What You Need To Know To Move From A Relational To A NoSQL Database

Dustin Sallings

Friday, December 16, 11
DATABASE TAXONOMY
JAMES HAMILTON, AMAZON

• Features–First
  • Oracle, SQL Server, DB2, MySQL, PostgreSQL, Amazon RDS

• Scale–First
  • Couchbase Server, CouchDB, Project Voldemort, Riak, Scalaris, Kai, Dynomite, MemcacheDB, ThruDB, Cassandra, HBase and Hypertable

• Simple Structured Storage
  • Amazon SimpleDB, Berkeley DB

• Purpose–Optimized Stores
  • StreamBase, Vertica, Aster Data, Netezza, Greenplum, VoltDB
WHY NOSQL?

"Zynga’s games serve over 235 million active users per month. We depend on technology from Couchbase to make that possible. We have improved the performance and availability of our games while reducing hardware and administration costs. We will continue to transition our data from relational databases to Couchbase technology."

Cadir Lee
Chief Technology Officer,
Zynga
# Interactive software – then and now

<table>
<thead>
<tr>
<th>Users</th>
<th>Circa 1975 “Online Applications”</th>
<th>Circa 2011 “Interactive Web Applications”</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000 “online” users</td>
<td>End Point</td>
<td>2,000 “online” users = Starting Point</td>
</tr>
<tr>
<td>Static user population</td>
<td></td>
<td>Dynamic user population</td>
</tr>
<tr>
<td>Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business process automation</td>
<td></td>
<td>Business process innovation</td>
</tr>
<tr>
<td>Highly structured data records</td>
<td></td>
<td>Structured, semi-structured and unstructured data</td>
</tr>
<tr>
<td>Infrastructure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data networking in its infancy</td>
<td></td>
<td>Universal high-speed data networking</td>
</tr>
<tr>
<td>Centralized computing (Mainframes and minicomputers)</td>
<td></td>
<td>Distributed computing (Network servers and virtual machines)</td>
</tr>
<tr>
<td>Memory scarce and expensive</td>
<td></td>
<td>Memory plentiful and cheap</td>
</tr>
</tbody>
</table>
Web application architecture

- Application Scales Out
  Just add more commodity web servers

- Database Scales Up
  Get a bigger, more complex server
Lacking market solutions, users forced to invent

Common characteristics of these “NoSQL” technologies

- No schema required before inserting data
- No schema change required to change data format
- Auto-sharding without application participation
- Distributed query support
- Data replication across servers and regions

Very few organizations want to (fewer can) build and maintain database technology. Couchbase was founded to create packaged, commercially-supported NoSQL database products.
Couchbase is a “document-oriented” NoSQL database

Simple. Flexible. Adjust to changing data management requirements with ease.

No schema required to insert data (or change data format later). Lightweight, cross-platform document format (JSON).
Couchbase is consistently fast

Decouple application performance (user experience) from sketchy database I/O.

Memcached, the most widely deployed in-memory caching technology on the planet, is built in to Couchbase enabling **consistently** low-latency data reads and writes. We wrote most of memcached.
Couchbase is elastic (scales out for increased capacity)

Grow with linear cost, constant performance and without downtime

Unlike other solutions, expanding (or contracting) a Couchbase cluster is **effortless** and requires **no application downtime**.
CASE STUDY
Tribal Crossing: Relational to NoSQL
Common steps on scaling up database:

- Tune queries (indexing, explain query)
- Denormalization
- Cache data (APC / Memcache)
- Tune MySQL configuration
- Replication (read slaves)

Where do we go from here to prepare for the scale of a successful social game?
Tribal Crossing: Challenges

- **Write-heavy requests**
  - Caching does not help
  - MySQL / InnoDB limitation (Percona)
- **Need to scale drastically over night**
  - My Polls – 100 to 1m users over a weekend
- **Small team, no dedicated sysadmin**
  - Focus on what we do best – making games
- **Keeping cost down**
My SQL with master–to–master replication and sharding

- Complex to setup, high administration cost
- Requires application level changes
- Scaling is invasive and requires much planning
SPEED, SPEED, SPEED

Immediate consistency

Interface is dead simple to use
  - “We are already using Memcache”

Low sysadmin overhead

Schema-less data store

Used and Proven by big guys like Zynga

... and lastly, because Tribal CAN
  - Bigger firms with legacy code base = hard to adapt
  - Small team = ability to get on the cutting edge
Tribal Crossing: Deploying Couchbase in EC2

- Amazon Linux AMI, 64-bit, EBS backed instance
- Set up swap space
- Install Couchbase Server
- Access web console http://<hostname>:8091
- Start the new cluster with a single node
- Add the other nodes to the cluster and rebalance
Moxi figures out which node in the cluster holds data for a given key.

- On each web server, install Moxi
- Start Moxi by pointing it to the DNS entry you created
- Web apps connect to Moxi that is running locally

```python
memcache->addServer('localhost', 11211);
```
Use case – simple farming game:

• A player can have a variety of plants on their farm.

• A player can add or remove plants from their farm.

• A Player can see what plants are on another player's farm.
Representing Objects

- Simply treat an object as an associative array
- Determine the key for an object using the class name (or type) of the object and an unique ID

Representing Object Lists

- Denormalization
- Save a comma separated list or an array of object IDs
Tribal Crossing: Representing Game Data in Couchbase

Player Object

**Key:** 'Player1'

**Array**

```
(  
    [Id] => 1  
    [Name] => Shawn  
)
```

Plant Object

**Key:** 'Plant201'

**Array**

```
(  
    [Id] => 201  
    [Player_Id] => 1  
    [Name] => Starflower  
)
```

PlayerPlant List

**Key:** 'Player1_PlantList'

**Array**

```
(  
    [0] => 201  
    [1] => 202  
    [2] => 204  
)
```
Tribal Crossing: Schema-less Game Data

• No need to “ALTER TABLE”
• Add new “fields” all objects at any time
  - Specify default value for missing fields
  - Increased development speed
• Using JSON for data objects though, owing to the ability to query on arbitrary fields in Couchbase 2.0
Get all plants belonging to a given player

Request: GET /player/1/farm

```php
$plant_ids = couchbase->get('Player1_PlantList');

$response = array();

foreach ($plant_ids as $plant_id) {
    $plant = couchbase->get('Plant' . $plant_id);
    $response[] = $plant;
}

echo json_encode($response);
```
Give a player a new plant

// Create the new plant
$new_plant = array(
    'id' => 100,
    'name' => 'Mushroom'
);

$couchbase->set('Plant100', $new_plant);

// Update the player plant list
$plant_ids = $couchbase->get('Player1_PlantList');
$plant_ids[] = $new_plant['id'];

$couchbase->set('Player1_PlantList', $plant_ids);
Concurrency issue can occur when multiple requests are working with the same piece of data.

Solution:
- CAS (check-and-set)
  - Implement optimistic concurrency control
- Locking (try/wait cycle)
  - GETL (get with lock + timeout) operations
  - Pessimistic concurrency control
• Record object relationships both ways
  - Example: Plots and Plants
    • Plot object stores id of the plant that it hosts
    • Plant object stores id of the plot that it grows on
  - Resolution in case of mismatch

• Don't sweat the extra calls to load data in a one-to-many relationship
  - Use multiGet
  - We can still cache aggregated results in a Memcache bucket if needed
First migrated large or slow performing tables and frequently updated fields from MySQL to Couchbase.
## Tribal Crossing: Deployment

### Monitor Servers

<table>
<thead>
<tr>
<th>Server Node Name</th>
<th>Swap Usage</th>
<th>RAM Usage</th>
<th>CPU Usage</th>
<th>Active Items</th>
<th>Replica Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>rescue.prod.mb1.tbxing.com</td>
<td>0%</td>
<td>91.7%</td>
<td>1.5%</td>
<td>9.11 M</td>
<td>9.35 M</td>
</tr>
<tr>
<td>rescue.prod.mb2.tbxing.com</td>
<td>0%</td>
<td>93.6%</td>
<td>2.31%</td>
<td>9.13 M</td>
<td>8.78 M</td>
</tr>
<tr>
<td>rescue.prod.mb3.tbxing.com</td>
<td>0%</td>
<td>90%</td>
<td>2.25%</td>
<td>9.07 M</td>
<td>8.81 M</td>
</tr>
<tr>
<td>rescue.prod.mb4.tbxing.com</td>
<td>0.476%</td>
<td>99.4%</td>
<td>1.51%</td>
<td>9.09 M</td>
<td>9.08 M</td>
</tr>
<tr>
<td>rescue.prod.mb5.tbxing.com</td>
<td>0.233%</td>
<td>98.5%</td>
<td>1.51%</td>
<td>9.1 M</td>
<td>8.75 M</td>
</tr>
<tr>
<td>rescue.prod.mb6.tbxing.com</td>
<td>0%</td>
<td>90.7%</td>
<td>1.53%</td>
<td>9.1 M</td>
<td>8.74 M</td>
</tr>
<tr>
<td>rescue.prod.mb7.tbxing.com</td>
<td>0%</td>
<td>88.7%</td>
<td>2.82%</td>
<td>9.15 M</td>
<td>8.74 M</td>
</tr>
<tr>
<td>rescue.prod.mb9.tbxing.com</td>
<td>0%</td>
<td>83.1%</td>
<td>1.26%</td>
<td>8.93 M</td>
<td>9.04 M</td>
</tr>
</tbody>
</table>
Tribal Crossing: Deployment
Tribal Crossing: Conclusion

- Significantly reduced the cost incurred by scaling up database servers and managing them.
- Achieved significant improvements in various performance metrics (e.g., read, write, latency, etc.)
- Allowed them to focus more on game development and optimizing key metrics
- Plan to use real-time MapReduce, querying, and indexing abilities provided by the upcoming Elastic Couchbase 2.0
COUCHBASE SERVER 2.0

Now with indexing & queries
Membase Server is now Couchbase Server

Membase Server 1.7 → Couchbase Server 2.0
New architecture influenced by CouchDB technology

CouchDB is the original “NoSQL” document database and the most widely deployed NoSQL database technology, period. It is also the only document database you can trust with your data.
THANK YOU