NoSQL Technology Overview

Sheetal Gangakhedkar

Class: Big Data: Tools, Concepts and Deployment

Instructor: Alakh K. Verma

March 12, 2017

Introduction

Applications from 1970s to 1990s mostly included dealing with structured data like online transactions, for example mostly stored in RDBMS systems, any business analysis or report generation involved using this structured data. These RDBMS systems are mostly used to store highly structured data records (accounts, products, purchases, bank transactions,…) and for business process automation using this data. These transaction processing systems accelerated the traditional manual record keeping tasks and reduced the probability of errors. Central computing infrastructure like mainframes, were used for these RDBMS systems because of the hardware costs associated with large memory, faster CPU, and greater storage components. To scale-out for a slow growing user base, these central computing systems were beefed-up with greater computing hardware and storage hardware.

Today's interactive software systems deal with dynamic user growth and user churn in millions, and handle huge amounts of unstructured and semi-structured data, in terabytes and petabytes generated by these users via interactions with Web applications. These systems are changing the world of communications, shopping, advertising, entertainment and relationship management. Distributed computing came to rescue and is allowing us to handle and process BigData, that is characterized by high volume, high velocity and large variety. These distributed processing systems, especially the cluster computing architectures use low-cost commodity hardware, while the memory, storage and computing components are getting cheaper. Users are flocking on these online services, with their personal devices, like PCs, Macs, Laptops, mobile phones and tablets generating large volumes of interactions data. To scale-out for a dynamic and rapidly growing users and connected devices, more low-cost commodity web servers (or nodes) are added to the cluster, and by distributing data and processing load across the nodes of a cluster. This increases system's fault-tolerance, and availability.

Keeping RDBMS Relevant for Cloud Computing

Relational database technology, is in some regards, is the last domino to fall in the inevitable march toward a fully-distributed software architecture. While a number of bandaids have extended the useful life of the technology (horizontal and vertical sharding, distributed caching and data denormalization), these tactics nullify key benefits of the relational model while increasing total system cost and complexity (*NoSQL Database Technology*. March 2011) [1].

Horizontal and Vertical Sharding

If the data for an application will not fit on a single server or, more likely, if a single server is incapable of maintaining the I/O throughput required to serve many users simultaneously, then a tactic known as sharding is frequently employed. In this approach an application will implement some form of data partitioning to manually spread data across servers. While this does work to spread the load, there are undesirable consequences to the approach.

1. When you fill a shard, it is highly disruptive to re-shard. When you fill a shard, you have to change the sharding strategy in the application itself.

2. You lose some of the most important benefits of the relational model. You can't do "joins" across shards
3. If you have new information you want to collect, you must modify the database schema on every server, then normalize, retune and rebuild the tables.

Data Denormalization

To support concurrency and sharding, data is frequently stored in a denormalized form when an RDBMS is used behind Web applications. At the limit the relational schema is more or less abandoned entirely, with data simply stored in key-value form, where a primary key is paired with a data "blob" that can hold any data. This approach allows the type of information being stored in the database to change without requiring an update to the schema. It makes sharding much easier and allows for rapid changes in the data model. Of course, just about all relational database functionality is lost. (*NoSQL Database Technology*. March 2011) [1]

Distributed Caching

Another tactic used to extend the useful scope of RDBMS technology has been to employ distributed caching technologies, such as Memcached. Most new Web applications now build Memcached into their data architecture from day one. Memcached "sits in front" of an RDBMS system, caching recently accessed data in memory and storing that data across any number of servers or virtual machines. Memcached architectures have problems of their own like: Accelerates only data reads, and write can be lost if the caching node is powered off. Cold cache thrash can happen with enough cache misses the RDBMS can be flooded with read requests.

Advent of NoSQL for Distributed Computing Systems

NoSQL stands for Not-only SQL, is designed for distributed data stores for large scale data storing needs. NoSQL database management systems share a common set of characteristics:

- No schema required - Data can be inserted in a NoSQL database without first defining a rigid database schema.
- Auto-sharding (sometimes called "elasticity") - A NoSQL database automatically spreads data across servers, without requiring applications to participate. Most NoSQL databases also support data replication, storing multiple copies of data across the cluster, and even across data centers, to ensure high availability and support disaster recovery.
- Distributed query support - NoSQL database systems retain their full query expressive power even when distributed across hundreds or thousands of servers.
- Integrated caching - To reduce latency and increase sustained data throughput, advanced NoSQL database technologies transparently cache data in system memory.

CAP Theorem and BASE System

Traditional RDBMS systems follow the ACID rule - Atomic, Consistent, Isolated and Durable. Because of the distributed data store nature of NoSQL databases, it cannot follow the ACID rule, the CAP theorem provides the reason for this non-compliance, by defining three basic requirements which exist in a special relation when designing applications for a distributed architecture. (*NoSQL. 2017)* [3].

- Consistency (C) - database remains consistent after the execution of an operation
- Availability (A) - the system/service is always available, no downtime
- Partition Tolerance (P) - the system continues to function even when the communication among the servers is unreliable.

The theorem asserts that it is impossible to fulfill all the three requirements in a distributed system. NoSQL databases follow different combinations of the C, A, P from the CAP theorem (*NoSQL. 2017)* [3]:

- CA - In a single site cluster, when a partition occurs the system blocks.
- CP - Some data may not be available, but the rest is still consistent.
- AP - System is still available under partitioning, but some of the data returned may not be consistent.

Basically, it implies that given one requirement is satisfied, then the solution has to make a choice between the other two requirements.

A BASE system gives up on Consistency (*NoSQL. 2017)* [3].

- **B**asically **A**vailable indicates that the system does guarantee availability, in terms of CAP theorem.
- **S**oft state indicates that the state of the system may change over time.
- **E**ventual consistency indicates that the system will become consistent over time.

NoSQL Data Models

NoSQL databases are generally implemented around one of the four data modeling concepts. (Pramod, 2014)[4]

- **Key-Value store** - Given a key query the database for the associated value, the database has no knowledge of what is in that value, basically it is schema-less data object.
- **Document store** - A document is a key value collection, usually represented in JSON. Document objects have implicit schema, and hence flexible and easy to change, and can be a store for different kinds of data in a complex data model. You can retrieve or update portions of the document by querying the database using the document keys and values.
- **Column store** - For each row key, the database can store multiple column families, each column family is a combination of columns that fit together. Each column has a key and value. To access the key-value, you have to specify the row-key and the column-family by name. This database allows you work with complex and rich data model.
- **Graph store** - Follows a Node and Edges structure, and provides flexibility in changing the relations between the nodes. Relational DBs have a hierarchical data-model which is one form of a Graph representation, but it is not easy to change the relations after the database is created. Graph DB is capable of elegantly representing any kind of data in a highly accessible way. Each node represents an entity and each edge represents a relationship between the

connected nodes. Every node and edge is defined by a unique identifier. Each node knows its adjacent nodes. As the nodes increase, the cost of local step remains the same. Also the database supports indexing for fast lookups.

NoSQL Implementations

Oracle NoSQL Database comes with an Administration Service and two major components: a Client Driver and a collection of Storage Nodes. The client driver implements the partition map and the RGST, while storage nodes implement the replication nodes comprising replication groups. A Storage Node Agent (SNA) runs on each storage node, monitoring that node’s behavior. The Administration Service, in addition to facilitating configuration changes, also collects and maintains performance statistics and logs important system events from SNA, providing online monitoring and input to performance tuning. (Oracle NoSQL Database. September 2011) [2].

Oracle NoSQL Database leverages the Oracle Berkeley DB Java Edition High Availability storage engine to provide distributed, highly-available key/value storage for large-volume, latency-sensitive applications or web services. It can also provide fast, reliable, distributed storage to applications that need to integrate with ETL processing. In its simplest form, Oracle NoSQL Database implements a map from user-defined keys (Strings) to opaque data items. It records version numbers for key/data pairs, but maintains the single latest version in the store, accomplished by using single-master replication; the master node always has the most up-to-date value for a given key, while read-only replicas might have slightly older versions. Applications can use version numbers to ensure consistency for read-modify-write operations.(Oracle NoSQL Database. September 2011) [2]

Common NoSQL databases based on Data Model

- Column-family: Cassandra, Apache HBase
- Key-Value: Redis, Project Voldemort, Riak, DynamoDB, Oracle NoSQL.
- Graph: Neo4j
- Document: CouchDB, RavenDB, MongoDB.

Choosing NoSQL Database

How do we choose which NoSQL database? here are some general guidelines (Pramod, 2014)[4]:

- **Key-value databases** - generally useful for storing session information, user profiles, preferences, shopping cart data.
- **Document databases** - are generally useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications.
- **Column family databases** are generally useful for content management systems, blogging platforms, maintaining counters, expiring usage, heavy write volume such as log aggregation.
- **Graph databases** are very well suited to problem spaces where we have connected data, such as social networks, spatial data, routing information for goods and money, recommendation engines.

References

[1] *NoSQL Database Technology*. (March 2011). Couchbase, Mountain View, CA, USA.

[2] Oracle NoSQL Database. (September 2011). Oracle, Redwood Shores, CA, USA.

[3] *NoSQL*. Retrieved on March 11, 2017 from http://www.w3resource.com/mongodb/nosql.php.

[4] Pramod Sadalage. (2 Oct 2014). NoSQL Databases Overview. Retrieved on March 11, 2017 from https://www.thoughtworks.com/insights/blog/nosql-databases-overview.