AutoShard - Declaratively Managing Hot Spot Data Objects in NoSQL Data Stores

Andreas Thor
Deutsche Telekom
University of Applied Sciences for Telecommunication, Leipzig

Stefanie Scherzinger, OTH Regensburg

HPI Symposium "Operating the Cloud", Oct. 28, 2014
Today's Web Applications

Energy | Description
--- | ---

165,427 votes on 298 questions from 13,687 people

Popular questions: Presentation view

"Will you finally end taxpayers subsidies for fossil fuels industries?"
Anna, Chicago | View responses (2) | Post a response

"Will you President to make the United States a leader in clean and renewable energy?"
Joshua Coppen, Jacksonville, FL | View response (1) | Post a response

"Will you defend the Clean Air Act against big business?"
Shaun Dakin, Dakin & Associates Falls Church, VA | View responses (4) | Post a response

Vote up
Write a response
NoSQL Data Stores

- Flexible data model
  - Query functionality mostly sufficient
- Impressive scalability handles large amounts of data
  - Build for massively parallel reads
- Strongly consistent writes and reads against single entities
  - Appropriate for most web scenarios (#reads >> #writes)

Google Cloud Datastore, Simple DB, CouchBase, …
Hot Spot Data Objects

- Frequently accessed/updated data objects
- Performance vs. scalability
  - Impressive scalability handles large amounts of data
  - Limited write throughput on single data objects (\( \approx 5-10/\text{sec} \))
- Frequently updated objects … not entirely new problem 😊
  - Examples: available seats on a plane, overall account balance, …
  - Previous work on hot spot objects for RDBMS
- New aspects for NoSQL data stores
Agenda

- Motivation
- NoSQL Data Stores and Hot Spot Objects
- Sharding
  - Property Sharding
  - Entity Group Sharding
- AutoShard
  - Architecture
  - Dynamic AST modification
- Evaluation
- Summary and Outlook
NoSQL data stores and Hot Spot Objects

- **Optimistic concurrency control**
  - „Execute Txs immediately, check at commit for conflicts“
  - No wait locks at the expense of possible aborts → Retry!
  - Appropriate for most web scenarios (#reads >> #writes)

- **Database as a Service**
  - Developers cannot modify the database in DaaS settings
  - Hot spot objects must be handled on the application level

- **No strong consistency**
  - Eventual consistency (clients may read stale data)
Property Sharding

- Logical property value is stored using multiple shards (i.e., physical values)
  - Writes are distributed across all shards
  - Aggregated read over all shards
- Example: Vote counter for questions
  - “VoteUp” on any shard; sum all shards to get number of votes

<table>
<thead>
<tr>
<th>Qld</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qld</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>0</td>
</tr>
<tr>
<td>A.2</td>
<td>0</td>
</tr>
<tr>
<td>A.3</td>
<td>0</td>
</tr>
</tbody>
</table>
Property Sharding: Implementation

- **Initialization**
  - Set shard value to neutral element

- **Write (set specific value)**
  - Set one (chosen at random) shard to specific value
  - Set all other shards to neutral element

- **Update (based on current value)**
  - Update one shard (chosen at random) using update function

- **Read**
  - Read all shards and aggregate using fold function

- **Manual implementation**
  - **Laborious**: complex implementation (and testing)
  - **Unnecessary** (overhead) for many objects / properties
Entity Group Sharding

- Entity group (set of entities) is stored using multiple shards (i.e., physical values)
  - Writes are distributed across all shards
  - Aggregated union over all shards
- Example: Responses for questions
  - “AddResponse” on any shard (subset of responses)
  - Unify all shards to get the complete list of responses
AutoShard

- Object mapper with **automatic** and **adaptive** sharding
  - Java objects ↔ NoSQL entities (in BigTable-like DS)
  - Automatic sharding on logical schema avoids scalability bottlenecks / write contention
  - Adaptive, i.e., automatic identification of hot spots

- Two kinds of sharding
  - Property sharding: distribute atomic values
  - Entity Group sharding: distribute sets of entities

- Easy-to-use
  - Definition using Java annotations
  - Implementation by automatic AST transformation
Example Annotations: Class question

```java
@Entity
class Question {
    @Id private int id;
    private String question;
    private String author;
    private List<Response> responses;
    @Shardable (neutral=0)
    private int votes = 0;

    @ShardMethod
    public void voteUp() {
        this.votes++;
    }

    @ShardFold
    public static int foldVotes(int x, int y) {
        return x + y;
    }
    ...
}
```
AutoShard Architecture: Deploy Time

- "Injection" of sharding functionality during compile time
- Automatic program modification based on annotations

Diagram:

1. Groovy Parser
2. Abstract Syntax Tree (AST)
3. Modified AST
4. Groovy / Java Compiler
5. Java bytecode with sharded values
6. Java Class annotated as `@Entity`
7. .groovy
8. .class
9. AutoShard Framework
10. AutoShard AST Transformer
Evaluation

- 2,000 users, 75 voting requests per sec across 16 questions
  - w/ Tx retry: re-start failed transactions (exception handling)

- 25% failed transactions without sharding vs. 4% failed transactions with AutoShard
Current Work: Adaptive Sharding

- Automatic identification of …
  - Properties / entities that should be sharded
  - Sharding parameters, e.g., number of shards

- Statistics
  - Time-based: Number of put requests per time and per entity
  - Exception-based: Number of raised exceptions

- Implementation
  - Add logging statements during AST transformation
  - Store all / aggregated statistics in MemCache
  - Rule-based decision
AutoShard Architecture: Run Time

- Put/get requests are logged into MemCache
  - Fast, distributed in-memory cache
- Workload statistics used to apply sharding on-demand

Diagram:

- .class (Java Class bytecode)
- Java Object put/get requests
- Java object
- AutoShard Data Store Service
- AutoShard Statistics Service
- Logging
- Aggregated workload statistics
- DS Entity put/get requests
- Entities
- Data Store
Summary / Outlook

- **AutoShard** = A novel object mapper that can declaratively manage hot spot data objects
  - Avoids schema-inherent performance bottlenecks in NoSQL-based web applications
- Implements database techniques (sharding) using programming techniques (annotation + AST transformation)

- **Adaptive Sharding**
  - When is sharding required (80/20 rule)?
  - What is good number of shards?
  - Background processes for compaction, ...