Scalable NoSQL Architectures

Lessons Learned

Felix Gessert
felix.gessert@baqend.com
May 18, AWS Summit 2017, Berlin
NoSQL Technologies:
• Polyglot Persistence
• Web Caching
• Cloud Data Management

PhD Thesis & Research

About me
Felix Gessert

CEO & Co-founder

Baqend:
High-Performance Backend-as-a-Service

www.baqend.com
INTRODUCTION

What are NoSQL data stores?
Architecture

Typical Data Architecture:

In Practice:

The era of **one-size-fits-all** database systems is over

→ **Many specialized** data systems
How to choose a database system?
Many Potential Candidates

**Question for this talk:**
How to approach the decision problem?
NoSQL Databases

Motivation

- Two main drivers:
  - User-generated data, Request load
  - Impedance Mismatch

Scalability

<table>
<thead>
<tr>
<th>ID</th>
<th>Line Item 1: ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Line Item2: ...</td>
</tr>
<tr>
<td>Payment: Credit Card, ...</td>
<td></td>
</tr>
</tbody>
</table>

Orders

Payment

Customers
NoSQL System Classification

Two common criteria:

- **Data Model**
  - Key-Value
  - Wide-Column
  - Document
  - Graph

- **Consistency/Availability Trade-Off**
  - **AP**: Available & Partition Tolerant
  - **CP**: Consistent & Partition Tolerant
  - **CA**: Not Partition Tolerant
Data Model Classes

Document

ID/Key

JSON Document

Wide Column

Key-Value

Version (timestamped)

Hypertable

Hypertable

Document

Cassandra

MongoDB

CouchDB

Graph

Neo4j

Graph

Row Key

Column

Graph

Couchbase

Redis

RavenDB

Project Voldemort

Aerospike

Data

Model

Classes

com.cnn.www

content: "<html>…"
title: "CNN"
crawled: ...

{order-id: 23,
customer: {name: "Felix Gessert", age: 29},
line-items: [{product-name: "x", ...}, ...]}

{23, 76, 233, 11}

[234, 3466, 86, 55]

Theme -> "dark", cookies -> "false"

{23, 76, 233, 11}

[234, 3466, 86, 55]

An opaque blob

Apple

John Doe

since: 1999

salary: 140K

works_for

name: John Doe

comp any

the graph database

Graph

Properties

Edges

Nodes

WORKS FOR since: 1999

salary: 140K

name: John Doe

comp any

the graph database
CAP-Theorem Classes

Consistency, Availability, Partition Tolerance - pick two:

- Block response until ACK arrives → Consistency (CP)
  Value = V₁
  N₁

- Response before successful replication → Availability (AP)
  Value = V₀
  N₂

Network partition

Eric Brewer, ACM-PODC Keynote, July 2000

BEYOND CAP AND DATA MODELS

What about concrete requirements, e.g. query capabilities?
Functional Requirements from the application enable Central techniques NoSQL databases employ, which in turn enable Operational Requirements.

- Functional Techniques: Scan Queries, ACID Transactions, Conditional or Atomic Writes, Joins, Sorting, Filter Queries, Full-text Search, Aggregation and Analytics, Sharding, Replication, Logging, Update-in-Place, Caching, In-Memory Storage, Append-Only Storage, Storage Management, Query Processing, Elasticity, Consistency, Read Latency, Write Throughput, Read Availability, Write Availability, Durability.

In-depth article on this: blog.baqend.com

2.1 Different Data Models

The most commonly employed distinction between NoSQL databases is the way they store and allow access to data. Each system covered in this paper can be categorised as either key-value store, document store or wide-column store.

2.1.1 Key-Value Stores. A key-value store consists of a set of key-value pairs with unique keys. Due to this simple structure, it only supports get and put operations. As the nature of the stored value is transparent to the database, pure key-value stores do not support operations beyond simple CRUD (Create, Read, Update, Delete). Key-value stores are therefore often referred to as schemaless: Any assumptions about the structure of stored data are implicitly encoded in the application logic (schema-on-read) and not explicitly defined through a data definition language (schema-on-write).

The obvious advantages of this data model lie in its simplicity. The very simple
Scaling Storage and Throughput

Partitioning strategies:
Hash-based, Range-based, Entity Group Sharding
Read Scalability and Fault Tolerance

- **When**: Synchronous, Asynchronous
- **Where**: Multi-Master, Master-Slave, Quorum-based

Storage Management:

**RAM**
- Fast but small
- Random and sequential access

**SSD**
- Limited write cycles
- Bad for random writes

**HDD**
- Large capacities
- Only fast sequentially

- **Data**
  - Improves latency.
  - In-Memory/Caching
  - Is good for read latency.
  - Update-In-Place
  - Append-Only I/O
  - Logging

- **Log**
  - Increases write throughput.
  - For durability of write operations.

- **Persistent Storage**
  - For durability of write operations.
Local Indexing – Partition by document:

WHERE color=blue

Implemented in:
- MongoDB
- Riak
- Cassandra
- DynamoDB
- Elasticsearch
- SolrCloud
- VoltDB
- F1/Cloud Spanner

Scatter-gather query pattern.

Global Indexing – Partition by term:

### Partition I
- **Key**: 12, 56, 77
- **Value**: Consistent maintenance requires distributed transactions

### Partition II
- **Key**: 04
- **Value**: Implemented in
  - DynamoDB
  - Oracle Datawarehouse
  - Riak (Search)
  - Cassandra (Search)
  - Couchbase
  - Megastore/DataStore
  - F1/Cloud Spanner

**Targeted Query**

```
WHERE color=blue
```
How are the techniques from the NoSQL toolbox used in data stores?
Dynamo (AP)
Distributed, Highly Available KV-Store

- Developed at Amazon (2007)
- Techniques: Consistent Hashing & Quorums
- Inspired Riak, Cassandra, Voldemort, DynamoDB

DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."
### Classification: Dynamo-style Systems Techniques

<table>
<thead>
<tr>
<th>Sharding</th>
<th>Range-Sharding</th>
<th>Hash-Sharding</th>
<th>Entity-Group Sharding</th>
<th>Consistent Hashing</th>
<th>Shared Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>Transaction Protocol</td>
<td>Sync. Replication</td>
<td>Async. Replication</td>
<td>Primary Copy</td>
<td>Update Anywhere</td>
</tr>
<tr>
<td>Storage Management</td>
<td>Logging</td>
<td>Update-in-Place</td>
<td>Caching</td>
<td>In-Memory</td>
<td>Append-Only Storage</td>
</tr>
<tr>
<td>Query Processing</td>
<td>Global Index</td>
<td>Local Index</td>
<td>Query Planning</td>
<td>Analytics</td>
<td>Materialized Views</td>
</tr>
</tbody>
</table>
Redis (CA)

In-Memory Key-Value Store

- **Remote Dictionary Server**
- **Data model**: rich data structures (list, sets, ...)
- Hash-Sharding & Asynchronous Replication

Redis Server

- **Event Loop**
- **One Thread**
- **Local Filesystem**
  - **Log**
    - AOF
  - **Dump**
    - RDB
    - Periodic
    - After X Writes
    - **SAVE**

Client

```text
SET mykey hello
+OK
```
## Classification: Redis

### Techniques

<table>
<thead>
<tr>
<th>Category</th>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sharding</strong></td>
<td>Range-Sharding</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Hash-Sharding</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Entity-Group Sharding</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Consistent Hashing</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Shared Disk</td>
<td>-</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>Transaction Protocol</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Sync. Replication</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Async. Replication</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Primary Copy</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Update Anywhere</td>
<td>-</td>
</tr>
<tr>
<td><strong>Storage Management</strong></td>
<td>Logging</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Update-in-Place</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Caching</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>In-Memory</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Append-Only Storage</td>
<td>-</td>
</tr>
<tr>
<td><strong>Query Processing</strong></td>
<td>Global Index</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Local Index</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Query Planning</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Analytics</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Materialized Views</td>
<td>-</td>
</tr>
</tbody>
</table>
Apache HBase (CP)
Consistent Wide-Column Store

- Open-Source Implementation of BigTable
- Range-Sharded & Synchronous Replication
- Optimized for Write Throughput
## Classification: HBase

### Techniques

<table>
<thead>
<tr>
<th>Sharding</th>
<th>Range-Sharding</th>
<th>Hash-Sharding</th>
<th>Entity-Group Sharding</th>
<th>Consistent Hashing</th>
<th>Shared Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>Transaction Protocol</td>
<td>Sync. Replication</td>
<td>Async. Replication</td>
<td>Primary Copy</td>
<td>Update Anywhere</td>
</tr>
<tr>
<td>Storage Management</td>
<td>Logging</td>
<td>Update-in-Place</td>
<td>Caching</td>
<td>In-Memory</td>
<td>Append-Only Storage</td>
</tr>
<tr>
<td>Query Processing</td>
<td>Global Index</td>
<td>Local Index</td>
<td>Query Planning</td>
<td>Analytics</td>
<td>Materialized Views</td>
</tr>
</tbody>
</table>
Apache Cassandra (AP)  
Available Wide-Column Store

- Published 2007 by Facebook
- **BigTable data model** with **Dynamo sharding**
- SQL-like (CQL), but **limited complexity**

**SELECT * FROM playlists**
WHERE id = 23423

<table>
<thead>
<tr>
<th>id</th>
<th>order</th>
<th>id</th>
<th>artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>23423</td>
<td>1</td>
<td>64563</td>
<td>Elvis</td>
</tr>
<tr>
<td>23423</td>
<td>2</td>
<td>f9291</td>
<td>Elvis</td>
</tr>
</tbody>
</table>
# Classification: Cassandra Techniques

<table>
<thead>
<tr>
<th>Classification</th>
<th>Range-Sharding</th>
<th>Hash-Sharding</th>
<th>Entity-Group Sharding</th>
<th>Consistent Hashing</th>
<th>Shared Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sharding</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>Transaction Protocol</td>
<td>Sync. Replication</td>
<td>Async. Replication</td>
<td>Primary Copy</td>
<td>Update Anywhere</td>
</tr>
<tr>
<td><strong>Storage Management</strong></td>
<td>Logging</td>
<td>Update-in-Place</td>
<td>Caching</td>
<td>In-Memory</td>
<td>Append-Only Storage</td>
</tr>
<tr>
<td><strong>Query Processing</strong></td>
<td>Global Index</td>
<td>Local Index</td>
<td>Query Planning</td>
<td>Analytics</td>
<td>Materialized Views</td>
</tr>
</tbody>
</table>
MongoDB (CP)
Document Store

- Allows **complex queries** and indexing
- Sharding (range- or hash-based)
- Replication (synchronous or asynchronous)
- **Pluggable** Storage Engine

**MongoDB**
- Model: Document
- License: GNU AGPL 3.0
- Written in: C++

*Replica Set*
- **Master**
- Slave
- **Slave**

*Replica Set*
- **Master**
- Slave
- **Slave**

*config*

*mongos*
## Classification: MongoDB Techniques

<table>
<thead>
<tr>
<th>Sharding</th>
<th>Range-Sharding</th>
<th>Hash-Sharding</th>
<th>Entity-Group Sharding</th>
<th>Consistent Hashing</th>
<th>Shared Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication</td>
<td>Transaction Protocol</td>
<td>Sync. Replication</td>
<td>Async. Replication</td>
<td>Primary Copy</td>
<td>Update Anywhere</td>
</tr>
<tr>
<td>Storage Management</td>
<td>Logging</td>
<td>Update-in-Place</td>
<td>Caching</td>
<td>In-Memory</td>
<td>Append-Only Storage</td>
</tr>
<tr>
<td>Query Processing</td>
<td>Global Index</td>
<td>Local Index</td>
<td>Query Planning</td>
<td>Analytics</td>
<td>Materialized Views</td>
</tr>
</tbody>
</table>
How can the choices for an appropriate system be narrowed down?
NoSQL Decision Tree

Access

Fast Lookups
- Volume
  - RAM
  - CAP

Complex Queries
- Volume
  - HDD-Size
    - Consistency
  - Unbounded
    - Query Pattern

Purpose

Architects
Find system candidates

DB vendors
Communicate Trade-Offs

Example Applications
- Cassandra
- Riak
- Voldemort
- Aerospike
- Shopping Basket
- HBase
- MongoDB
- CouchBase
- DynamoDB
- Cassandra
- HBase
- Riak
- MongoDB
- Redis
- Memcached
- Voldemort
- HDBMS Neo4J
- RavenDB
- MarkLogic
- RDBMS
- CouchDB
- MongoDB
- SimpleDB
- Hadoop, Spark
- Parallel DWH
- Cassandra, HBase
- Riak, MongoDB
- Cache
- Order History
- OLTP
- Website
- Social Network
- Big Data
- OLTP
- Website
- Social Network
- Big Data
IN PRACTICE

How are NoSQL databases used in scalable architectures?
Scalable NoSQL Architectures
The 3-Tier Stack

- >100 individual requests
- ~100ms latency per request
- Blocks delivery of website
- Render
- HTML, CSS, JS, ...
- User Action
- Business logic in the server
- Server
- NoSQL DB
- Needs to be scalable
- Data
- Update
- Business logic in the server
- NoSQL DB
- Needs to be scalable
Scalable NoSQL Architectures
The 2-Tier Stack

Challenges
- User Management
- Access Control
- Server-side Business Logic
- **Latency** and **scalability** still an issue

Single Page Applications
- Client-side rendering
- Content dynamically loaded
- Implements business logic

Cloud Databases
- Serving data and files
- Inserting and updating data
- Scaling with reads and writes

Tools
- Angular
- React
- Vue.js
- DynamoDB
- Elasticache
- MongoDB
The Serverless Paradigm
FaaS, BaaS and everything in between
Presentation is loading
Why latency matters

Loading…

Average: 9.3s

- 1000ms

- 4% Traffic

- 4% Visitors

- 1% Revenue
If perceived speed is such an important factor

...what causes slow page load times?
The Problem
Three Bottlenecks: Latency, Backend & Frontend
Network Latency: Impact

Page Load Time as bandwidth increases

Page Load Time as latency decreases

Network Latency: Impact

- 2× Bandwidth = Same Load Time
- ½ Latency ≈ ½ Load Time

Solution: Global Caching

Fresh Data from Ubiquitous Web Caches

Low Latency

Less Processing
New Caching Algorithms
Solve Consistency Problem
New Caching Algorithms Solve Consistency Problem

7 Years Research & Development
Advantage: 15x Faster

What impact does Baqend have in practice?

TRY THIS
benchmark.baqend.com
Baqend Architecture
AWS Infrastructure

REST and HTTP Caching

Backend-as-a-Service

Polyglot Storage in NoSQL DBs

Middleware

Desktop
Mobile
Tablet

Internet
Content-Delivery-Network

Caches
Baqend

redis
mongoDB
elasticsearch
Baqend Architektur
AWS Infrastructure

Baqend Cloud on
fastly

CDN on

Baqend Cloud on
amazon web services
Example Use Case
Heavy Traffic Peaks

Expected:
- 3.5M viewers
- 300,000 visitors in 30 minutes
- 20,000 requests per second
- 4 weeks for development & tests

< 1 second
Page Load Time

7.8%
Conversion Rate

3%
Server Usage

Concurrent
Users
Both queries and objects from MongoDB cached.

Peak >20,000 RPS

High throughput for Bloomfilter → Redis & Microcaching

>3.2 Gigabit/s
The World's Fastest Backend

Build websites and apps that load instantly.

Sky-rocket your Development

Start building now. Baqend Cloud is free and easy to get started with.
Summary

- Categories **CAP** and **data model** too coarse
- **NoSQL Toolbox:**
  - Techniques: Sharding, Replication, ...
  - Promote
  - Functional Requirements
  - Non-functional Requirements

- Cloud-DBs for **single-page apps**: Backend-as-a-Service
- **In-depth**: blog.baqend.com

fg@baqend.com  @baqendcom