COUCHBASE

Shruti Arora
Pulkit Dhir
Shreya Goel
Varun Maheshwari
What is Couchbase?

- Distributed NoSQL document-oriented database.
- Specialized to provide low-latency data management for large-scale applications.
- Supports both key-value and document-oriented use cases.
- Obtained as packaged software in both enterprise and community editions.
Why NoSQL?

❖ Industry after industry is shifting to the Digital Economy.

❖ At the heart of every Digital Economy business are its web, mobile, and Internet of Things (IoT) applications.

❖ Today’s web, mobile, and IoT applications share the following characteristics -
  - Support large numbers of concurrent
  - Deliver highly responsive experiences
  - Be always available
  - Handle semi- and unstructured data
  - Rapidly adapt to changing requirements

❖ The new enterprise technology architecture needs to be far more agile, and requires an approach to real time data management.
Why relational databases fall short?

- Relational databases were born in the era of mainframes and business applications.
- These databases were engineered to run on a single server.
- Traditional databases don’t address the need to develop with agility and to operate at any scale.
- NoSQL databases emerged as a result of the exponential growth of the Internet and the rise of web applications.
Five Trends Create New Technical Challenges that NoSQL Addresses

- More customers are going online
- The Internet is connecting everything
- Big Data is getting bigger
- Applications are moving to the cloud
- The world has gone mobile
Power of SQL

Flexibility of JSON

Scalability of NoSQL
Features

Develop with Agility
- Easier, Faster Development
- Flexible Data Modeling
- Powerful Querying & Indexing
- Big Data Integration

Operate at Any Scale
- Elastic Scalability
- Always-on Availability
- Consistent High Performance
Document Databases

❖ Each record in the database is a self-describing document
❖ Each document has an independent structure
❖ Documents can be complex
❖ All databases require a unique key
❖ Documents are stored using JSON or XML or their derivatives
❖ Content can be indexed and queried
❖ Offer auto-sharding for scaling and replication for high-availability
Figure 1. Relational model for flight schedules

- **Airline**
  - Carrier code
  - Airline Name
  - Description
  - Address

- **Flight**
  - Flight Number
  - Carrier code
  - Aircraft
  - From
  - To

- **Schedule**
  - Flight Number
  - Day
  - Departure Time
  - Arrival Time

Figure 2. Document data model for flight bookings

- **Route**
  - From
  - To

- **Schedule [ ]**
  - Flight Number
  - Airline
  - Day
  - Departure Time
  - Arrival Time
Relational Data Model vs Document Data Model

**Relational data model**

Highly-structured table organization with rigidly-defined data formats and record structure.

You must define a schema before adding records to a database.

Within a table, you need to define constraints in terms of rows and named columns as well as the type of data that can be stored in each column.

**Document data model**

Collection of complex documents with arbitrary, nested data formats and varying “record” format.

A document-oriented database contains documents, which are records that describe the "schema" of the data in the document, as well as the actual data.

You can also use one or more documents to represent a real-world object.
**Figure 1:** RDBMS – An explicit schema prevents the addition of new attributes on demand

**Figure 2:** JSON – The data model evolves as new attributes are added on demand.
**Figure 4:** RMDBS – Queries return duplicate data, applications have to filter it out.

**Figure 3:** RDBMS – Applications “shred” objects into rows of data stored in multiple tables.
Figure 5: JSON – Applications can store objects with nested data as single documents.
## Example: User Profile

To get information about specific user, you perform a join across two tables.
### Document Example: User Profile

All data in a single document

**User Info**

<table>
<thead>
<tr>
<th>KEY</th>
<th>First</th>
<th>Last</th>
<th>ZIP_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dipti</td>
<td>Borkar</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Joe</td>
<td>Smith</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Baxter</td>
<td>Dodson</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Lari</td>
<td>Gorin</td>
<td>3</td>
</tr>
</tbody>
</table>

**Geo Info**

<table>
<thead>
<tr>
<th>ZIP_Id</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEN</td>
<td>CO</td>
<td>30303</td>
</tr>
<tr>
<td>2</td>
<td>MV</td>
<td>CA</td>
<td>94040</td>
</tr>
<tr>
<td>3</td>
<td>CHI</td>
<td>IL</td>
<td>60609</td>
</tr>
<tr>
<td>4</td>
<td>NY</td>
<td>NY</td>
<td>10010</td>
</tr>
</tbody>
</table>

```json
{
  "ID": 1,
  "FIRST": "Dipti",
  "LAST": "Borkar",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA"
}
```
### Making a Change Using RDBMS

#### User Table

<table>
<thead>
<tr>
<th>User ID</th>
<th>First</th>
<th>Last</th>
<th>Zip</th>
<th>Country ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dipti</td>
<td>Borkar</td>
<td>94040</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>Joe</td>
<td>Smith</td>
<td>94040</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td>Ali</td>
<td>Dodson</td>
<td>94040</td>
<td>001</td>
</tr>
<tr>
<td>4</td>
<td>Sarah</td>
<td>Gorin</td>
<td>NW1</td>
<td>002</td>
</tr>
<tr>
<td>5</td>
<td>Bob</td>
<td>Young</td>
<td>30303</td>
<td>003</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>Baker</td>
<td>10010</td>
<td>004</td>
</tr>
<tr>
<td>7</td>
<td>Ray</td>
<td>Jones</td>
<td>31311</td>
<td>005</td>
</tr>
<tr>
<td>8</td>
<td>Lee</td>
<td>Chen</td>
<td>V5V3M</td>
<td>006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td>Doug</td>
<td>Moore</td>
<td>04252</td>
<td></td>
</tr>
<tr>
<td>50001</td>
<td>Mary</td>
<td>White</td>
<td>SW195</td>
<td></td>
</tr>
<tr>
<td>50002</td>
<td>Lisa</td>
<td>Clark</td>
<td>12425</td>
<td></td>
</tr>
</tbody>
</table>

#### Photo Table

<table>
<thead>
<tr>
<th>User ID</th>
<th>Photo ID</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>d043</td>
<td>NYC</td>
</tr>
<tr>
<td>2</td>
<td>b054</td>
<td>Bday</td>
</tr>
<tr>
<td>5</td>
<td>c036</td>
<td>Miami</td>
</tr>
<tr>
<td>7</td>
<td>d072</td>
<td>Sunset</td>
</tr>
<tr>
<td>5002</td>
<td>e086</td>
<td>Spain</td>
</tr>
</tbody>
</table>

#### Country Table

<table>
<thead>
<tr>
<th>Country ID</th>
<th>Country name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>USA</td>
</tr>
<tr>
<td>002</td>
<td>UK</td>
</tr>
<tr>
<td>003</td>
<td>Argentina</td>
</tr>
<tr>
<td>004</td>
<td>Australia</td>
</tr>
<tr>
<td>005</td>
<td>Aruba</td>
</tr>
<tr>
<td>006</td>
<td>Austria</td>
</tr>
<tr>
<td>007</td>
<td>Brazil</td>
</tr>
<tr>
<td>008</td>
<td>Canada</td>
</tr>
<tr>
<td>009</td>
<td>Chile</td>
</tr>
<tr>
<td>130</td>
<td>Portugal</td>
</tr>
<tr>
<td>131</td>
<td>Romania</td>
</tr>
<tr>
<td>132</td>
<td>Russia</td>
</tr>
<tr>
<td>133</td>
<td>Spain</td>
</tr>
<tr>
<td>134</td>
<td>Sweden</td>
</tr>
</tbody>
</table>

#### Status Table

<table>
<thead>
<tr>
<th>User ID</th>
<th>Status ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a42</td>
<td>At conf</td>
</tr>
<tr>
<td>4</td>
<td>b26</td>
<td>excited</td>
</tr>
<tr>
<td>5</td>
<td>c32</td>
<td>hockey</td>
</tr>
<tr>
<td>12</td>
<td>d83</td>
<td>Go A’s</td>
</tr>
<tr>
<td>50000</td>
<td>e34</td>
<td>sailing</td>
</tr>
</tbody>
</table>

#### Affiliations Table

<table>
<thead>
<tr>
<th>User ID</th>
<th>Aff ID</th>
<th>Aff Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a42</td>
<td>Cal</td>
</tr>
<tr>
<td>4</td>
<td>b96</td>
<td>USC</td>
</tr>
<tr>
<td>7</td>
<td>c14</td>
<td>UW</td>
</tr>
<tr>
<td>8</td>
<td>e22</td>
<td>Oxford</td>
</tr>
</tbody>
</table>

### Country Table

<table>
<thead>
<tr>
<th>Country ID</th>
<th>Country name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>USA</td>
</tr>
<tr>
<td>002</td>
<td>UK</td>
</tr>
<tr>
<td>003</td>
<td>Argentina</td>
</tr>
<tr>
<td>004</td>
<td>Australia</td>
</tr>
<tr>
<td>005</td>
<td>Aruba</td>
</tr>
<tr>
<td>006</td>
<td>Austria</td>
</tr>
<tr>
<td>007</td>
<td>Brazil</td>
</tr>
<tr>
<td>008</td>
<td>Canada</td>
</tr>
<tr>
<td>009</td>
<td>Chile</td>
</tr>
<tr>
<td>130</td>
<td>Portugal</td>
</tr>
<tr>
<td>131</td>
<td>Romania</td>
</tr>
<tr>
<td>132</td>
<td>Russia</td>
</tr>
<tr>
<td>133</td>
<td>Spain</td>
</tr>
<tr>
<td>134</td>
<td>Sweden</td>
</tr>
</tbody>
</table>
Making the Same Change with a Document Database

```
{
  "ID": 1,
  "FIRST": "Don",
  "LAST": "Pinto",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA",
  "STATUS": {
    "TEXT": "At Conf"
  },
  "GEO_LOC": "134",
  "COUNTRY": "USA"
}
```
Logical Data Modeling

The logical data modeling phase focuses on describing your entities and relationships. It is done independently of the requirements and facilities of the underlying database platform.

The key definitions you need from your logical data modeling exercise:

❖ Entity keys
❖ Entity attributes
❖ Entity relationships
Physical Data Modeling

The physical data model takes the logical data model and maps the entities and relationships to physical containers.

<table>
<thead>
<tr>
<th>Couchbase Server</th>
<th>Relational databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buckets</td>
<td>Databases</td>
</tr>
<tr>
<td>Buckets or Items (with type designator attribute)</td>
<td>Tables</td>
</tr>
<tr>
<td>Items (key-value or document)</td>
<td>Rows</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
</tbody>
</table>
Items
Items consist of a key and a value. A key is a unique identifier within the bucket. Value can be a binary or a JSON document. You can mix binary and JSON values inside a bucket.

Keys
Each value (binary or JSON) is identified by a unique key. Keys are immutable. Thus, if you use composite or compound keys, ensure that you use attributes that don’t change over time.

Values
- **Binary values**: Binary values can be used for high performance access to compact data through keys.
- **JSON values**: JSON provides rich representation for entities. Couchbase Server can parse, index and query JSON values. JSON provide a name and a value for each attribute.

Buckets
Couchbase Server also provides a container called a bucket to group items. Buckets are primarily used to control resource allocation and to define security and storage properties.
JavaScript Object Notation (JSON)

It is a lightweight data-interchange format which is easy to read and change. JSON is language-independent although it uses similar constructs to JavaScript.

JSON supports the following basic data types:
- Numbers, including integer and floating point
- Strings, including all unicode characters and backslash escape characters
- Boolean: true or false
- Arrays, enclosed in square brackets: ["one", "two", "three"]
- Objects, consisting of key-value pairs, and also known as an associative array or hash. The key must be a string and the value can be any supported JSON data type.
Data access

The ways to access the data:

1. Key value access pattern
2. Querying data
   - MapReduce
   - N1QL
   - Full text search (FTS)
N1QL

❖ A declarative query language that extends SQL for JSON.

❖ N1QL enables you to query JSON documents without any limitations - sort, filter, transform, group, and combine data with a single query.

❖ No longer limited to “single table” and “table per query” data models.
N1QL vs SQL

SQL STATEMENT

1. SELECT name, author
2. FROM books

N1QL STATEMENT

1. SELECT name, author
2. FROM books

SQL RESULTS (ROWS)

<table>
<thead>
<tr>
<th>name</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ender's Game</td>
<td>Orson Scott Card</td>
</tr>
<tr>
<td>Foundation</td>
<td>Isaac Asimov</td>
</tr>
<tr>
<td>Neuromancer</td>
<td>William Gibson</td>
</tr>
<tr>
<td>Consider Phlebas</td>
<td>Iain M. Banks</td>
</tr>
<tr>
<td>Revelation Space</td>
<td>Alastair Reynolds</td>
</tr>
</tbody>
</table>

N1QL RESULTS (DOCUMENT)

```json
{
  "results": [
    {"name": "Ender's Game", "author": "Orson Scott Card"},
    {"name": "Foundation", "author": "Isaac Asimov"},
    {"name": "Neuromancer", "author": "William Gibson"},
    {"name": "Consider Phlebas", "author": "Iain M. Bank"},
    {"name": "Revelation Space", "author": "Alastair Reynolds"}
  ]
}```
Indexing

❖ An index is a data-structure that provides quick and efficient means to query and access data, that would otherwise require scanning a lot more documents.

❖ Couchbase Server speeds up data access with indexes.

❖ Couchbase provides both local and global indexes.
Types of indexes

1. Composite Indexes
2. Covering Indexes
3. Filtered Indexes
4. Function-based Indexes
5. Sub-document Indexes
6. Incremental mapreduce views
7. Spatial Views
8. Full-text Indexes
Data management

- Atomicity properties
- Strong consistency and durability
- Consistency of indexes and Replicas
- Tunable durability Requirements
- Concurrency
- Document Expiration
Distributed data management

❖ Couchbase Server is a distributed system that is built from the ground up for easy scale out and management.

❖ Couchbase Server has a peer-to-peer topology and all the nodes are equal and communicate to each other on demand.
Multidimensional scaling

- MDS enables users to turn on or off specific services on each Couchbase Server node so that the node in effect becomes specialized to handle a specific workload.

- Advantages:
  - Independent Services
  - Quick and efficient
  - Customize machines
  - Workload isolation
BUCKETS vs vbuckets

DOCUMENTS

USER/APPLICATION DATA

APP SIDE

READ FROM / WRITTEN TO

DOCUMENTS

BUCKETS

LOGICAL KEY SPACES

SERVER SIDE

BUCKETS

DISTRIBUTED ACROSS THE

CLUSTER

DYNAMALLY SCALABLE
Data Change protocol

- Data Change Protocol (DCP) is a high-performance streaming protocol that communicates the state of the data using an ordered change log with sequence numbers.

Replication

- creates copies of active data, distributes those replicas across the nodes in the cluster, ensuring that every copy is located on a separate node, and then continues to maintain the replicas over time.
Couchbase Server delivers key high availability features such as zero downtime administration and maintenance, built-in data redundancy, and automatic failover.

Factors that increase system uptime and availability include:

- Number of replicas
- Number of racks or availability zones
- Number of clusters
Cross Datacenter Replication (XDCR)

It is used to replicate data between clusters in different data centers and geographic regions, and can also be used to sync a second Couchbase Server cluster within the same data center.

XDCR serves an important role in high availability / disaster recovery, performance, and load distribution.

- For disaster recovery, one or more clusters can act as hot standbys, enabling cluster-level failover by taking over load as soon as a cluster stops responding.
- In case of serious failures, XDCR can also be used to recover data from a remote cluster. The result is similar to recovery using a backup but often faster.
- In geographically distributed data centers, XDCR can improve performance by placing data close to end users.
Couchbase offers security mechanisms that help protect against threats and breaches.

1. Authentication and Authorization
2. Encryption
3. Auditing
Security

- **Outside network**
  - Web browsers and mobile applications
  - End users

- **Perimeter network**
  - External Firewall
    - Load balancer
    - Only allow web server ingress and outgress ports through external firewall
  - Web server
    - IT administrators
    - Application developers

- **Internal network**
  - Internal Firewall
  - Couchbase cluster
    - IT administrators
    - Database administrators
The core architecture is designed to simplify building modern applications with a flexible data model and simpler high availability, high scalability, high performance, and advanced security.

The applications connect to a Couchbase Server cluster to perform read and write operations, and run queries with low latencies (sub millisecond) and high throughput (millions of operations per second).
## Terminology

The key terms and concepts used in the Couchbase Server architecture:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Node</td>
</tr>
<tr>
<td>2.</td>
<td>Cluster</td>
</tr>
<tr>
<td>3.</td>
<td>Bucket</td>
</tr>
<tr>
<td>4.</td>
<td>Item</td>
</tr>
<tr>
<td>5.</td>
<td>vBucket</td>
</tr>
<tr>
<td>6.</td>
<td>Cluster map</td>
</tr>
<tr>
<td>7.</td>
<td>vBucket map</td>
</tr>
<tr>
<td>8.</td>
<td>Replication</td>
</tr>
<tr>
<td>9.</td>
<td>Rebalance</td>
</tr>
<tr>
<td>10.</td>
<td>Failover</td>
</tr>
<tr>
<td></td>
<td>a. Graceful Failover</td>
</tr>
<tr>
<td></td>
<td>b. Hard Failover</td>
</tr>
<tr>
<td></td>
<td>c. Automatic Failover</td>
</tr>
<tr>
<td>11.</td>
<td>Node Lifecycle</td>
</tr>
</tbody>
</table>
Storage Operations

Application Server

set/add/replace

Couchbase Server

RAM Cache

EP Engine

Disk Write Queue

Replication Queue

Replica Couchbase Cluster Machine
Retrieval Operations
Horizontal Scale-Rebalance

Application Servers

MAP

MAP

MAP

MAP

256 Partitions
3 IO Workers
8 GB RAM

256 Partitions
3 IO Workers
8 GB RAM

1024 Partitions
6 IO Workers
16 GB RAM

TOTAL
Intra-cluster replication

Diagram showing a server cluster with three Couchbase servers. Each server contains replicas of shards for improved data availability and reliability.
Cross data center replication
Storage architecture

- Couchbase uses multiple storage engines:
  - Data Service,
  - MapReduce Views,
  - Spatial Views,
  - Couchstore
  - Index Service,
  - Search Service, and
  - ForestDB
Caching layer

- Data service uses a managed cache
- Index and Search services manage the cache
- Query service manages memory to calculate query responses
Ram quotas

❖ RAM quota allocation is governed through individual services. Each service in Couchbase Server tunes its caching based on its needs.

❖ Services that use RAM quotas:
  ❖ Data service
  ❖ Index and search service
  ❖ Query service
Querying data & query data service

- Retrieving data with document key
- Querying data using View queries
- Querying data using Spatial queries
- Querying data using N1QL queries
Use Cases

Real-Time Big Data
Leverage streaming integration with Hadoop and Storm to support and enable real-time analytics.

Mobile Applications
Build mobile apps with offline support via an embedded database and automatic synchronization.

Digital Communication
Support real-time interaction and communication with low latency read/write access to messages.

Customer 360° View
Aggregate customer information from multiple sources with different data models.
LinkedIn Monitors Massive Data with Couchbase. Couchbase Server provides the scalability and performance the site engineering team needs to power its metric visualization engine.

General Electric set out to bring together device connectivity, data integration and management, data analytics, cloud, and mobility all in a way that works seamlessly together and intuitively for all the members of its business.

Marriott decided it was time to replace its legacy infrastructure to better compete in the Digital Economy. The company evaluated several NoSQL solutions before deciding to switch to Couchbase.
Couchbase VS MongoDB

❖ Concurrency - Couchbase Server was able to handle over 3x as many concurrent clients as MongoDB.

❖ Throughput - Couchbase Server was able to provide 2.5x the throughput of MongoDB.

❖ Latency - Couchbase Server was able to provide 4-5x lower latency than MongoDB.

❖ Price / Performance Ratio - The cost per operation for Couchbase Server would be 22-40% of that for MongoDB.
Couchbase VS Cassandra

Couchbase Server Outperforms Cassandra by 6X on Google Cloud Platform

Results of the Google Cloud benchmark are summarized in the following table:

<table>
<thead>
<tr>
<th></th>
<th>COUCHBASE</th>
<th>CASSANDRA</th>
<th>COUCHBASE ADVANTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>1.1 million writes/sec</td>
<td>1 million writes/sec</td>
<td>-</td>
</tr>
<tr>
<td>Latency</td>
<td>27ms</td>
<td>23ms</td>
<td>-</td>
</tr>
<tr>
<td>Nodes</td>
<td>50 @ 16 cores</td>
<td>300 @ 8 cores</td>
<td>6x</td>
</tr>
<tr>
<td>Total Cores</td>
<td>800</td>
<td>2400</td>
<td>3x</td>
</tr>
<tr>
<td>Price/Hour</td>
<td>$56</td>
<td>$330</td>
<td>6x</td>
</tr>
<tr>
<td>Price per 10K Transactions</td>
<td>$0.51 / 10K ops</td>
<td>$3.30 / 10K ops</td>
<td>6x</td>
</tr>
</tbody>
</table>
Thank you