Towards a Scalable and Unified REST API for Cloud Data Stores

Felix Gessert
felix.gessert@gmail.com
Context: Database-as-a-Service & Backend-as-a-Service
Cloud Databases

- Backend-as-a-Service
  - Managed NoSQL Databases
    - Orestes
    - MongoHQ
    - Cloudant
    - SQL Azure
    - Amazon RDS
    - Xeround
  - Managed RDBMS
    - Heroku
    - Amazon Web Services

- Cloud-only DBaaS-Systems
  - DynamoDB
  - BigQuery
  - Google F1
  - EMR

- Database-as-a-Service

- Platform-as-a-Service

- Infrastructure-as-a-Service

- Analytics-as-a-Service

- Storage APIs

- Cloud-Deployment of DBMSs
Motivation

3-Tier Architecture

- Web Server
- Web Server
- Data (e.g. JSON)

Database

Application

Client

Redundance
Applications reimplement backend functionality

Database Dependence
Strong coupling between application and database system

Performance & Scalability
Error-prone application-specific scaling policies

Difficult Development
Both client and server have to be implemented
When increasing load time of search results by **500ms**, traffic decreases by 20%.

Study by **Google**
Vision: ORESTES

- Automated Choice of Database System (Polyglot Persistence)
- Extensible High-Level Database and Backend Abstractions:
  - User-Management
  - Access Control
  - Schema
  - Transactions

Unified REST API

Cloud

DB-Cluster

Orestes

REST Server

Unified REST API

Internet

Rich Client

Low Latency

Transparent Caching
Outline

- Orestes Middleware Architecture:
  - Stateless REST servers exposing the unified REST API

- High-Level Backend Abstractions:
  - Authentication, Authorization, Data Validation, Stored Procedures, Standard Schemas, Push Updates, Files

- High-Level Database Abstractions:
  - Polyglot persistence
  - Schema Management
  - Transparent web-caching
  - Cache-aware optimistic transactions
Outline

- Orestes Middleware Architecture:
  - Stateless REST servers exposing the unified REST API

- High-Level Backend Abstractions:
  - Authentication, Authorization, Data Validation, Stored Procedures, Standard Schemas, Push Updates, Files
  - Polyglot persistence
  - Schema Management
  - Transparent web caching
  - Cache-aware optimistic transactions

Thesis:
Modern applications can be developed on a *unified REST API* with superior performance- and functional properties provided on top of existing database systems.
Unified REST API

- Platform-specific interfaces map to unified REST API

REST API leverages existing HTTP infrastructure
- Load-Balancer *(stateless communication REST constraint)*
- Web-Caches *(caching REST constraint)*
Unified REST API

- Defined in REST specification

Orestes-Server

db/bucket/oid Resource

public OrestesObject

load(ObjectInfo objInfo)

Data Module for specific DB

### Orestes Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td></td>
<td>Orestes Methods</td>
</tr>
<tr>
<td>crud</td>
<td></td>
<td>Create Read Update Delete (CRUD) Object Methods</td>
</tr>
<tr>
<td>GET</td>
<td>/db/{bucket}</td>
<td>Create object</td>
</tr>
<tr>
<td>GET</td>
<td>/db/{bucket}/{oid}</td>
<td>Get object</td>
</tr>
<tr>
<td>PUT</td>
<td>/db/{bucket}/{oid}</td>
<td>Replace object</td>
</tr>
<tr>
<td>DELETE</td>
<td>/db/{bucket}/{oid}</td>
<td>Deletes the object</td>
</tr>
</tbody>
</table>

### Object oriented methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/db/all_schemas</td>
<td>Get all available class schemas</td>
</tr>
<tr>
<td>POST</td>
<td>/db/all_schemas</td>
<td>Create new class schemas and patch existing class schemas</td>
</tr>
<tr>
<td>PUT</td>
<td>/db/all_schemas</td>
<td>Replace all currently created schemas with the new ones</td>
</tr>
<tr>
<td>DELETE</td>
<td>/db/all_schemas</td>
<td>Remove all currently created schemas</td>
</tr>
<tr>
<td>GET</td>
<td>/db/bucket/schema</td>
<td>Get the class schema</td>
</tr>
<tr>
<td>POST</td>
<td>/db/bucket/schema</td>
<td>Update the class schema</td>
</tr>
</tbody>
</table>

Implementation Notes
Modify the schema definition of the class by adding all missing fields

Response Class
Model | Model Schema | object
Unified REST API

Defined in REST specification

Example

Create a new object

- POST /db/:bucket
- JSON-Object
- 201 Created
Unified REST API

Defined in REST specification

Example

**Load object**

> GET /db/:bucket/:oid

< JSON-Object
Unified REST API

Defined in REST specification

Example

*Perform Paginated Query*

```plaintext
> GET /db/:bucket?query&start=0&count=-1
< List of matching IDs
```
Downsides of REST database APIs

- **Request-Response** model limits throughput
  - *Proposal*: Multipart Caching
  - Multiplexing is core of HTTP/2.0
- Timestamps have [s]-precision
  - *Proposal*: ISO-8601 dates
- Networking infrastructure bugs
  - *Solved*: Non-persistent methods
  - *Solved*: TCP-Deadlock caused by *Delayed ACK* and *Nagle* interaction
Middleware Architecture

- Infrastructure-as-a-Service Cloud
- REST-Server
- Scalable Database System
- HTTP
- REST-Server
- Browser
- Mobile Client
- Global Content Delivery Network
- Application Server
- Proxy Caches
- Platform-as-a-Service Cloud
- Mobile Client
- ISP
- Load Balancer
- Cache
- REST-Servers
- DB protocols
- Scalable Database System
- REST-API
- Transactions
- Schema Management
- Cache Consistency
- Auto-Scaling
- Multi-Tenancy
- Security and Access Control
- Mobile Client
- Application Server
- Platform-as-a-Service Cloud
- Mobile Client
High-Level Backend Abstractions

- Access Control (Authorization)
  - User
    - PUT `db/posts/{id}`
    - User-Token
    - JSON Object
  - Orestes
    - `db.posts.update(..., _acl.allow:{$in:... _acl.deny:{$nin:...})`
  - MongoDB
    - `schema-level` ACL check using Token
    - `object-level` ACL check through condition

- Data Validation, Stored Procedures
  - `OnUpdate`, `OnCreate`, `OnDelete`, `Validate` Handler
  - Same JS API as client
  - Executed in `Node.JS` process in isolated Docker container
High-Level Database Abstractions

- **Schema Management**
  - Primitive
  - Collection
  - Reference
  - Embedded
  - JSON
  - Validate schema for every operation on objects

- **Optimistic Transactions**
  - Orestes-Servers perform version checks on write-sets (BOCC+)
  - Applicable to all systems with consistent read option

- **Coherent web-caching**

- Annotation-based **polyglot persistence**
**Cache Coherence: The Cache Sketch**

- *Bloom filter* of stale objects for consistent caching

For each request:
- Check in Cache Sketch if object is stale

**Cache Hit Maximization**

On connect:
- Piggyback *Cache Sketch*

- *Cache Hit*: fresh objects
- *Revalidation*: stale objects

- *Write object*

- *Invalidate* only if non-expired

**Invalidation Minimzation**
Polyglot Persistence

Application

Schema annotations contain SLA

Parse SLA & route data

Orestes servers

Resolve mapping

Polyglot Persistence Mediator

Manage materialisation

Redis

MongoDB

Db4o
Results:
Article-Objects with Impression Count

Speedup with PPM:
- 50-1000% (depending on throughput)
- 66% performance of Varnish
Cloud Evaluation of ORESTES

Amazon EC2 Ireland
- Client Machine
- Client Machine
- Client Machine
- Web Cache

EC2 USA
- Orestes Server
- Versant DB

165 ms

30,000 Objects
500 Req./Clients
10/1 Read/Write

1. Run
2. Run
3. Run

- Squid 2
- Squid 3
- Squid 3 patched
- Microsoft TMG
- No cache
- VOD native
- Orestes

Time [s]

250
Cloud Evaluation of ORESTES

Amazon EC2 Ireland

Client Machine
Client Machine
Client Machine

Web Cache

165 ms

EC2 USA

Orestes Server
Versant DB

30,000 Objects
500 Requests/Client
10/1 Read/Write

Time [s]
Summary

- Backend-as-Service Abstractions
- Database-as-Service Abstractions

- Users & Authorization
- Standard Schemas
- Handlers & Stored Proc.
- Schema Management
- Coherent Caching
- Polyglot Persistence

- Unified REST API
  - „Middleware“ design to enhance existing database systems
  - Decouples applications from concrete systems
  - Adds critical functional and non-functional capabilities
Future & Current Work

- **Unfied REST API:**
  - Scalable Streaming Queries

- **Caching:**
  - Adaptive Cache-Expirations for optimal Cache Sketches

- **Polyglot Persistence:**
  - Design of functional- and non-functional annotations & mediator implementation

- **Transactions:**
  - Database-independent validation, stochastic analysis of Cache Sketch effects
Baqend

- Orestes as a startup
Thank you

Contact:

gessert@informatik.uni-hamburg.de

http://orestes.info

http://baqend.com