

NoSQL Alternatives



Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. Deep Dive - MongoDB Architecture

Usage Guidelines



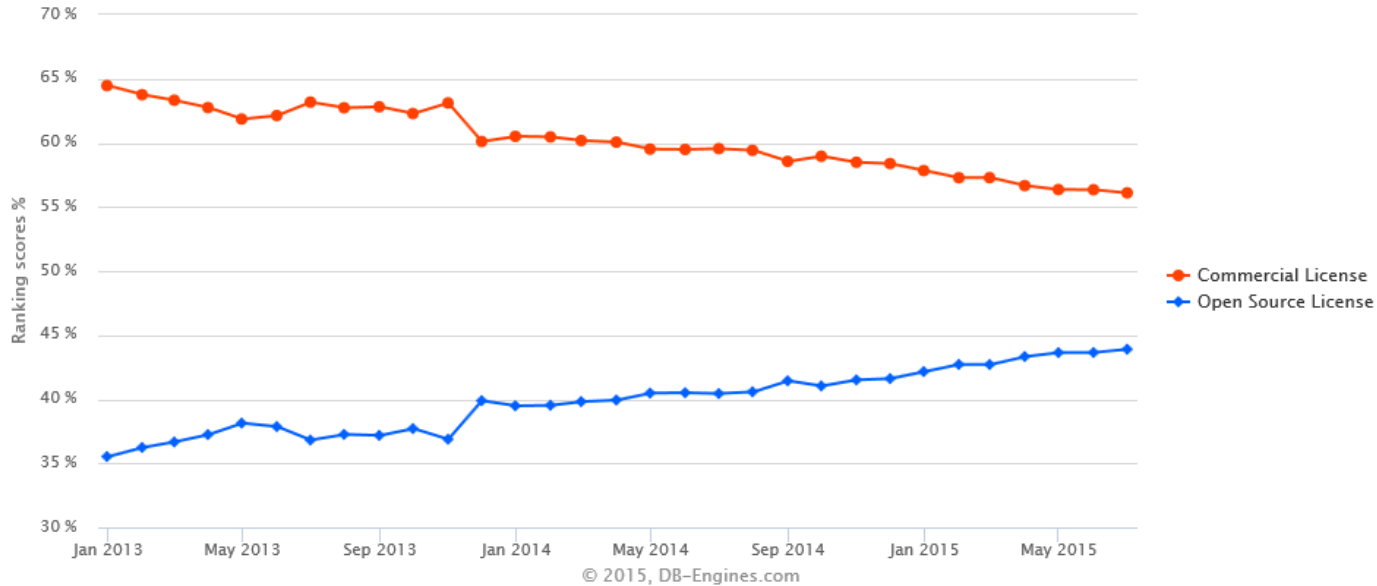
Do not forward this document to any non-Infosys mail ID. Forwarding this document to a non-Infosys mail ID may lead to disciplinary action against you, including termination of employment.

Contents of this material cannot be used in any other internal or external document without explicit permission from ArchHigh@Infosys.com

Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. Deep Dive - MongoDB Architecture

The rise of Open Source Database



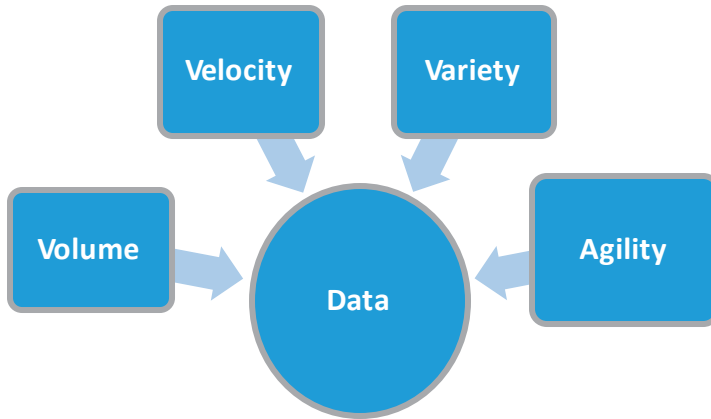
Changing Trends (275+ DBMS to choose)

Rank	DBMS	Score	Changes
1.	Oracle	1456.72	-9.64
2.	MySQL	1283.33	+ 4.98
3.	Microsoft SQL Server	1103.05	-15.00
4.	MongoDB	287.40	+ 8.34
5.	PostgreSQL	272.83	-8.08
6.	DB2	198.12	-0.58
7.	Microsoft Access	144.30	-2.19
8.	Cassandra	112.71	+ 3.80
9.	SQLite	105.87	-2.10
10.	Redis	95.08	-0.41
Copyright © July 2015 DB-Engines.com			

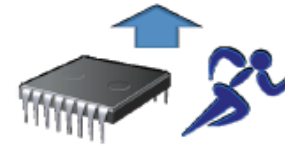
And there are more on the rise – CouchBase, Neo4J, DynamoDB

Why NoSQL?

The Internet era needs



The Architecture shift



Scale Up



Scale Out

RDBMS – Advantages and Limitations

Advantages

- ✓ Proven & Standardized (SQL)
- ✓ Transaction control and concurrency
- ✓ Access control and security
- ✓ Structured Data (Table, Joins, Indexing)

Limitations

- Cluster Management – more complex
- Impedance mismatch
- Adaptability to newer workload / data
- High cost

NoSQL Databases – Key Attributes

Non
Relational

Flexible
schema

Support for
clusters

Scalability

Availability

Open
Source

Cost
effective

NoSQL Alternatives

Key – Value



Memcached



Voldemort



Redis



Riak

Column based



Cassandra



HBase



Hypertable



Accumulo

Document



CouchDB



MongoDB



OrientDB



RavenDB

Graph



HyperGraphDB



Infinite Graph

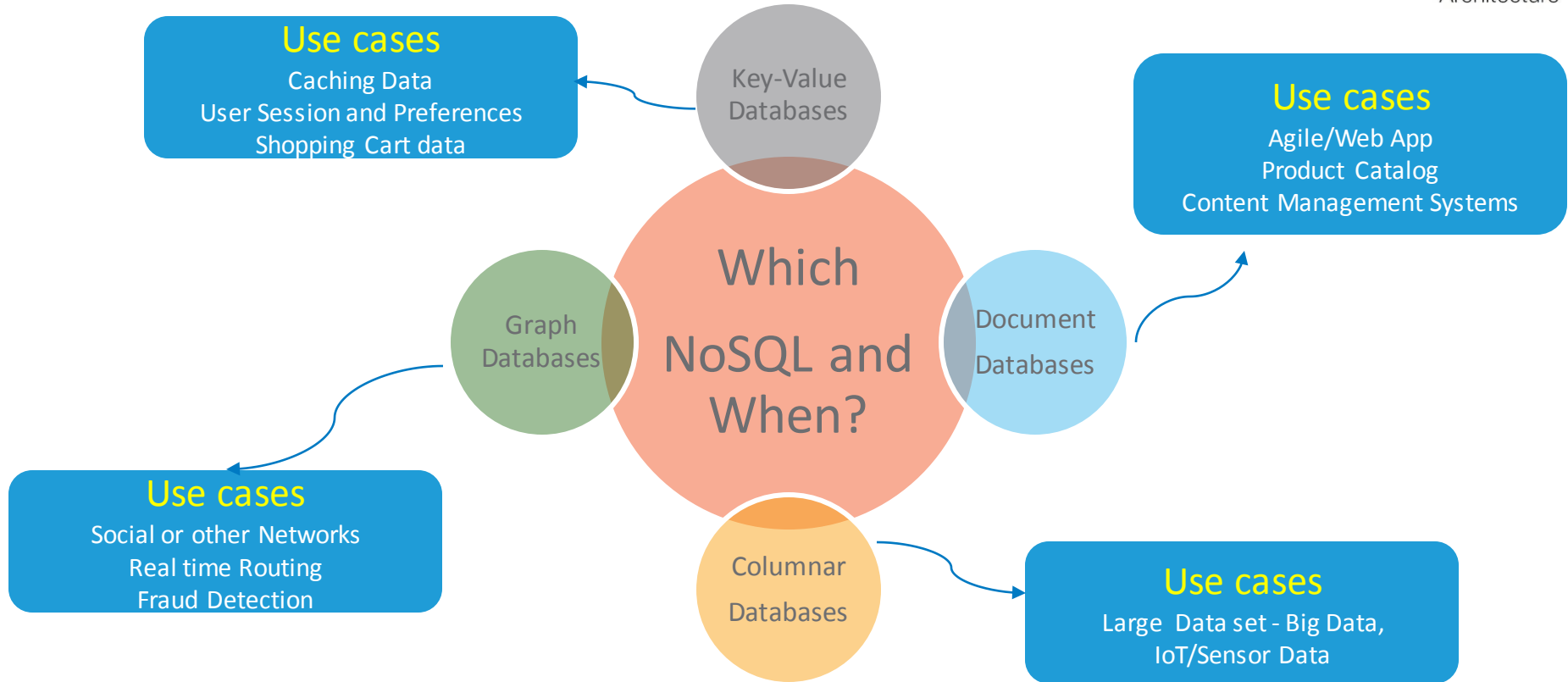


Neo4J



OrientDB

Choosing the right NoSQL



Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. Deep Dive - MongoDB Architecture

Key-Value Databases - Topics



Key-value NoSQL Overview

Key-Value NoSQL Overview

Definition

- Big Hash Table of keys & values
- A bucket with logical group of keys

Key Use-cases

- Data with indeterminate form
- Data that is big in size or quantity
- Data with no relationships

CAP Theorem

- High Availability and high Partition
- Low Consistency

Popular products

- Amazon DynamoDB, Redis and Riak

Key-Value NoSQL Overview

Advantages

- Extremely simple and extremely fast even amongst NoSQL
- Virtually no query engine
- Direct data access over REST like API
- Easy integration with big data processing

Anti-patterns

- Normalized data
- Modelling relations in data

Design considerations

- Access Patterns
- Modelling happens at application layer
- ACID and Transactions at application layer
- Encryption, versions, mutable / immutable

Future roadmap

- Query engines, Strong consistency and Secondary Indexes

Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. Deep Dive - MongoDB Architecture

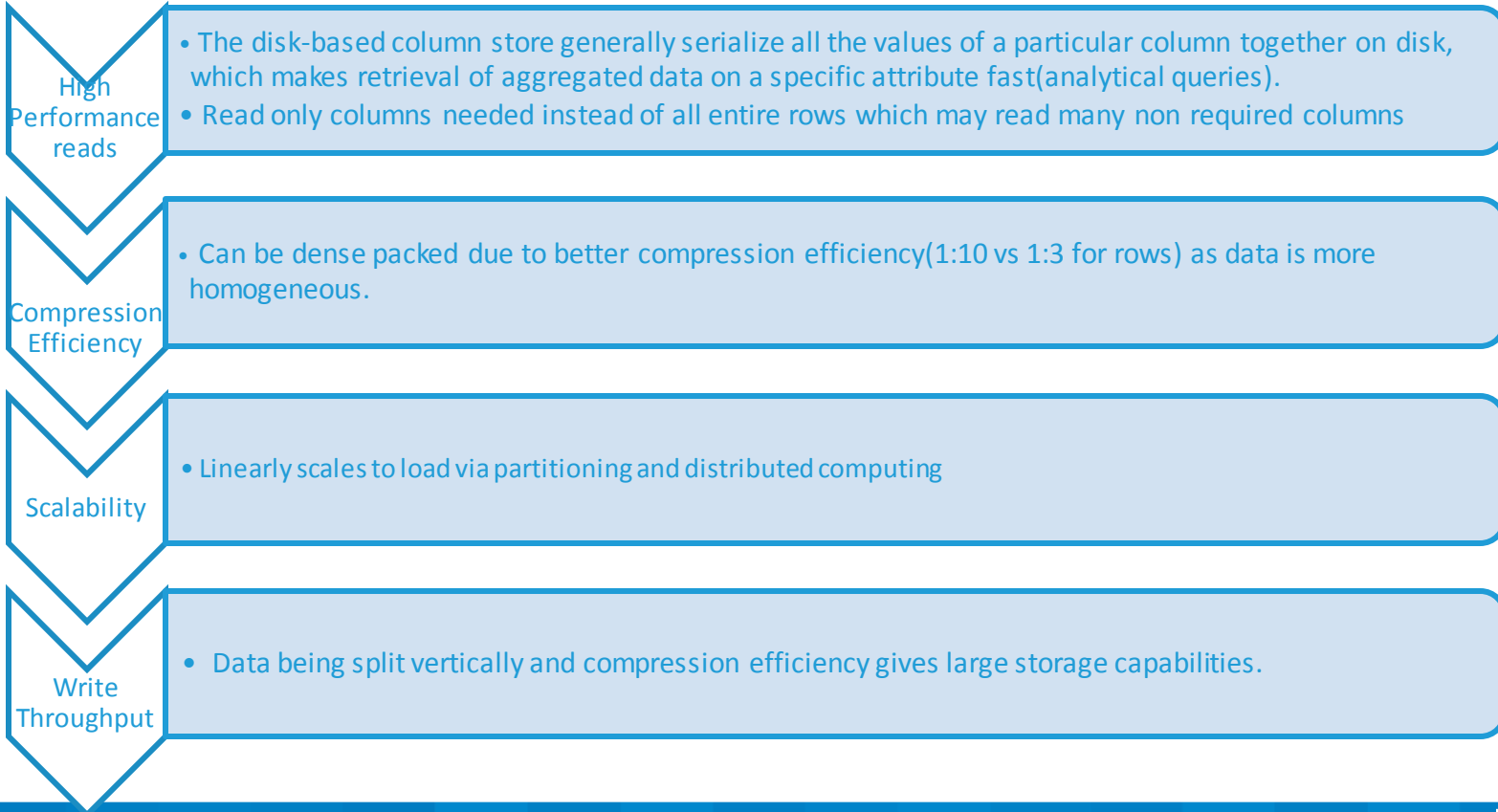
Columnar Databases - Topics



Columnar Databases Overview

Case study – Understanding Customer Trends in Banking Domain

Why Columnar Database?



Columnar Database – Key Attributes

Column Aggregate
oriented

Sorted Key-Value
Map Storage

Typically
Collections,String,Blob
format

Highly Fault
Tolerant

No predefined
schema

Queries

- Partition Key based

Consistency
- Eventual/Tunable

Popular Columnar Databases

Columnar Database	Cassandra	HBase	HyperTable
Implementation	Java	Java	C++
Secondary indexes	Restricted	No	Restricted
Consistency	Tunable	Immediate	Immediate
Access Control	Yes	No	Yes
Peer-to-Peer Architecture	Yes	No(Master-Slave)	Yes
Map Reduce Support	Supports MR	Best for Batch/ETL using MR+HDFS	Supports MR

Case Study – Cassandra

Problem Statement

- Need to Log more data became prolific to aid better understanding customer trends towards channel / products
- Oracle(11g) which is a RDBMS database have known limitations to achieve scalability due to (shared disk) SAN storage architecture. It was a single point of failure and cost to achieve more scalability was growing.
- Growing volumes of transactions and increasing associated cost made business and technology to look for an alternatives beyond traditional/legacy solutions available.

Requirements

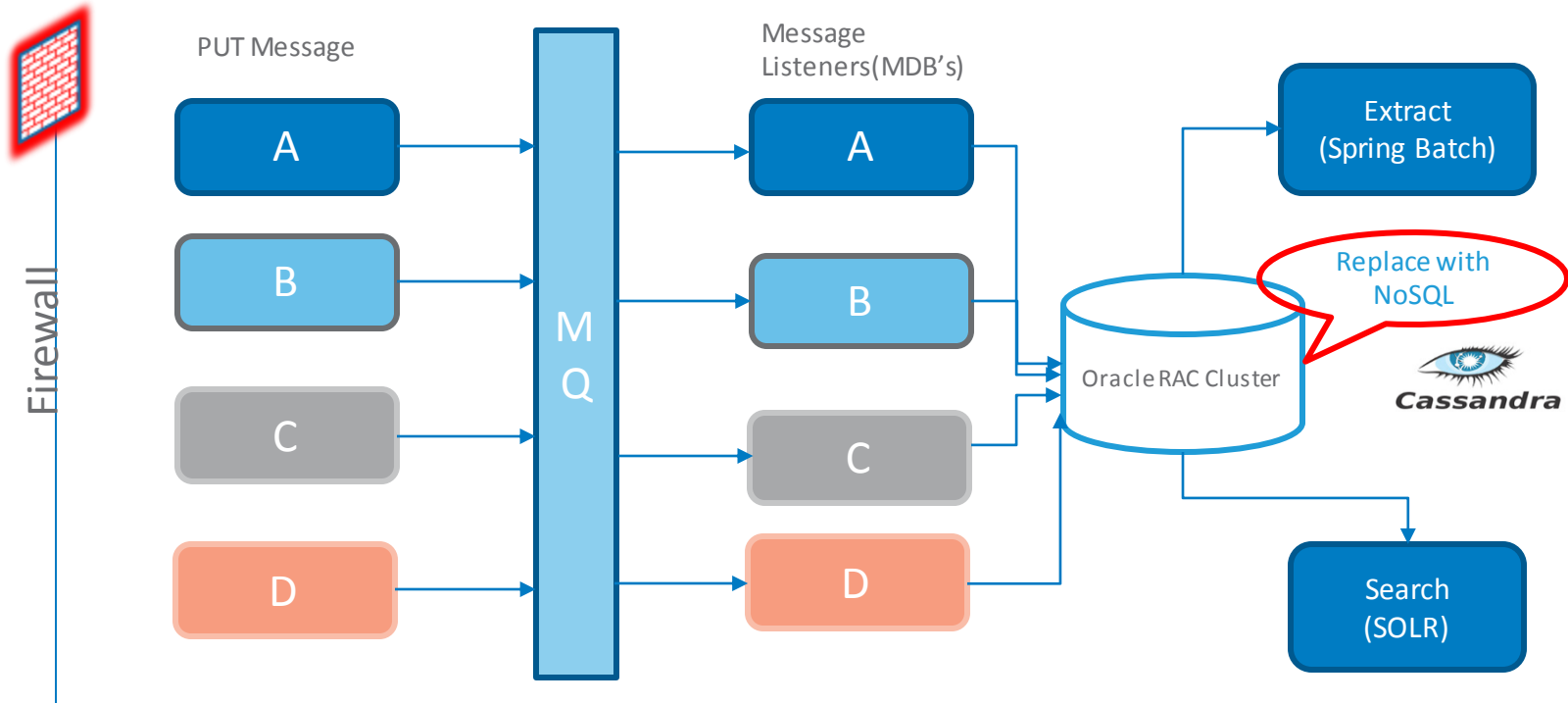
Functional

- I. Able to write performance data, client audit data, rule input/output and other transactional non-relational data into database successfully.
- II. Able to extract the data to create flat files (sent to DW for reporting & analytics) from logged data in database.

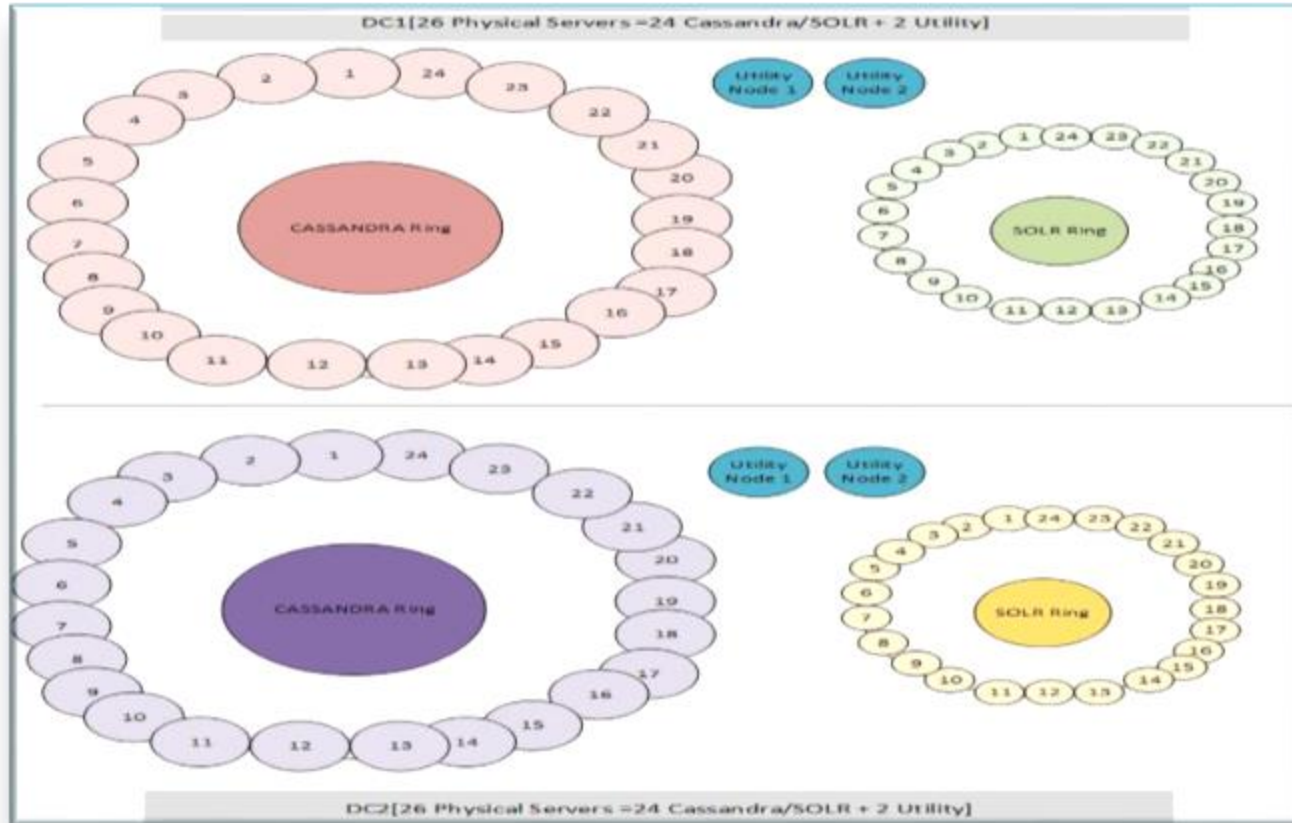
Non-Functional

- I. Throughput (Read 5 Million/Min – Writes 16k/sec)
- II. Continuous Availability
- III. Cost reduction (As less as possible; < Oracle Incremental Expansion ~ 1.3M/yr – based on yearly volume forecast)

Architecture Views - Technology



Architecture Views - Deployment



Conquering Challenges

Challenges

- ❑ NoSQL – A new DB paradigm.
- ❑ No Support of NoSQL Data Source in Spring Batch.
- ❑ Frequent release cycle by DataStax of Cassandra.
- ❑ Cassandra Client & PT Tools limitations.
 - Hit multiple roadblocks in EPT [Inadequate capacity planning(resources ,efforts); Wasn't able to push the volumes required in production due to inherent system design issues(downstream systems availability; data quality issues)
- ❑ Write/Read Performance
- ❑ Support for Container Managed Transactions(JDBC Data Source/Connection-JSR 322 compliant ResourceAdapater for Cassandra Session)
- ❑ Spring Data Cassandra API issues

Overcoming Challenges

- ❑ Extended Spring Batch Item Readers to support Cassandra
- ❑ Implemented Custom Multithreading using Java 5 Executor API within Spring Batch ItemReader for faster reads(30T per job)
- ❑ Performance Tuning at Server(various policies) and Application Level.
- ❑ Application managed connections using Spring-Data-Cassandra(takes care of cluster & session management)
- ❑ Caching Prepared Statements

Learnings

Data Size on Spindle Drive

- Cassandra Nodes mounted on Spindle Drives wasn't able to take data size > 1TB/node compared to 5TB/node prescribed by DataStax.

Data Modeling

- Don't think of relational tables; think of nested sorted map instead.

Performance/Memory Usage

- For better write performance, use `executeAsynchronously()` instead of `ingest()` from Spring Data Cassandra
- Modified code to create dynamic Queries to avoid NULL values inserts into C* column families.
- Reduced the TTL and GC Grace values to lower data retention time(disk/in-memory)

Load Balancing

- Make use of load balancing TokenAware Policy
- Wrapped DC Aware Policy(which is wrapped by TokenAware policy) since we have MultiDC setup.

Compaction Strategy

- DTCS(Date-tiered Compaction Strategy) is chosen which suites time-series data

Best Practices

Data Modeling

- Model Column Families around Query Patterns(& your use case)
- De-normalize and duplicate for performance.

Partition Key

- Decide on proper row key – “partition key”(along with indexes)
- Use default MurMur3Partitioner which provides faster hashing and better performance to RandomPartitioner.

Logging

- Parallel Logging on Cassandra and Oracle based on rollout flag with auto-switch and auto-shut off mechanism.

Storage

- Keep names of Column Family names as small as possible as it has storage capacity overhead.

Failure Handling – Retry logic

- Inbuilt retry logic for failure handling for read during retries.

Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. **Graph Databases**
5. Document Databases
6. Deep Dive - MongoDB Architecture

Graph Databases - Topics



Graph Databases Overview

*Work in Progress

Overview – Graph Databases



- **Graph Database(DB)** with support for representing “entities” and “relation between entities”, can model non-linear relationship, with **significant reduction in overhead and administration cost** as compared to relational database
- **Graph DB** can map, homogenous or heterogeneous, domains together like human relation, IT infrastructure, financial transactions, product line management, with significant improvements in business service and agility.
- A few important **Graph DB** implementations are – **Neo4j**, Infinite Graph, OrientDB, FlockDB.

Neo4j is a popular Open Source No-SQL graph database and follows directed graph model. Few important features:

- Neo4j provides high performance with read and write scalability
- Neo4j meets reliability and transactional integrity requirement of carrier grade applications
- Neo4j can be queried using cypher language. Libraries are available in Java, Python to build applications
- Neo4j can run as independent entity or can be embedded within an application
- Neo4j provides backup capability
- Neo4j provides in-built web based administration tool

Graph Databases – Use Cases

Example Domain	Sample Use Case
Logistics	Direct mapping of logistic network in Graph Database prevents the “conversion overhead” from “Route Network” to “Relational Database” and vice-versa. Graph Database can act as <u>source of route data</u> while performing route calculations.
Finance	High volume financial transactions present real-time computational challenge to associate individual data records for fraud/money laundering detection. Graph Database is well suited to <u>process evolving data records and uncover difficult-to-detect patterns</u> .
Social Network	Exponential growth of social network present scalability challenges. Graph Databases can <u>provide real-time recommendations/notifications</u> to the end users.
Security	Graph Databases can track identity based access management of resources, thereby aiding organizations to easily maintain different access roles, groups and link <u>access policies</u> .
Network	Graph Databases allow network entities and their relations to be <u>represented as “network”</u> rather than relational table. Thus improving performance of network queries along with ease of network maintenance.

Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. Deep Dive - MongoDB Architecture

Document Databases - Topics



Document Databases Overview

Typical Use Cases

Why Document Database?

Flexible Schema

- Documents within the same Collection can have different structure
- Can Store Unstructured and Semi-Structured Data
- No need for Object-Relational Mapper (ORM)
- Allows query based on document content

Scalability

- Related data stored together (in a single unit)
- Scaling Out using Sharding

Performance

- Extensive use of RAM to speed up database operations
- Used when Eventual Consistency is sufficient compared to Strict Consistency in RDBMS

Availability

- Through Replication and Failover

Document Databases – What are they?

Aggregate oriented

Document Storage

Typically XML, JSON, BSON
format

Complex data
types

maps, collections, another
document

No predefined
schema

Queries

- Key based
- Fields inside document

Document Database Use Cases

- Collect and Store machine generated data from
 - logging systems
 - application output

Operational
Intelligence

- e-commerce systems
 - building product catalogs
 - managing inventory

product data
management

- Building Content Management Systems (CMS)

content
management

Popular Document Databases

	MongoDB	CouchBase
Data Model Representation	BSON (Binary JSON) - Supports Arrays, Binary Data and sub-documents	JSON format
Consistency	Strong Consistency CAP Theorem – CP	Eventual Consistency CAP Theorem - AP
Replication & Availability	Replica Set – Primary & Secondary (Master -> Slave)	Replication – Peer to Peer
Security	Authentication, Collection-Level Access Control, Auditing, Enabling SSL, Data Encryption...	SSL Support

Session Plan

1. Introduction
2. Key-Value Databases
3. Column Databases
4. Graph Databases
5. Document Databases
6. **Deep Dive - MongoDB Architecture**

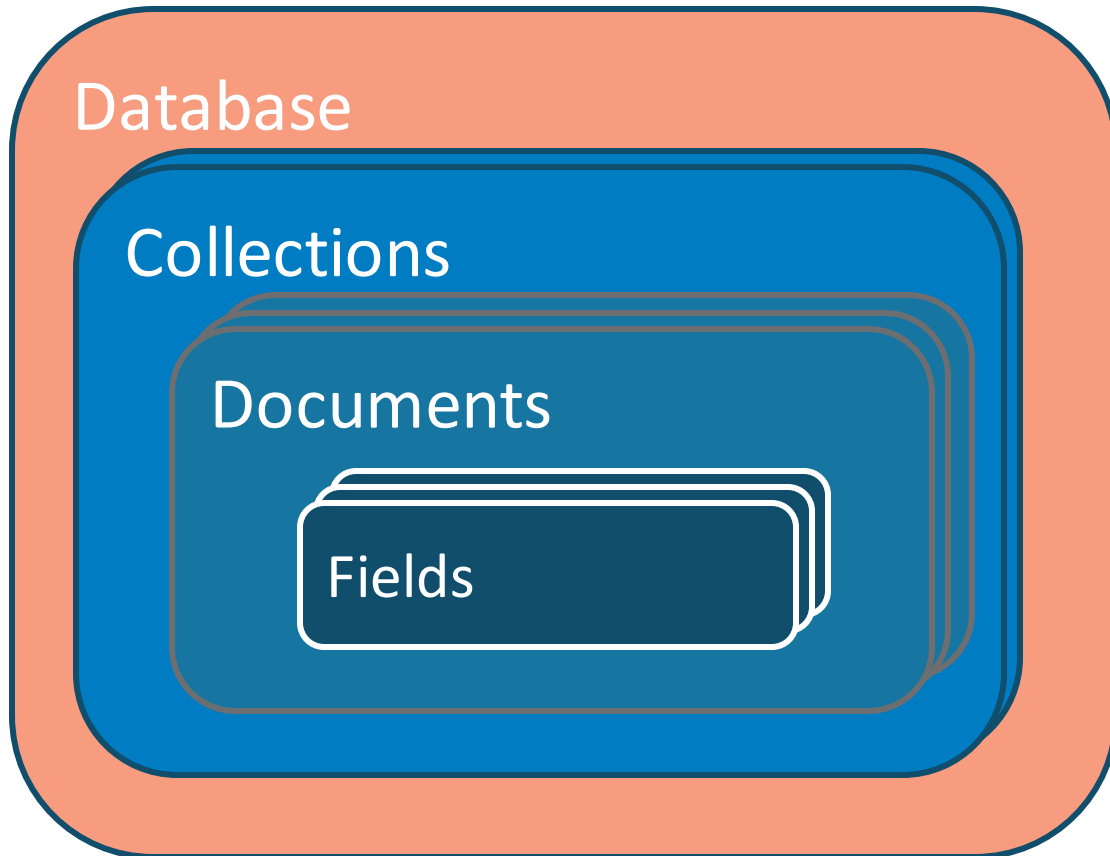
MongoDB Deep Dive - Topics

MongoDB Basics

MongoDB Schema Design

MongoDB Architecture

Database, Collections and Documents



Documents and Collections

- Documents
 - Stored in BSON (Binary JSON) format
 - contains field-value pairs
 - Each document stored in a collection
 - Can embed other documents
 - Have Flexible Schema (Schemaless)
- Collections
 - Have index set in common
 - Documents in a Collection need not have uniform structure

Simple document:

```
{
  "_id" : ObjectId("3rm5gc8d7fdrs78236321015"),
  "handle" : "ramk",
  "name" : "Ram Kumar"
}
```

- Unique, immutable
- Serves as Primary key for collection

Document with embedded documents

```
{
  "_id" : ObjectId("3rm5gc8d7fdrs78236321015"),
  "handle" : "ramk",
  "name" : "Ram Kumar",
  "courses" : [
    { "title" : "MongoDB",
      "NumStudents" : "50",
    },
    { "title" : "Hadoop",
      "NumStudents" : "30",
    }
  ]
}
```

MongoDB – Schema Design – Use Case

- Consider building website for **E-Commerce** Company selling Mobile Products with products under various categories. The users place order for the same online and purchase them. The products are shipped to the shipping address upon completion of the order and corresponding payment.

Some of the typical operations (Illustrative)

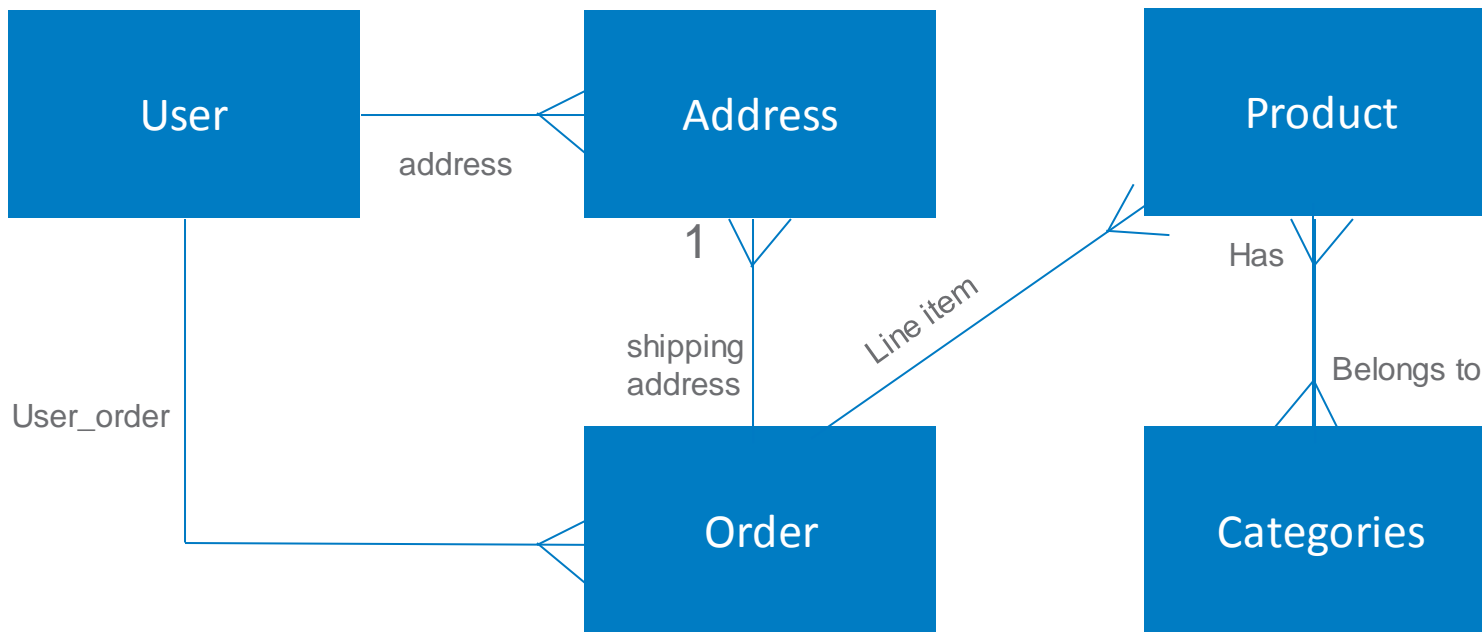
Data Retrieval :

- Get all the mobile products of a given category
- Get all the Users who have bought “XYZ” Product
- Search for Orders given UserID and a Product and Order Date
- Get all the Orders for a given product in last 1 month
- Get total sales in a given month

Data Modifications :

- Add new products
- Delete a product
- Delete products of a given category
- Add new users
- A given User places an order for Products (say P01, P10, P21) to be shipped to Home Address
- Change User Profile and update Home Address
- Change User Profile and update Work Address
- Cancel an Order given an Order ID

Relational Schema for Ecommerce Application



Schema Design Considerations

- Access Patterns?
 - Read / Write Ratio
 - Types of updates
 - Types of queries
 - Data life-cycle
- Considerations
 - No Joins
 - Single Document writes are atomic

Schema Design – One to One

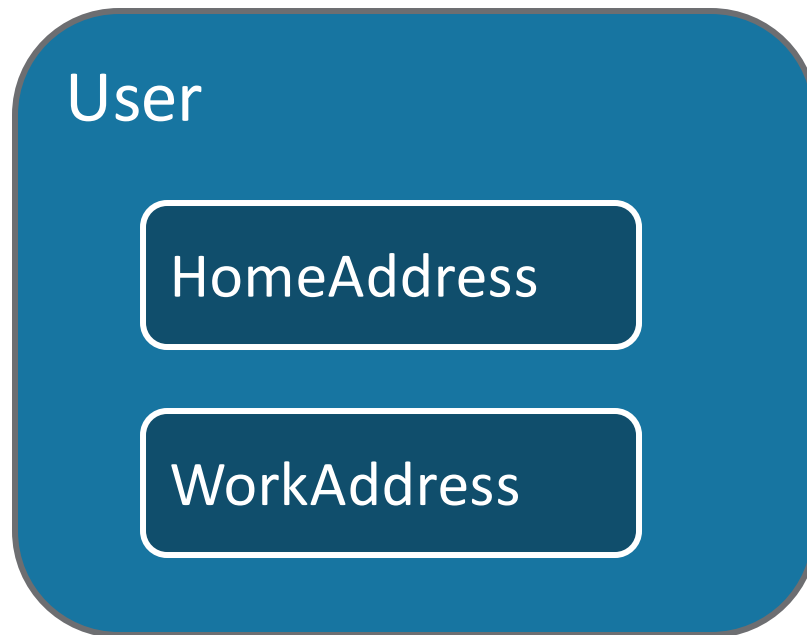
- Order – Shipping Address
 - Shipping Address is not queried upon independent of Order
 - Option Chosen: Embed

Order


ShippingAddress

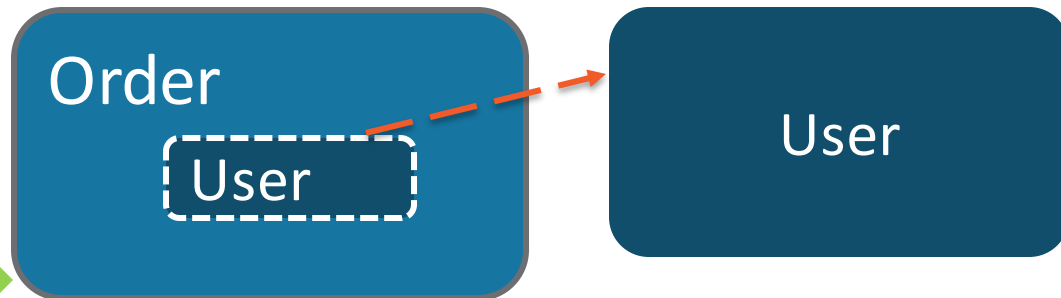
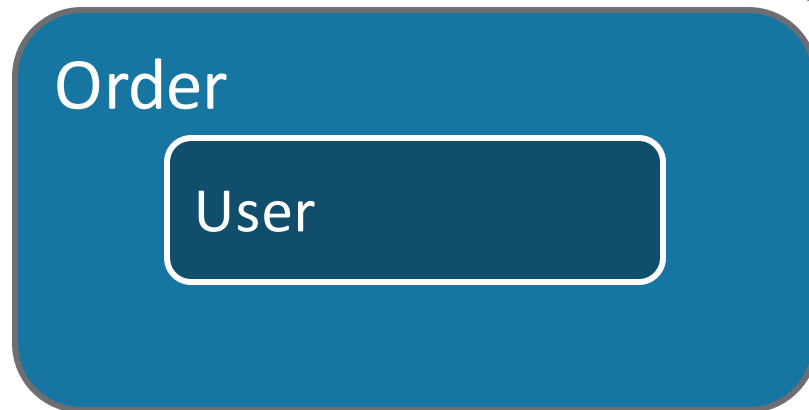
Schema Design – One to Many

- User – Address
- Low Cardinality
 - Address is not queried upon independent of User
 - Option Chosen: Embed



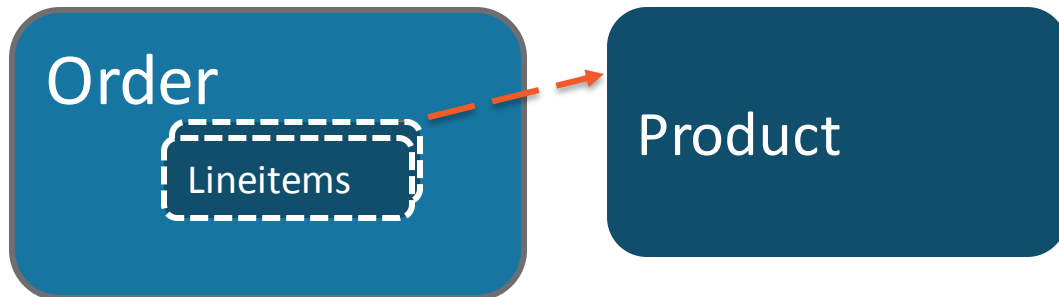
Schema Design – One to Many

- User → Order
- User can be queried upon independent of Order and vice-versa
- UserID and UserName does not change
- Options:
 - Embed Entire User Document within Order Object
 - Link User with Order Object  (copying UserID and UserName attributes)



Schema Design – One to Many

- Order → Lineitems (Products)
 - Order can be queried upon independent of Products and vice-versa
 - Usually the lineitems (products) are fixed once order is placed
 - Options:
 - Embed Entire Products Document within Order Object
 - Link Products (Lineitems) with Order Object
- With attributes copied
- Without copying attributes

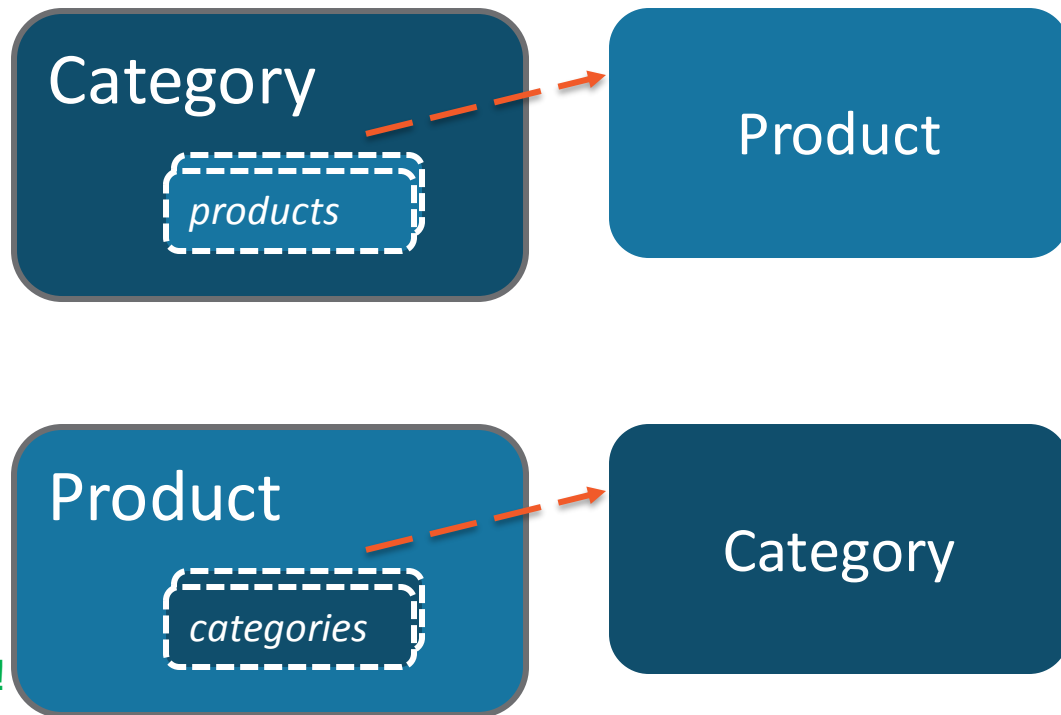


Schema Design – Many to Many

- Product \leftrightarrow Categories
- Categories can be queried upon independent of Products and vice-versa
- Categories are much less in Number compared to Products
- Options:
 - Link Products (*products*) with Category Object
 - Link Categories (*categories*) with Product Object


Preferred!

Both options are alright



MongoDB Architecture



Data Management

- Document: BSON format, schema-less and agile, contiguous blocks storage
- Query Model: Key-value, Range, Geospatial, Text search, Map-reduce functions
- Types of Indexes: Unique, Compound, Array, Text Search, Geospatial

Auto Sharding

- Automatic data distribution as data grows
- Types of Sharding : Range-based, Hash based and Tag aware
- Sharding transparent to the applications – Query Router - ConfigDB

Transaction Management

- guarantees atomicity at the single document level
- Does not support Multi-document transactions

Availability

- Multiple copies of data are stored in different servers with native replication
- Automatic failover to secondary nodes, racks and data centers

Replication in MongoDB

- Purpose of Replication/Redundancy
 - Failover/Availability
 - Scaling reads
 - Primaries, secondaries and arbiters
- **Primary** – Read and Write
- **Secondaries** – Read
 - Can become Primary
- **Arbiter**
 - Voting
 - Cannot become Primary

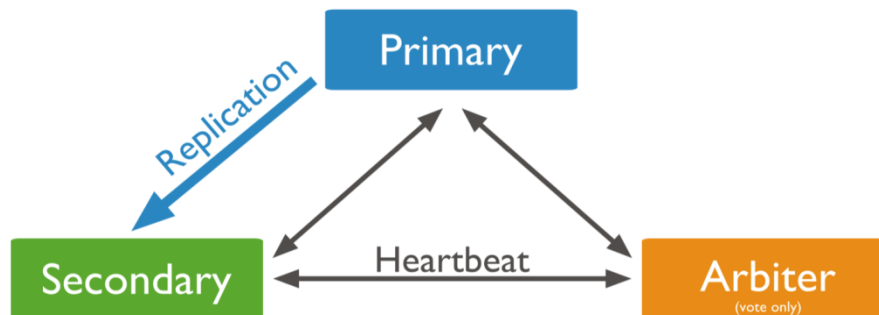
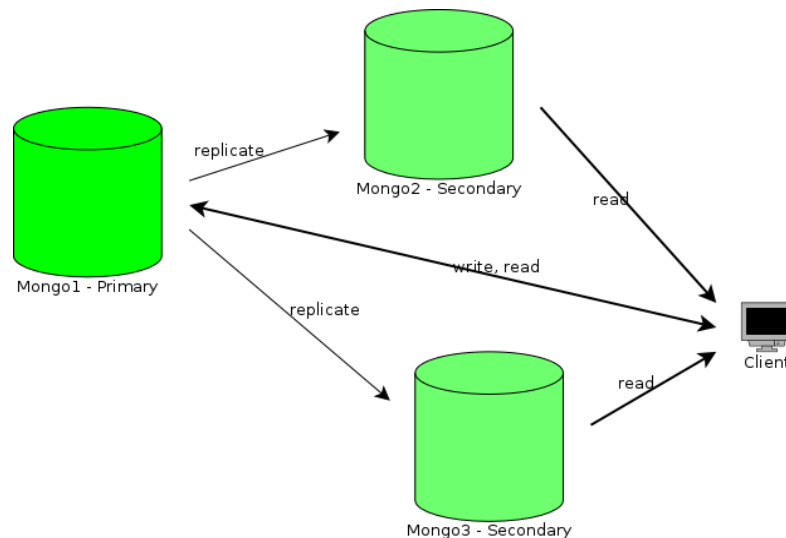
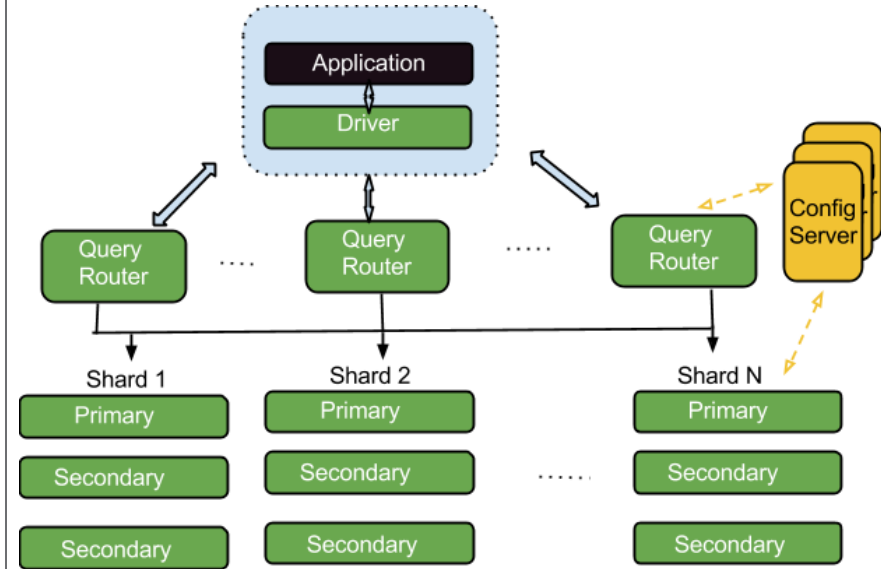


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Sharding in MongoDB

- Based on defined shard key.
- Enables horizontal scaling across multiple nodes.
- Application connects to the sharded cluster through a mongos process, which routes operations to the appropriate shard(s).
- Sharding is performed on a per-collection basis.
- Small collections need not be sharded.
- Auto-Balances as shard servers are added or removed
- Failover handled through replica sets.



Thank You



© 2016 Infosys Limited, Bangalore, India. All Rights Reserved. Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this documentation nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/or any named intellectual property rights holders under this document.