation

State of the Art:
- Web interface based on HTML, files, APIs, and application logic
- Performance defined by critical rendering path

Problems:
- No direct access or integration to data management
- Storage maintained manually
State of the Art:
- Service interfaces use REST & HTTP
- Web caching can reduce end-to-end latency

Problem:
- Web caching not compatible with consistent dynamic data
**Background & Motivation**

**State of the Art:**
- SaaS, PaaS, and IaaS models
- Scalability & multi-tenancy

**Problems:**
- Combination of cloud services entails **high latency**
- Common application **building blocks** often re-implemented
Background & Motivation

State of the Art:
- **Scalability** & **high availability** through NoSQL systems
- **Sharding** & **replication**
- **Database-as-a-Service** (DBaaS) model

Problems:
- Lack of common **data management abstractions**
- **DBaaS model** not supported
- **Polyglot persistence** manual & error-prone
Background & Motivation

Data Management

Critical Data
Nested Data

RDBMS
Document Store

Challenge:
How to tackle the mapping problem?

data database
NoSQL Toolbox → Decision Tree

NoSQL Tool Box

Access

Fast Lookups

Volume

RAM

Unbounded

CAP

Redis

Cassandra

Riak

Voldemort

Aerospike

HBase

MongoDB

CouchBase

DynamoDB

RDBMS

Neo4j

RavenDB

MarkLogic

CouchDB

MongoDB

SimpleDB

Complex Queries

HDD-Size

Volume

Unbounded

Query Pattern

Ad-hoc

Analytics

ACID

Availability

ElasticSearch, Solr

Hadoop, Spark

Parallel DWH

Cassandra, HBase

Accumulo

Riak, MongoDB

CAP Query Pattern

Consistency

Example Applications

Cache

Online Transaction Processing (OLTP)

Order History

Website

Social Network

Big Data

Decision Tree

Access

Fast Lookups

Volume

Unbounded

RAM

CAP

Redis

Cassandra

Riak

Voldemort

Aerospike

HBase

MongoDB

CouchBase

DynamoDB

RDBMS

Neo4j

RavenDB

MarkLogic

CouchDB

MongoDB

SimpleDB

HDD-Size

Volume

Consistency

Ad-hoc

Analytics

ElasticSearch, Solr

Hadoop, Spark

Parallel DWH

Cassandra, HBase

Accumulo

Riak, MongoDB

CAP Query Pattern

Consistency

Example Applications

Cache

Shopping-basket

Order History

OLTP

Website

Social Network

Big Data

Decision Tree

Access

Fast Lookups

Volume

Unbounded

RAM

CAP

Redis

Cassandra

Riak

Voldemort

Aerospike

HBase

MongoDB

CouchBase

DynamoDB

RDBMS

Neo4j

RavenDB

MarkLogic

CouchDB

MongoDB

SimpleDB

HDD-Size

Volume

Consistency

Ad-hoc

Analytics

ElasticSearch, Solr

Hadoop, Spark

Parallel DWH

Cassandra, HBase

Accumulo

Riak, MongoDB

CAP Query Pattern

Consistency

Example Applications

Cache

Shopping-basket

Order History

OLTP

Website

Social Network

Big Data

Decision Tree

Access

Fast Lookups

Volume

Unbounded

RAM

CAP

Redis

Cassandra

Riak

Voldemort

Aerospike

HBase

MongoDB

CouchBase

DynamoDB

RDBMS

Neo4j

RavenDB

MarkLogic

CouchDB

MongoDB

SimpleDB

HDD-Size

Volume

Consistency

Ad-hoc

Analytics

ElasticSearch, Solr

Hadoop, Spark

Parallel DWH

Cassandra, HBase

Accumulo

Riak, MongoDB

CAP Query Pattern

Consistency

Example Applications

Cache

Shopping-basket

Order History

OLTP

Website

Social Network

Big Data

Decision Tree

Access

Fast Lookups

Volume

Unbounded

RAM

CAP

Redis

Cassandra

Riak

Voldemort

Aerospike

HBase

MongoDB

CouchBase

DynamoDB

RDBMS

Neo4j

RavenDB

MarkLogic

CouchDB

MongoDB

SimpleDB

HDD-Size

Volume

Consistency

Ad-hoc

Analytics

ElasticSearch, Solr

Hadoop, Spark

Parallel DWH

Cassandra, HBase

Accumulo

Riak, MongoDB

CAP Query Pattern

Consistency

Example Applications

Cache

Shopping-basket

Order History

OLTP

Website

Social Network

Big Data
How can cloud data management be unified & combined with low latency?
Orestes: Goals
A Data Management Middleware for Low Latency

Database Independence
DBaaS & BaaS Functionality

Scalable, Available, Multi-Tenant
Low Latency with Tunable Consistency
Orestes Concept

Overview

- Unmodified Database Systems
- DBaaS/BaaS Middleware
- Data and Default Modules
- Unified REST API
- Web Caching for Low Latency
- Web and Mobile Applications

Heterogeneous Data Stores

Scalable Data Management Platform (Multi-Tenancy, Scaling, Caching, Failover, ...)

[GBR14, GB13]
How can dynamic data be accelerated through web caching?
The Web‘s Caching Model

Expiration-Based Caches:
› An object $x$ is considered fresh for $TTL_x$ seconds
› Server assigns TTLs for each object

Invalidation-Based Caches:
› Expose object eviction operation to the server

[GSW+15]
Web Caching for Data Management
Overview of Cache Sketch Method

Without Cache Sketch: Stale Cached Data

Validate Freshness

Expiration

Invalidation

Cache

Compact Cache Sketch

Add to Server Cache Sketch

Data Cached for Fixed TTL

[GSW+15, GSW+17]
The Cache Sketch Approach

Minimize Staleness

1. Initialization from Cache
2. Δ-Atomic Consistency
3. Cache-Aware Transactions

Minimize Invalidations

4. Invalidation Minimization

Client

Client Cache Sketch

Expiry-based Caches
Invalidation-based Caches

Request Path

Cache Hits

Invalidations, Objects

Server/DB

10101010

Bloom filter

Report Expirations and Writes
Needs Revalidation?

Non-expired Object Keys

Counting Bloom Filter

Server Cache Sketch

10101010

10201040

Initialization from Cache

Periodic every Δ seconds at transaction begin

Minimize Staleness

Minimize Invalidations

[GSW+15]
Cache Sketch
Main Properties

To ensure $\Delta$-atomicity the Cache Sketch at time $t$ contains key($x$) of every object $x$ that was written before it expired in all caches.

[GSW+15, GSW+17]
Cache Sketch

Construction

To ensure compactness the Cache Sketch stores \( n \) keys in a Bloom filter with \( m \) bits, \( k \) hash functions and a false positive rate of \( f \approx 1 - \exp\left(\frac{k \cdot n}{m}\right) \).

Example

20,000 entries & 5% false positives
↓
11 KB in size

[GSW+15, GSW+17]
Controllable Consistency Levels

Cache Sketch Guarantees

- Linearizability
- Sequential Consistency
- Causal Consistency
- PRAM
- Δ-Atomicity
- (Δ,p)-Atomicity

Controllable Staleness
Default Guarantees
Opt-in Guarantees
With Cache Bypassing

[WG+18, GWR, GWFR16, GR16, FWGR14, FWGR14]
Determining TTLs

Trade-Off

Shorter TTLs

\[\downarrow \text{Invalidations}\]
\[\downarrow \text{False Positive Rate}\]

Longer TTLs

\[\downarrow \text{Cache Misses}\]

[GSW+17, GSW+15]
TTL Estimation
Optimizing Expiration & Cacheability

1. Collect **workload statistics** for reads and writes
2. Estimate **time to next write** $E[T_w]$ or mark uncacheable

**Constrained Adaptive TTL Estimator**
- Ideal for Poisson Processes

**C-LM Model**
- Quick Adaption to Changes

**LWMA Estimator**
- Converges for Static Workloads

**EWMA Estimator**
- Highly Space-Efficient

[GSW+15]
Evaluation Results
Simulation & Benchmarking

Setup:

- Client
- CDN
- Orestes
- MongoDB

Page load times with cached initialization (Simulation):

Average throughput for YCSB workloads A and B (YCSB benchmark):

[GSW+15]
Evaluation Results

Industry Evaluation of Commercial Implementation

[GSW+17, Ges17]
How can object caching be extended to query results?
Query Caching

Challenges

Invalidation Detection
When do query results change?

Cache Coherence
How to apply Cache Sketches to queries?

Query Result Representation
What is the best result structure for caching?

[GSW+17]
Invalidation Detection
Cache Coherence for Query Results

Scalable Streaming System (InvaliDB)

Query Events
- Add
- Change
- Remove

Cached Query Result

Updated Cache Sketch

Query Expression

Normalized String

Real-Time Queries

Update Orestes

[WRG18, WGF+17, GSW+17, WGFR16]
Learning Result Representations
Handling Changes to Query Results

ID Lists
[url₁, url₂, url₃]

Object Lists

[{id: obj₁, name: "alice"}, {id: obj₂, name: "bob"}, {id: obj₃, name: "eve"}]

Solution: Cost-based decision model weighs expected round-trips vs. invalidations

[GSW+17]
Evaluation Results
Query Caching for YCSB-Based Workloads

**Throughput** with growing request parallelism:

- **11x Throughput Improvement**

**Average end-to-end query and read latency:**

- **47x Lower Query Latency**
- **9x Lower Read Latency**

[GSW+17]
Can Cache Sketches improve transaction performance?
Problem of Optimistic Transactions
Abort Rates Depend on Latency

Transaction Abort Rates Increase Exponentially with Latency

\[ \text{Transaction Duration with Retries in s} \]

\[ \text{Accessed Objects n} \]

\[ 150 \text{ ms} \quad 100 \text{ ms} \quad 50 \text{ ms} \]

\[ 10 \text{ ms} \]

[GBR14]
Distributed Cache-Aware Transactions
DCAT Solves Latency Problem

1. Cache Sketch: **staleness barrier** at transaction begin
2. Shorter duration through **cached reads**

![Diagram showing the process of distributed cache-aware transactions](image)

[GBR14]
Results
Simulation-Based Abort Analysis

15× Faster Transactions
7× More Objects Before Exceeding 2 Seconds
Can polyglot data management be automated in the future?
Vision
Automated Choice of Databases

Polyglot Persistence Mediator

Application

Annotated Schema

Latency < 20ms

DB₁, DB₂, DB₃

[SGR15]
Towards Automated Polyglot Persistence
Three-Step Process

1. Requirements

Requirements specified as **SLA annotations** for schemas (based on *NoSQL Toolbox*)

2. Resolution

Find or provision a suitable combination of databases through *ranking algorithm*

3. Mediation

Mediate *data allocation* and *database operations* between applications and databases

Counter

- Top-k Query
- 20 ms Write Latency

Update

Counter

- Redis
- MongoDB

Counter

Redis

[SGR15]
Evaluation Results
Case Study

Scenario:
News Articles With Impression Counts

2.1×
Higher Throughput

10 ms
Predictable Write Latency

[SGR15]
Future Work

Three Promising Areas

**Proactive SLA Enforcement**
- Monitor & predict database behavior
- **Action**: change routing, live migration, polyglot scaling

**Reinforcement Learning of Caching Decisions**
- Learn best **TTLs** for any workload
- Applications define goals

**Polyglot Transaction Processing**
- Optimal choice of **concurrency** & **commit protocol** – across DBs

[SKE+18, SG16]
What are the main contributions?
Publications (1/4)


Publications (2/4)


Publications (3/4)


Publications (4/4)


Main Contributions

Summary

1. **Cache Coherence** for Files, Records & Queries
2. **TTL Estimation** & Result Structure
3. Unified Data Management Interface
4. Database-Independent DBaaS and BaaS
5. Scalable Cache-Aware Transactions
6. Polylgot Persistence Mediation