NoSQL and MongoDB

An Introduction

NoSQL: Features

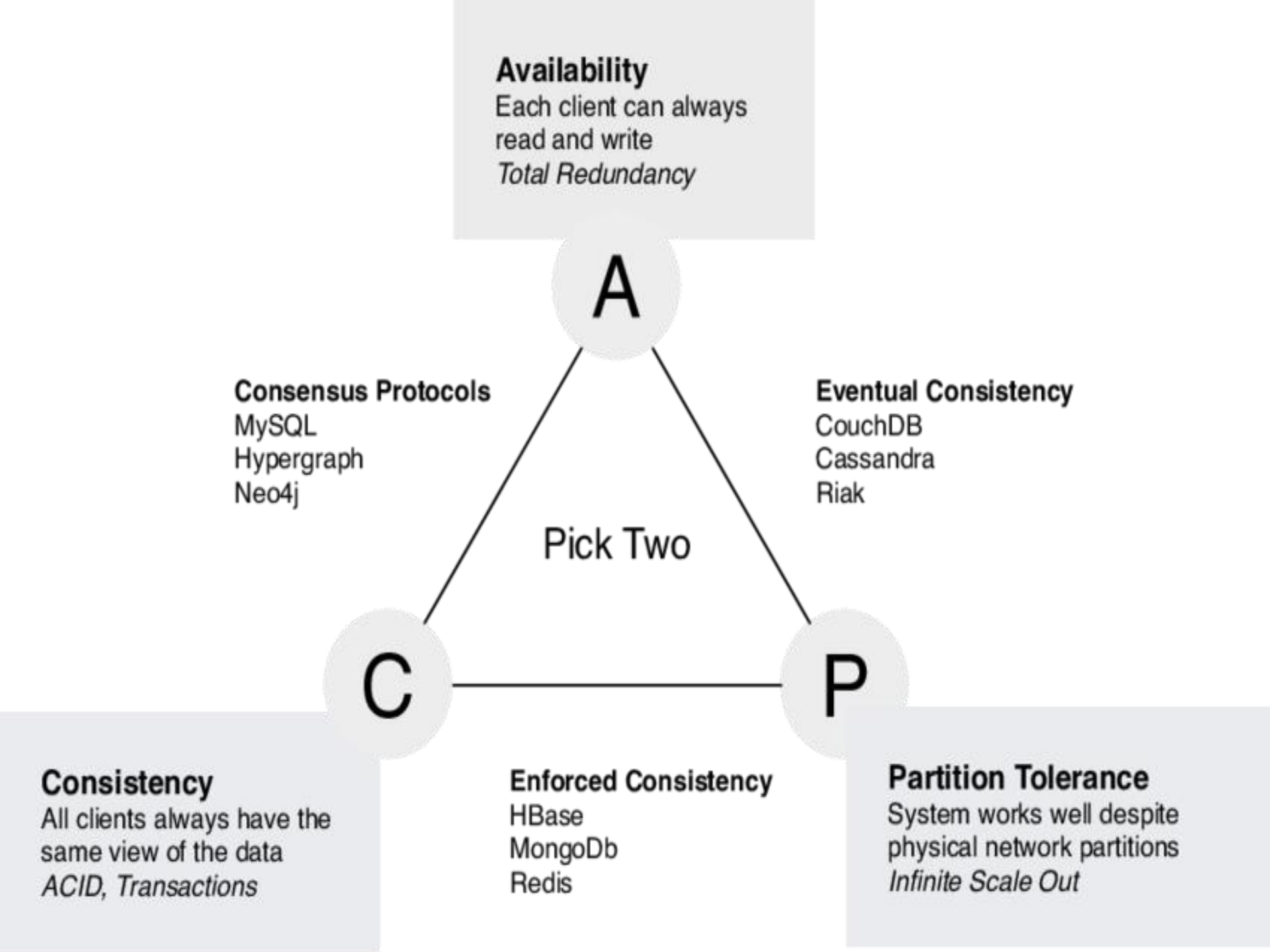
- NoSQL stands for “Not Only **S**QL”
- Term ‘NoSQL’ was coined by Carlo Strozzi in the year 1998.
- Next Generation Database
  - Open Source
  - Non Relational
  - **Distributed**
  - **Horizontally Scalable**
  - Can be deployed on **Commodity Hardware**
  - Can handle huge amount of **Semi-Structured and Unstructured** data as well

NoSQL: Web Scale Database

- Modern Web Scale Database
  - Simple **API**
  - **Schema-free** (flexible schema)
  - Easy **replication** support, Automatic **Failovers**
  - Follows Eric Brewer's **CAP theorem**
  - **BASE** Compliant, NOT **ACID**

NoSQL: CAP Theorem

- Applicable on Distributed Systems.
- Also known as Brewer's Theorem – Named on Scientist Eric Brewer
- No Distributed System can met following 3 requirements simultaneously -
  - **C**onsistency - Every read would get the most recent write. Commits are atomic across the distributed system.
  - **A**vailability - Every node (if not failed) always executes queries.
  - **P**artition Tolerance **-** Even if the connections between nodes are down, the other two (A & C) promises, are kept


Availability
Each client can always read and write
Total Redundancy
A
Consensus Protocols
MySQL
Hypergraph
Neo4j
Eventual Consistency
CouchDB
Cassandra
Riak
Pick Two
C
P
Consistency
All clients always have the same view of the data
ACID, Transactions
Enforced Consistency
HBase
MongoDb
Redis
Partition Tolerance
System works well despite physical network partitions
Infinite Scale Out

NoSQL: BASE

- BASE Compliant
  - **B**asically **A**vailable
  - **S**oft state
  - **E**ventually consistent
- **Basically Available** indicates that the system *does* guarantee availability, in terms of the CAP theorem.
- **Soft state** indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.
- **Eventual consistency** indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

NoSQL Databases: Types

- Key-Value/Tuple Store
  - Simplest NoSQL databases
  - Every single item in the database is stored as an attribute name (key) together with its value
  - e.g. BerkeleyDB, Oracle's NoSQL Database
- Wide Column Store
  - Optimized for queries over large datasets
  - Store columns of data together instead of rows.
  - e.g. Hadoop HBase, Apache Cassandra, Cloudera

NoSQL Databases: Types

- Document Store
  - Pair each key with a complex data structure known as a document
  - Documents can contain many different key-value pairs or key-array pairs, or even nested documents
  - e.g. **MongoDB**, Apache CouchDB
- Graph Databases
  - Used to store information about networks, such as social connections.
  - e.g. Neo4J, HyperGraphDB

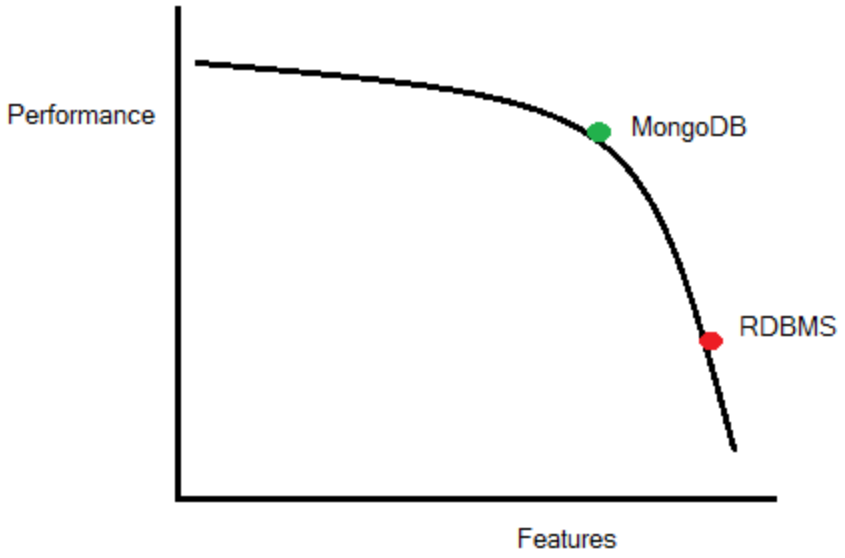
MongoDB

- Free Software Foundation's GNU AGPL 3.0 License
- Commercial License available from MongoDB Inc.
- Available for Windows/Linux/Mac/Solaris. Also available on Cloud as a Service through MongoDB Atlas
- Supports **Big Data** and **Map-Reduce**
- Uses Document-Oriented Model
- Dynamic Schema/Schema less

MongoDB

- Highly Scalable
- Distributed Horizontal Scaling (Scale-Out)
- High Performance
- High Availability
- Advanced GUI, Monitoring and Backup Service
  - MongoDB Compass - GUI
  - MongoDB Monitoring Service

Performance Vs. Features



MongoDB: Features

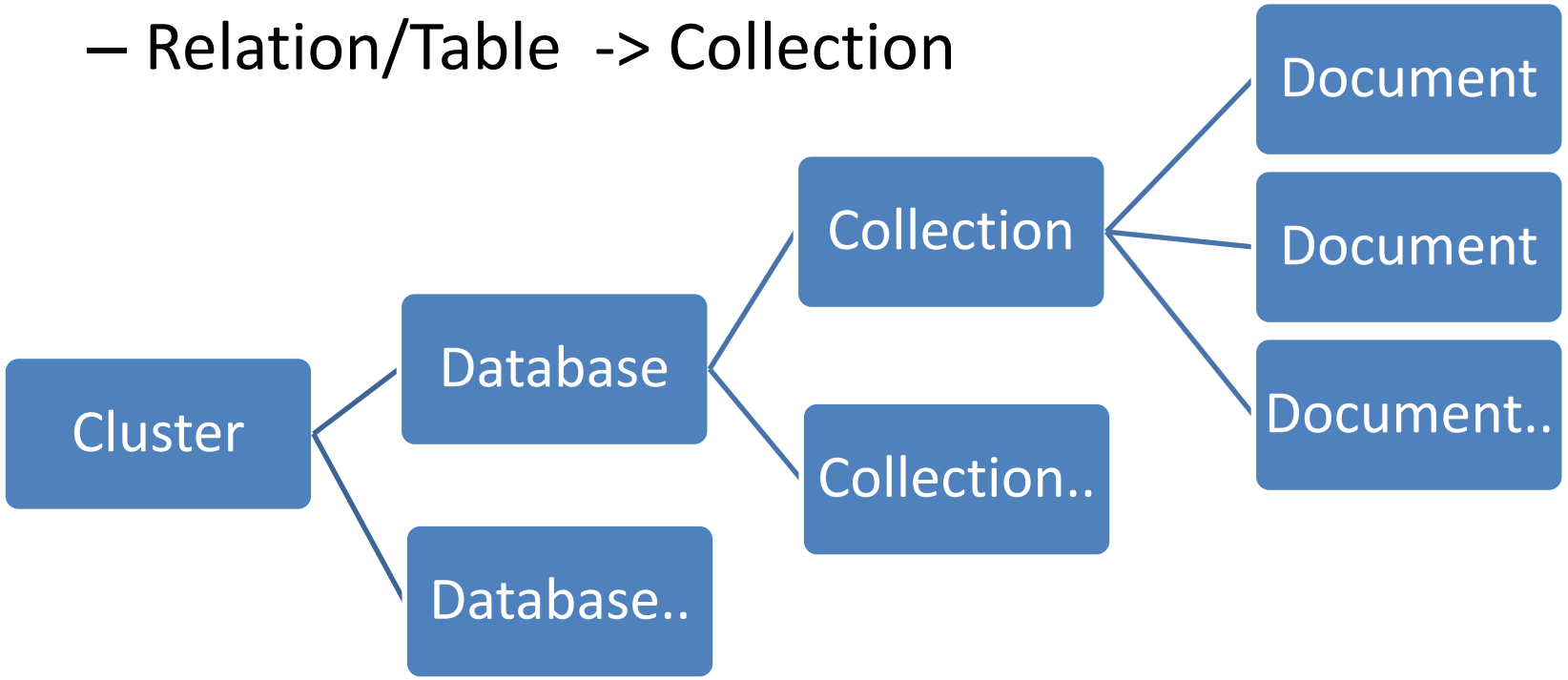
- Provides **Consistency** & **Partition Tolerance** as per CAP Theorem. Some data may not be available, but the available data is Consistent & Accurate.
- Handling of Complex data possible
- Fast application development
- **GridFS** for storing Large Objects
  - Can store 100s of TBs of Data by splitting into smaller chunks
  - A **BSON** object can't be larger than 16MB
- Does NOT support Complex Transactions
- Does NOT support Joins

MongoDB: Document Oriented

- Stores data in Key-Value pair
- Documents are of two types –
  - JSON : Java Script Object Notation
  - BSON : Binary JSON

MongoDB Object Hierarchy

- Terminology compared to RDBMS
  - Row/Record -> Document
  - Relation/Table -> Collection



JSON Sample Document

JSON {

id : “A1”

X : 3

Y : “abc”

Z : {1,2}

E :

}

JSON

- Supports Nested Documents
- No Relations – No Joins

Connecting with Mongo Shell

- To connect to Mongo shell - (‘test’ Database)

  > mongo

  OR

  > mongo localhost/test

- Mongo Commands

  > show dbs

  > show collections

  > db

  > help

  > exit

Starting with MongoDB

- mongod -> binary to start Mongo Process/Demon
- mongo -> binary to start Mongo Shell

- To Start Mongo Process
  > **mongod**
- Other important demon parameters -
  > mongod --version
  > mongod --help
  > mongod --dbpath C:\data\dbDefault

- Mongo db runs on Port : 27017 by default

Creating a Database/Collection

- Creating a database

  > use <dbname>

  Eg. > use sengdb

- Creating a collection

  > db.createCollection(“collectionname”)

  > db.<collectionname>.insert({“Key”: “Value”})

Mongoimport

- Sample JSON data can be found at http://media.mongodb.org/zips.json

  > mongoimport --stopOnError --db <db name>
  --collection <collectionname> < <datafilename>

  E.g.

  > mongoimport --stopOnError --db sengdb
  --collection mycollection < sampledata.json

CRUD Operations

- Data Selection/Manipulation Operations
  - **C**reate : insert()
  - **R**ead : find()
  - **U**pdate: update()
  - **D**elete : remove()

  - **Up**date, OR In**sert** if does not Exist : upsert()

CREATE: insert()

Create => Insert

> db.mycollection.insert({“name”: “Tom”})

_id field is insert automatically by mongo if not specified

_id field indentifies each record uniquely and used for indexing

READ: find()

- db.mycollection.find()
  - Top 20 docs, ‘it’ command to display more docs
- db.mycollection.find({“city” : “Mumbai” })
- db.mycollection.find().limit(10)
- db.mycollection.find().limit(2).pretty()
- db.mycollection.find().skip(1)

UPDATE: update()

- Update are of two types
  1. Full Document Update / Replacement
  2. Partial Document Update
- db.<collectionname>.update(<criteria>, <doc/partial update>)
- _id field can not be updated through update operation

UPDATE: Update()

- db.mycollection.update({“_id”:"35004"}, {**$set**:{country:"GHANA"}})
- db.mycollection.update({"_id":"35004"}, **{$push**:{ arr: "hi" } })
  - (updates in an array) – if updated field does not preexists it creates it.
- db.mycollection.update({"_id":"35004"}, {**$addToSet**:{ arr: "hi" } })
  - (updates in an array) - Only add once if not there

DELETE: remove()

- db.mycollection.remove({})
  - removes all documents from the collection
- db.mycollection.remove({“_id”:"35004"})
- db.mycollection.remove({ “x” : /hi/ })
  - removing documents using regular expressions

Operators

- $gt
- $gte
- $lt
- $lte
- $exist
- $in
- $nin
- $type
- $or
- $not

JSON Data Types

- Number (Integer or Floating Point)
- String (In double quotes)
- Boolean (true or false)
- Array (In square brackets – [])
- Object (In curly brackets – {})
- NULL

- Keys must always be string and must be written in double quotes

BSON

- Standard to represent JSON in binary format
- Internal format used by mongo database and drivers
- Lightweight
- Provides Scannability & Additional data types
- BSON additional data types –
  - Date
  - BinData (a byte array of photos, UUIDs etc.)
  - ObjectID
- Additional Info @ http://bsonspec.org

ObjectID

- Primary Key for each document in a Collection
- Named as “_ID” field
- Its value can be of any data type
- Assigned automatically if not explicitly defined

MongoDB Schema

- Theoretically Schema-less
- Have Dynamic Schema
- Flexible – agile
- Polymorphic data representation

- NO ‘ALTER TABLE’ required
- Dynamically Typed, Things have types but they are resolved at runtime.

References

- www.nosql-database.org
- university.mongodb.com
- docs.mongodb.org

Mongo Shell Commands Quick Reference

- https://docs.mongodb.com/manual/reference/mongo-shell/