Developing polyglot persistence applications

Chris Richardson
Author of POJOs in Action
Founder of the original CloudFoundry.com
@crichardson
chris@chrisrichardson.net
http://plainoldobjects.com
Presentation goal
The benefits and drawbacks of polyglot persistence
and
How to design applications that use this approach
About Chris
About Chris()
About Chris
About Chris

http://www.theregister.co.uk/2009/08/19/springsource_cloud_foudry/
Agenda

• Why polyglot persistence?
• Optimizing queries using Redis materialized views
• Synchronizing MySQL and Redis
Food To Go Architecture - 2006

- Hungry Consumer
- Restaurant

Order taking

Restaurant Management

MySQL Database

Success = inadequate
Solution: Spend Money

http://upload.wikimedia.org/wikipedia/commons/e/e5/Rising_Sun_Yacht.JPG

OR

http://www.trekbikes.com/us/en/bikes/road/race_performance/madone_5_series/madone_5_2/#

@crichardson
Solution: Use NoSQL

**Benefits**
- Higher performance
- Higher scalability
- Richer data-model
- Schema-less

**Drawbacks**
- Limited transactions
- Limited querying
- Relaxed consistency
- Unconstrained data
### Example NoSQL Databases

<table>
<thead>
<tr>
<th>Database</th>
<th>Key features</th>
<th>Companies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Extensible column store, very scalable, distributed</td>
<td>Netflix, eBay</td>
</tr>
<tr>
<td>MongoDB</td>
<td>Document-oriented, fast, scalable</td>
<td>craigslist, foursquare</td>
</tr>
<tr>
<td>Redis</td>
<td>Key-value store, very fast</td>
<td>github, flickr, Yahoo</td>
</tr>
</tbody>
</table>

http://nosql-database.org/ lists 122+ NoSQL databases
Redis: Fast and Scalable

In-memory = FAST

Consistent Hashing

Redis master

Redis slave

File system

Durable

Redis slave

File system

Redis master

Redis slave

File system
Redis: advanced key/value store

- **String**: SET, GET
- **List**: LPOP, LPUSH, ...
- **Set**: SADD, SMEMBERS, ...
- **Sorted Set**: ZADD, ZRANGE
- **Hashes**: HSET, HGET
Sorted sets

Members are sorted by score
Adding members to a sorted set

```
zadd myset 5.0 a
```
Adding members to a sorted set

zadd myset 10.0 b
Adding members to a sorted set

```
zadd myset 1.0 c
```

Redis Server

```
myset
  c 1.0
  a 5.0
  b 10.0
```
Retrieving members by index range

```
zrange myset 0 1
```

Redis Server

```
myset
    c  1.0
    a  5.0
    b  10.0
```
Retrieving members by score

```
zrangebyscore myset 1 6
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Min value</th>
<th>Max value</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>10.0</td>
<td></td>
</tr>
</tbody>
</table>
Redis is great but there are tradeoffs

- No rich queries: PK-based access only
- Limited transaction model:
  - Read first and then execute updates as batch
  - Difficult to compose code
- Data must fit in memory
- Single-threaded server: run multiple with client-side sharding
- Missing features such as access control, ...

Hope you don’t need ACID or rich queries one day
The future is polyglot

e.g. Netflix
- RDBMS
- SimpleDB
- Cassandra
- Hadoop/Hbase
Benefits

**RDBMS**
ACID transactions
Rich queries
...

**NoSQL**
Performance
Scalability
...

+
Drawbacks

Complexity

Development

Operational
Polyglot Pattern: write to SQL and NoSQL

NEW TWEET

INSERT

LPUSHX LTRIM

Avoids expensive MySQL joins

MySQL

Tweets

Follows

Followers

Redis: Timeline

Follower 1

Follower 2

Follower 3

...

TWEET
Polyglot Pattern: replicate from MySQL to NoSQL

Reader
Query Service
query()
Redis
Materialized view
replicate and denormalize
MySQL Database
Update Service
update()
Writer
System of Record
@crichardson
Agenda

• Why polyglot persistence?
  
  • Optimizing queries using Redis materialized views

• Synchronizing MySQL and Redis
Finding available restaurants

Available restaurants =
   Serve the zip code of the delivery address
   AND
   Are open at the delivery time

public interface AvailableRestaurantRepository {
   List<AvailableRestaurant> ! findAvailableRestaurants(Address deliveryAddress, Date deliveryTime);
   ...
}
# Database schema

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ajanta</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Montclair Eggshop</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant_id</th>
<th>zipcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94707</td>
</tr>
<tr>
<td>1</td>
<td>94619</td>
</tr>
<tr>
<td>2</td>
<td>94611</td>
</tr>
<tr>
<td>2</td>
<td>94619</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant_id</th>
<th>dayOfWeek</th>
<th>openTime</th>
<th>closeTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monday</td>
<td>1130</td>
<td>1430</td>
</tr>
<tr>
<td>1</td>
<td>Monday</td>
<td>1730</td>
<td>2130</td>
</tr>
<tr>
<td>2</td>
<td>Tuesday</td>
<td>1130</td>
<td>...</td>
</tr>
</tbody>
</table>
Finding available restaurants on Monday, 6.15pm for 94619 zipcode

Straightforward three-way join

```sql
select r.*
from restaurant r
inner join restaurant_time_range tr
  on r.id = tr.restaurant_id
inner join restaurant_zipcode sa
  on r.id = sa.restaurant_id
where '94619' = sa.zip_code
  and tr.day_of_week='monday'
  and tr.openingtime <= 1815
  and 1815 <= tr.closingtime
```
How to scale this query?

• Query caching is usually ineffective
• MySQL Master/slave replication is straightforward BUT
  • Complexity of slaves
  • Risk of replication failing
  • Limitations of MySQL master/slave replication
Query denormalized copy stored in Redis

CONSUMER
Order taking
Redis
findAvailable()

RESTAURANT OWNER
Restaurant Management
MySQL Database
update()

Copy
System of Record

findAvailable()
update()
BUT how to implement findAvailableRestaurants() with Redis?!

```sql
select r.*
from restaurant r
    inner join restaurant_time_range tr
        on r.id = tr.restaurant_id
    inner join restaurant_zipcode sa
        on r.id = sa.restaurant_id
where '94619' = sa.zip_code
    and tr.day_of_week='monday'
    and tr.openingtime <= 1815
    and 1815 <= tr.closingtime
```
**ZRANGEBYSCORE** = Simple SQL Query

```sql
ZRANGEBYSCORE myset 1 6
```

= 

```sql
select value 
from sorted_set 
where key = 'myset' 
  and score >= 1 
  and score <= 6
```

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to transform the SELECT statement?

We need to denormalize

```sql
select r.*
from restaurant r
  inner join restaurant_time_range tr
      on r.id = tr.restaurant_id
  inner join restaurant_zipcode sa
      on r.id = sa.restaurant_id
where '94619' = sa.zip_code
  and tr.day_of_week='monday'
  and tr.openingtime <= 1815
  and 1815 <= tr.closingtime
```
Simplification #1: Denormalization

<table>
<thead>
<tr>
<th>Restaurant_id</th>
<th>Day_of_week</th>
<th>Open_time</th>
<th>Close_time</th>
<th>Zip_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monday</td>
<td>1130</td>
<td>1430</td>
<td>94707</td>
</tr>
<tr>
<td>1</td>
<td>Monday</td>
<td>1130</td>
<td>1430</td>
<td>94619</td>
</tr>
<tr>
<td>1</td>
<td>Monday</td>
<td>1730</td>
<td>2130</td>
<td>94707</td>
</tr>
<tr>
<td>1</td>
<td>Monday</td>
<td>1730</td>
<td>2130</td>
<td>94619</td>
</tr>
<tr>
<td>2</td>
<td>Monday</td>
<td>0700</td>
<td>1430</td>
<td>94619</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SELECT restaurant_id
FROM time_range_zip_code
WHERE day_of_week = 'Monday'
AND zip_code = 94619
AND 1815 < close_time
AND open_time < 1815

Simpler query:
- No joins
- Two = and two <
Simplification #2: Application filtering

```sql
SELECT restaurant_id, open_time
FROM time_range_zip_code
WHERE day_of_week = 'Monday'
    AND zip_code = 94619
    AND 1815 < close_time
    AND open_time < 1815
```

Even simpler query
- No joins
- Two = and one <
Simplification #3: Eliminate multiple =’s with concatenation

<table>
<thead>
<tr>
<th>Restaurant_id</th>
<th>Zip_dow</th>
<th>Open_time</th>
<th>Close_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94707:Monday</td>
<td>1130</td>
<td>1430</td>
</tr>
<tr>
<td>1</td>
<td>94619:Monday</td>
<td>1130</td>
<td>1430</td>
</tr>
<tr>
<td>1</td>
<td>94707:Monday</td>
<td>1730</td>
<td>2130</td>
</tr>
<tr>
<td>1</td>
<td>94619:Monday</td>
<td>1730</td>
<td>2130</td>
</tr>
<tr>
<td>2</td>
<td>94619:Monday</td>
<td>0700</td>
<td>1430</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SELECT restaurant_id, open_time
FROM time_range_zip_code
WHERE zip_code_day_of_week = ‘94619:Monday’
AND 1815 < close_time

key

range
Simplification #4: Eliminate multiple SELECT VALUES with concatenation

SELECT open_time_restaurant_id, close_time
FROM time_range_zip_code
WHERE zip_code_day_of_week = '94619:Monday'
AND 1815 < close_time
Using a Redis sorted set as an index

<table>
<thead>
<tr>
<th>zip_dow</th>
<th>open_time_restaurant_id</th>
<th>close_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>94707:Monday</td>
<td>1130_1</td>
<td>1430</td>
</tr>
<tr>
<td>94619:Monday</td>
<td>1130_1</td>
<td>1430</td>
</tr>
<tr>
<td>94707:Monday</td>
<td>1730_1</td>
<td>2130</td>
</tr>
<tr>
<td>94619:Monday</td>
<td>1730_1</td>
<td>2130</td>
</tr>
<tr>
<td>94619:Monday</td>
<td>0700_2</td>
<td>1430</td>
</tr>
</tbody>
</table>

...
Finding restaurants that have not yet closed

Key
94619:Monday
94707:Monday

Sorted Set [ Entry:Score, …]
[0700_2:1430, 1130_1:1430, 1730_1:2130]
[1130_1:1430, 1730_1:2130]

Delivery zip and day

Delivery time

ZRANGEBYSCORE 94619:Monday 1815 2359

{1730_1}

1730 is before 1815  ➔ Ajanta is open
NoSQL $\Rightarrow$ Denormalized representation for each query
Sorry Ted!

Agenda

• Why polyglot persistence?
• Optimizing queries using Redis materialized views
• Synchronizing MySQL and Redis
MySQL & Redis need to be consistent
Two-Phase commit is not an option

• Redis does not support it

• Even if it did, 2PC is best avoided

http://www.infoq.com/articles/ebay-scalability-best-practices
Atomic Consistent Isolated Durable

Basically Available Soft state Eventually consistent

BASE: An Acid Alternative http://queue.acm.org/detail.cfm?id=1394128
Updating Redis #FAIL

begin MySQL transaction
update MySQL
update Redis
rollback MySQL transaction

Redis has update
MySQL does not

begin MySQL transaction
update MySQL
commit MySQL transaction
<<system crashes>>
update Redis

MySQL has update
Redis does not
Updating Redis reliably
Step 1 of 2

begin MySQL transaction
update MySQL
queue CRUD event in MySQL
commit transaction

Event Id
Operation: Create, Update, Delete
New entity state, e.g. JSON
for each CRUD event in MySQL queue  
    get next CRUD event from MySQL queue  
    If CRUD event is not duplicate then  
        Update Redis (incl. eventId)  
    end if  
    begin MySQL transaction  
        mark CRUD event as processed  
    commit transaction
Generating CRUD events

- Explicit code
- Hibernate event listener
- Service-layer aspect
- CQRS/Event-sourcing
- Database triggers
- Reading the MySQL binlog
Step 1

INSERT INTO ...

ENTITYCrud_EVENT

<table>
<thead>
<tr>
<th>ID</th>
<th>JSON</th>
<th>processed?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2

EntityCrudEvent Processor

SELECT ... FROM ...

Timer

Redis Updater

Redis

@crichardson
Representing restaurants in Redis

- 94619:Monday
  - [0700_2:1430, 1130_1:1430, ...]

- 94619:Tuesday
  - [0700_2:1430, 1130_1:1430, ...]

- restaurant: 1
  - ...JSON...

- restaurant:lastSeenEventId: 1
  - 55
Optimistic locking

WATCH restaurant:lastSeenEventId:<restaurantId>

lastSeenEventId = GET restaurant:lastSeenEventId:<restaurantId>

if (lastSeenEventId >= eventId) return;

MULTI

SET restaurant:lastSeenEventId:<restaurantId> eventId
...
SET restaurant:<restaurantId> restaurantJSON
ZADD ZipCode:<DayOfTheWeek> ...
... update the restaurant data...

EXEC

Duplicate detection

Transaction
Summary

• Each SQL/NoSQL database = set of tradeoffs
• Polyglot persistence: leverage the strengths of SQL and NoSQL databases
• Use Redis as a distributed cache
• Store denormalized data in Redis for fast querying
• Reliable database synchronization required