NoSQL and Key-Value Databases: Concepts and Redis Overview
Briefing Document: NoSQL and Key-Value Databases
Source: Excerpts from "05 - NoSQL Intro + KV DBs.pdf" by Mark Fontenot, PhD, Northeastern University.
Date: October 26, 2023
Overview: This document provides a briefing on NoSQL databases, with a specific focus on Key-Value (KV) stores. It covers fundamental concepts like concurrency control (ACID vs. Optimistic), the definition and evolution of NoSQL, the CAP Theorem and its implications for distributed systems, the BASE alternative to ACID, and a detailed exploration of KV databases including their characteristics, use cases, and the popular KV store, Redis.
Main Themes and Important Ideas/Facts:
1. Concurrency Control in Databases:

- **ACID (Atomicity, Consistency, Isolation, Durability):** Traditional relational databases often rely on ACID properties to ensure data safety. ACID employs a **pessimistic concurrency model**, assuming conflicts are likely.
  - "**considered a pessimistic concurrency model because it assumes one transaction has to protect itself from other transactions**"
  - Conflicts are prevented by **locking resources** (read and write locks) until a transaction is complete.
  - The "Write Lock Analogy" of borrowing a library book illustrates this: "***If you have it, no one else can.***"
- **Optimistic Concurrency:** This model assumes conflicts are unlikely. Transactions do not acquire locks during read or write operations.
  - "**Optimistic because it assumes conflicts are unlikely to occur**"
  - It uses mechanisms like **last update timestamps and version numbers** to detect conflicts at the end of a transaction. If a conflict is detected, the transaction can be rolled back and retried.
  - Optimistic concurrency is suitable for **low conflict systems** (backups, analytical databases) and **read-heavy systems**, as it allows for higher concurrency. However, it can be less efficient in **high conflict systems** due to frequent rollbacks.

2. Introduction to NoSQL Databases:

- The term "NoSQL" was initially used in 1998 to describe a relational database system that did not use SQL.
- The more common, modern understanding of "NoSQL" is "**Not Only SQL**," although it is often thought of as non-relational databases.

- The development of NoSQL databases was partly a response to the challenges of processing **unstructured web-based data**.

3. CAP Theorem:

- The CAP Theorem states that a distributed database system can satisfy at most two of the following three guarantees:
  - **Consistency:** "**Every user of the DB has an identical view of the data at any given instant**" (Note: this differs from ACID Consistency).
  - **Availability:** "**In the event of a failure, the database system remains operational**" (every request gets a response).
  - **Partition Tolerance:** "**The database can maintain operations in the event of the network's failing between two segments of the distributed system**".
- Different NoSQL systems often prioritize different pairs of these guarantees, leading to trade-offs.
  - **CA:** Consistency and Availability. Systems always respond with the latest data, but may not handle network partitions well.
  - **CP:** Consistency and Partition Tolerance. If the system responds, it's with the latest data, but requests might be dropped during partitions.
  - **AP:** Availability and Partition Tolerance. The system always responds based on the distributed store, but the data might not be the absolute latest.

4. BASE as an Alternative to ACID in Distributed Systems:

- For distributed systems, many NoSQL databases adopt the **BASE** properties:
  - **Basically Available:** "**Guarantees the availability of the data (per CAP), but response can be "failure"/"unreliable" because the data is in an inconsistent or changing state**". The system appears to work most of the time.
  - **Soft State:** "**The state of the system could change over time, even w/o input. Changes could be result of eventual consistency. Data stores don't have to be write-consistent. Replicas don't have to be mutually consistent**".
  - **Eventual Consistency:** "**The system will eventually become consistent**". All writes will eventually propagate to all nodes/replicas.

5. Key-Value (KV) Databases:

-

Core Concept: KV stores are based on a simple **key = value** data model.

- **Design Principles:** They are designed around:
  - **Simplicity:** "**the data model is extremely simple**". This leads to simple CRUD operations and API creation.
  - **Speed:** "**usually deployed as in-memory DB**". Retrieving a value by its key is typically an **O(1) operation** due to the use of hash tables or similar data structures. They avoid complex queries or joins.
  - **Scalability:** "**Horizontal Scaling is simple - add more nodes**". They typically prioritize eventual consistency in distributed environments. "***Typically concerned with eventual consistency, meaning in a distributed environment, the only guarantee is that all nodes will eventually converge on the same value.***"
- **Use Cases:**
  - EDA/Experimentation Results Store
  - Feature Store (for low-latency retrieval)
  - Model Monitoring
  - Storing Session Information: "**everything about the current session can be stored via a single PUT or POST and retrieved with a single GET …. VERY Fast**"
  - User Profiles & Preferences
  - Shopping Cart Data
  - Caching Layer (in front of disk-based databases)

6. Redis: A Popular Key-Value Store:

- Redis (Remote Directory Server) is an **open-source, in-memory database**.
- It's sometimes called a **data structure store** as it supports various data structures beyond simple key-value pairs.
- While primarily a KV store, it can also be used with other models like Graph, Spatial, Full Text Search, Vector, and Time Series.
- **Durability:** Despite being in-memory, Redis supports data persistence through:
  -

    - Saving snapshots to disk at specific intervals.
    - An append-only file (AOF) that logs every write operation for roll-forward recovery.
- **Performance:** Can achieve high throughput (e.g., "> 100,000 SET ops / second").
- **Limitations:** Does not handle complex data or support secondary indexes. Only supports lookup by Key.
- **Data Types:** Redis supports various value data types associated with keys:
    - Strings (text, serialized objects, binary arrays)
    - Lists (linked lists)
    - Sets (unique unsorted strings)
    - Sorted Sets
    - Hashes (string to string maps)
    - Geospatial data
    - JSON
- **Database Organization:** Redis provides **16 databases by default**, numbered 0 to 15, with no other naming convention.
- **Interaction:** Interaction with Redis is through a set of commands for setting and getting key-value pairs. Many language libraries are also available.
- **Security Note:** Exposing the Redis port (6379) in a production environment without proper security (like a password) is a major security vulnerability.

7. Redis Data Type Details and Commands:

- **Strings:** Simplest type, used for caching, configuration, token management, and counting. Basic commands include `SET`, `GET`, `EXISTS`, `DEL`, `KEYS`, `SELECT`, `INCR`, `INCRBY`, `DECR`, `DECRBY`, and `SETNX`.
- **Hashes:** Value is a collection of field-value pairs, useful for representing basic objects, managing session information, and tracking user/event data. Commands include `HSET`, `HGET`, `HGETALL`, `HMGET`, and `HINCRBY`.
-

Lists: Value is a linked list of strings, suitable for implementing stacks, queues, message passing, logging, and social media feeds. Queue operations use `LPUSH` (add to front) and `RPOP` (remove from back). Stack operations use `LPUSH` and `LPOP` (remove from front). Other commands include `LLEN` and `LRANGE`.

•

JSON: Full support for JSON with JSONPath syntax for navigation. Stored internally as a binary tree for fast access.

•

Sets: Unordered collection of unique strings, useful for tracking unique items, representing relations, access control, and social network features. Supports set operations. Commands include `SADD`, `SISMEMBER`, `SCARD`, `SINTER` (intersection), `SDIFF` (difference), `SREM`, and `SRANDMEMBER`.

This briefing document highlights the key concepts related to NoSQL databases and provides a detailed overview of Key-Value stores, particularly focusing on the characteristics, benefits, and practical applications of Redis.