Week1_NoSQL_Overview

Basics of NoSQL

What is NoSQL

Stands for Not Only SQL

- Refers to a family of databases that vary widely in style and technology
- However, they share a common trait
  - Non-relational in architecture
  - Not standard row and column type RDBMS
- Provide new ways of storing and querying data
  - Address several issues for modern apps
- Are designed to handle a different breed of scale - 'Big Data'
- Tipically address more specialized use cases
- Simpler to develop app functionality for than RDBMS

Characteristics of NoSQL Databases

- Types of NoSQL databases
- **Key-value**
- **Document**
- **Column based**
- **Graph**
- The majority of the NoSQL databases have their roots in the open source community and have been used and leveraged in an open source manner. Companies often develop a commercial version of the database alongside the open source version. Eg:
  IBM Cloudant, datastax and MongoDB.
- Most NoSQL databases:
  - Are built to scale horizontally
  - Share data more easily than RDBMS
  - Use a global unique key to simplify data sharding
  - Are more use case specific than rdbms
  - Are more developer friendly than RDBMS
  - Allow more agile development via flexible schemas

Benefits of NoSQL Databases

- Scalability (horizontally scale across servers, clusters, server racks and data centers)
- Performance (fast response times with large datasets and high concurrency)
- Availability (run on cluster of servers)
- Cloud Architecture
- Cost
- Flexible schemas
- Varied Data Structures
- Specialized Capabilities (modern https apis, data replication robustness, indexing and querying)

Key-Value NoSQL

- Least complex architecture speaking
- Represented as hashmap
- Ideal for basic CRUD operations
- Scale well
- Shard easily
- Not intended for complex queries
- Atomic for single key operations only
- Value blobs are opaque to database
    - Less flexible data indexing and querying

Suitable Use-cases

- For quick basic CRUD operations on non-interconnected data
    - Storing and retrieving session information for web applications
- Storing in-app user profiles and preferences
- Shopping cart data for online stores

Unsuitable Use cases

- For data that is interconnected with many-to-many relationships
    - social networks
    - recommendation engines
- When high-level consistency is required for multi-operation transactions with multiple keys
    - Need a database that provides ACID transactions
- When apps runs queries based on value vs key

Examples: amazon dynamoDB, oracle nosqldatabase, redis, aerospike, riak, memcached and project voldemort

Document-Based NoSQL

- Values are visible and can be queried
- Each piece of data is considered a document
    - Tipically a JSON or XML
- Each document offers a flexible schema
    - No two documents need to contain the same information
- Content of document databases can be indexed and queried
    - Key and value range lookups and search
    - Analytical queries with MapReduce
- Horizontally scalable
- Allow sharding across multiple nodes
- Typically only guarantee atomic operations on single documents

Suitable Use Cases

- Event logging for apps and processes - each event instance is represented by a new document
- Online blogs - each user, post, comment, like, or action is represented by a document
- Operational datasets and metadata for web and mobile apps - designed with Internet in mind (JSON, restfulAPIs, unstructured data)

Unsuitable Use Cases

- When you require ACID transactions
    - Document databases can't handle transactions that operate over multiple documents
    - Relational database would be a better choice
- If your data is in an aggregate-oriented design
    - If data naturally falls into a normalized tabular model
    - Relational database would be a better choice

Examples: IBM Cloudant, mongoDB, chouchdb, terrastore, orientdb, chouchbase, ravendb

Column-based NoSQL

- Spawned from Google's bigtable
- Columnar od wide-columns databases
- Store data in columns or groups of columns
- Column 'families' are several rows, with unique keys, belonging to one or more columns
    - Grouped in families as often accessed together
- Rows in a column family are not required to share the same columns
    - Can share all, a subset, or none

    - Columns can be added to any number of rows, or not

Suitable use cases

- Great for large amounts of sparse data
- Column databases can handle being deployed across cluster of nodes
- Column databases can be used for event logging and blogs
- Counters are a unique use case for column databases
- Columns can have a TTL parameter, making them useful for data with an experation value

Unsuitable Use cases

- For traditional ACID transaction
    - Reads and writes are only atomic at the row level
- In early development, query patterns may change and require numerous changes to column-based databases
    - Can be costly and can slow down the production timeline

Examples: cassandra, hbase, hypertable, accumulo

Graph Databases

- Graph databases store information entities (or nodes), and relationships (or edges)
- Graph databases are impressive when your data set resembles a graph-like data structure
- Graph databses do not shard well
    - Traversing a graph with nodes split across multiple servers can become difficult and hurt performance
- Graph database are ACID transaction compliant

Suitable Use-cases

- For highly connected and related data
- Social networking
- Routing, spatial and map apps
- Recommendation engines

Unsuitable Use-cases

- When looking for advantages offered by other NoSQL database categories
- When an application needs to scale horizontally
    - You will quickly reach the limitations associated with these types of data stores
- When trying to update all or a subset of nodes with a given parameter
    - These types of oparations can prove to be difficult and non-trivial

Examples: neo4j, orientdb, arangodb, amazon neptume, apache giraph, jenus graph

Database Deployment Options

This reading introduces you to the concept of Database-as-a-Service (or DBaaS) and enables you to differentiate the available database hosting options.

The major consideration when choosing the best database for your applications and organization is around where to host it and how it is being managed. It is important to understand all of the components to make sure you end up with the simplest and most cost-effective option.

In a traditional in-house do-it-yourself scenario, you will own the setup of the underlying hardware and operating system, installation and configuration of the chosen database management system, overall administration including patching and support, and of course how the application data is designed.

This contrasts a little bit with hosted database solutions. A hosted database solution means the provider is choosing what hardware your database runs on, and they are provisioning it for you. But it is your responsibility to provide and install the software, perform the administrative tasks, and do the design. So, at the end of the day, the hosting provider are handing the administrative keys over to you and it's really up to your team of database administrators to keep things scaling and running smoothly.

In comparison, using a fully managed database-as-a-service (or DBaaS) is really meant to eliminate the complexity and risk of doing it all in-house, and help development teams get to market faster, scale more smoothly and massively, and provide better performance and availability for end users.

The only concern for users of a fully managed DBaaS is the design and development of their product. Guaranteed uptime, availability, and scalability are all the result of a fully managed DBaaS.

You mitigate risk by offloading database administration and data layer management issues from your development team. And you ensure that your developers need only concern themselves with what really matters – developing better applications for your customers.

Working with Distributed Data

ACID vs BASE

They are both consistency models

ACID
Atomic, Consistent, Isolated, Durable

- Used by relational databases
- Ensures a performed transaction is always consistent
- Databases that can ahndle many small simultaneous transactions
- Fully consistent system

Financial institutions - money transfers

BASE
Basically Available, Soft State, Eventually consistent

- few requirements for immediate consistency, data freshness, and accuracy
- availability, scale and resilience
- Used by:
  - marketing and customer service companies
  - social media apps
  - worldwide available online services
- Favors availability over consistency of data
- Fully available system

Worldwide online services - user's access to service (netflix, apple, spotify, uber)

Distributed Databases

It is a collection of multiple interconnected databases that are spread physically across various locations that communicate via computer network. Fragments and replicate the data, following the BASE model

- Fragmentation
  - Fragmenting yout data (partitioning, sharding)
    - Grouping keys lexically (for example, all records starting with A or A-C)
    - Grouping records by key (for example, all records with key StoreId = 123)
- Replication
  - Protection of data for node failures
  - all data fragments are stored redundantly in two or more sites
  - Increases the availability of data
  - Replicated data needs to be synchronized, prone to inconsistency

Advantages

- Reliability and Availability
- Improved performance
- Query processing time reduced
- Ease of growth/scale
- Continuous availability

Challenge

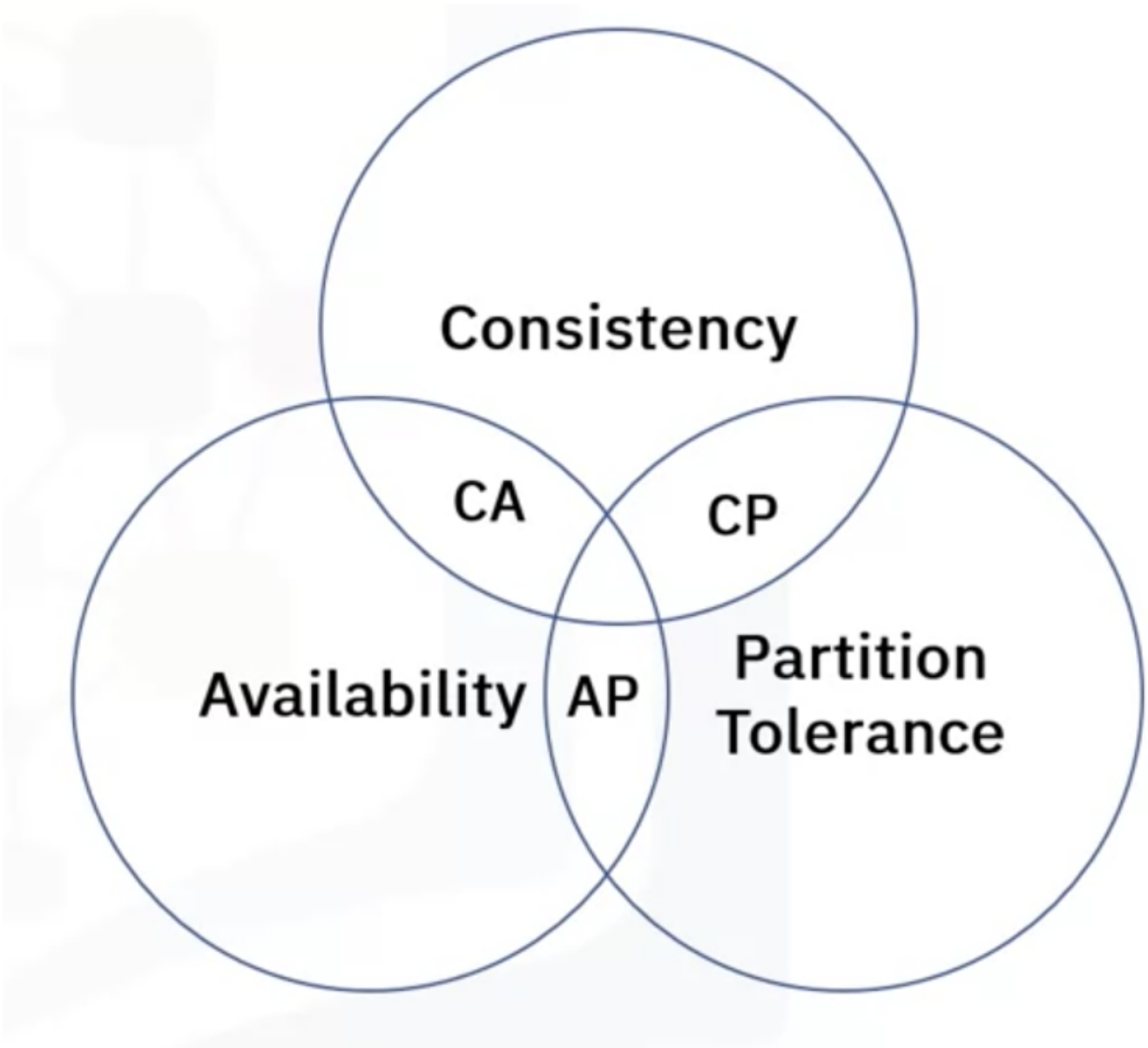
- Concurrency control => consistency of data

    - WRITES/READS to a single node per fragment of data, data is synch in the background
    - WRITE operations go toa ll nodes holding a fragment of data, READS to a subset of nodes per Consistency
    - Developer-driven consistency of data
- No transactions support (or very limited)

The CAP Theorem

- Consistency, Availability and Partition Tolerance
- Consistency: all clients see the same data at the same time
- Availabilty: the system continues to operate even in the presence of node failures
- Partition Tolerance: the system continues to operate despite network failures
    - Partition - a lost or temporarily delayed connection between nodes
    - Distributed systems cannot avoid partitions and must be partition tolerant

Nosql - A Choice between consistency and availability: CP or AP



Challenges in Migrating from RDBMS to NoSQL databases

- RDBMS and NoSQL cater to different use cases
- Decision should be based on careful case-by-case analysis (need for performance, flexibility, etc)

Generic Situations:

- Consistency, Structured data, transactions, joins: RDBMS
- High performance, Unstructured data, Availability, Scalability: NoSQL
- Data driven model to query driven data model
    - RDBMS: Starts from the data integrity, relations between entities
    - NoSQL: Starts from your queries, not from your data. Models based on the way application interacts with the data
- Normalized to Denormalized data
    - NoSQL: Think how data can be structured based on your queries
    - RDBMS: Starts from your normalized data and then build the queries