Data Manipulation at Scale

CISC-525

Philip A Grim II

Abstract

Relational databases have dominated the data storage landscape for decades, but in recent years, new data storage techniques based on clusters of commodity computers and schema-less, dynamic data models have made great strides. These new data stores, which have come to be known as NoSQL databases, offer some advantages in scalability and flexibility that make them a viable alternative to RDBMSs despite the tradeoffs in data consistency that they require.

Keywords

NoSQL Database

Distributed Database

NoSQL Distributed Database Model

ACID

CAP Theorem

Contents

1. Introduction

For many years, when one mentioned databases, the assumption was that one was speaking of relational databases. Relational database management systems (RDBMS) have been the cornerstone of data storage for decades. In the last few years, however, the proliferation of big data applications and analytics have made clustered computing more and more popular. Relational database systems do not lend themselves well to clustered computing; even though some RDBMS offerings have a clustered component, such as Oracle Real Application Cluster (RAC), they still rely on big hardware and a shared storage array. (Sadalage & Fowler, 2013)

What has evolved is a family of database implementations that are more suitable to clustered computing, defined as a number of interconnected commodity-grade compute nodes. These implementations differ in a number of ways, but all share the characteristics of being schema-less and not relying on Structured Query Language for data definition and manipulation. These databases have come to be referred to as NoSQL databases.

2. NoSQL Distributed Databases and Models

The family of NoSQL databases consists of four data storage models, three of which lend themselves well to distributed database implementations. The four models include key-value stores, document stores, column-family stores, and graph databases. Of these, graph databases are better suited to single-node implementations much like relational databases. (Sadalage & Fowler, 2013)

Key-Value Stores are the simplest form of a NoSQL database. As the name implies, data is stored in the form of a key and an associated value. (MongoDB, 2014) The values in a key-value store can be a single item of data, or can be a more complex structure, sometimes referred to as an aggregate. (Sadalage & Fowler, 2013) These aggregates could be serialized Java classes, JavaScript Object Notation (JSON) objects, or just about anything else (subject to limitations imposed by the implementation, such as maximum size of a value). This gives an advantage over the relational model in that an aggregate data structure can be stored and retrieved as a single value rather than as a complex join over normalized relations. One disadvantage of this model is that aggregates stored as values are opaque to the database system, that is, values inside the aggregate object cannot be queried – the key is the only thing that can be queried, and the entire aggregate is the unit that is retrieved. (Sadalage & Fowler, 2013) Some key-value stores do allow for metadata about values to be stored and queried, mitigating this disadvantage somewhat. (MongoDB, 2014)

Some examples of key-value stores include Riak, Redis, BerkeleyDB, and Aerospike. (NoSQL Archive, 2014)

Document databases store a key and a document, which can be a complex data structure similar to an aggregate. The main difference between a document store and a key-value store with a complex value is that in a document store, the database system has visibility into the values in the document and can query over those values. Partial returns are possible, rather than the full aggregate return in a key-value store. (Sadalage & Fowler, 2013)

Some examples of document stores include MongoDB, CouchDB, and MarkLogic. (NoSQL Archive, 2014)

Column-family data stores, exemplified by Google's Bigtable, provide a more structured approach to data storage. The data store is characterized as a multi-dimensional sorted map indexed by a row key, a column key, and a timestamp. The values in the map are uninterpreted arrays of bytes. (Chang, et al., 2006) Groups of columns are stored as column families, and these families are physically stored for efficient retrieval. Although column-family data stores have the constructs of tables and columns, the schema is not fixed – each row in a table can have entirely different columns from every other row. (Sadalage & Fowler, 2013)

Some examples of column-family stores include HBase, Accumulo, and Cassandra. (NoSQL Archive, 2014)

3. Consistency, Availability, and Partitioning

One of the characteristics of relational databases is data integrity and consistency enforcement, commonly referred to by the acronym ACID (Atomic, Consistent, Isolated, and Durable). Relational databases enforce data consistency using transactions – many rows in many relations can be inserted, updated, or deleted in one atomic transaction that takes effect when the transaction is committed. Other clients are not able to update the same data at the same time, nor to read an incomplete update. In general, distributed NoSQL databases do not support atomic transactions in this manner, so do not conform to the ACID specification (with the exception of graph databases). (Sadalage & Fowler, 2013)

Distributed databases use two main techniques to provide high performance, high availability and resist network partitioning – sharding and replication. Sharding occurs when a data store is divided across multiple servers in the cluster. Each shard contains part of the key space, and a master server maintains an index of what keys are assigned to what shards. This allows reads and writes to be spread across multiple nodes and increases performance. Replication is the duplication of data files across multiple nodes, such that there are several copies of each data file in the cluster. If a node becomes unavailable, one of the replicas can provide the same data. Replication comes in two flavors, master-slave and peer-to-peer. In master-slave replication, one replica is the master and all writes occur in that replica, to be later replicated to the slave replicas. In peer-to-peer replication, writes can occur to any replica and will eventually be propagated to the other replicas. (Sadalage & Fowler, 2013)

If a distributed database is sharded but not replicated, then a network partitioning, that is, the loss of connectivity between nodes such that the cluster is effectively divided into two separate parts, can cause some amount of data to be unavailable from the point of view of each partition. For this reason, most distributed databases use some combination of sharding and replication to defend against network partitioning.

Distributed NoSQL databases approach consistency from a different perspective from RDBMSs, that of "eventual consistency." Since it would be impractical (if not impossible) to support atomic transactions across multiple records perhaps stored on many different nodes in a cluster, no attempt at record locking is made. Replicated data stores take time to bring each replica into a consistent state, which can lead to inconsistent reads from different replicas. In many cases, this can be tolerated, although it can lead to application errors. Careful application design is necessary to manage the possibility of inconsistency. (Sadalage & Fowler, 2013)

The CAP theorem states that out of Consistency, Availability, and Partition Tolerance, only two of the three are possible at any one time. Sadalage and Fowler make the argument that the CAP theorem is not so black and white when it comes to NoSQL databases – instead, each characteristic requires a bit of tradeoff between the other two. A data store that is highly partition tolerant may need to sacrifice consistency for high availability, or may need to limit availability to maintain consistency, but a spectrum of possibility exists between the two. (Sadalage & Fowler, 2013)

4. Descriptive and Predictive Applications of NoSQL Databases

While NoSQL databases may not be best suited to Online Transaction Processing (OLTP) applications that require absolute data integrity and ACID compliance, there are many applications where distributed NoSQL databases can be very useful. In the context of retail sales, a distributed NoSQL database makes a fine platform for processing online ordering and sales. In such a case, there can be a high volume of traffic registering new customers, authenticating existing customers, maintaining a shopping cart, and processing a checkout.

Consider the following entity-relationship diagram for customer and sales management:

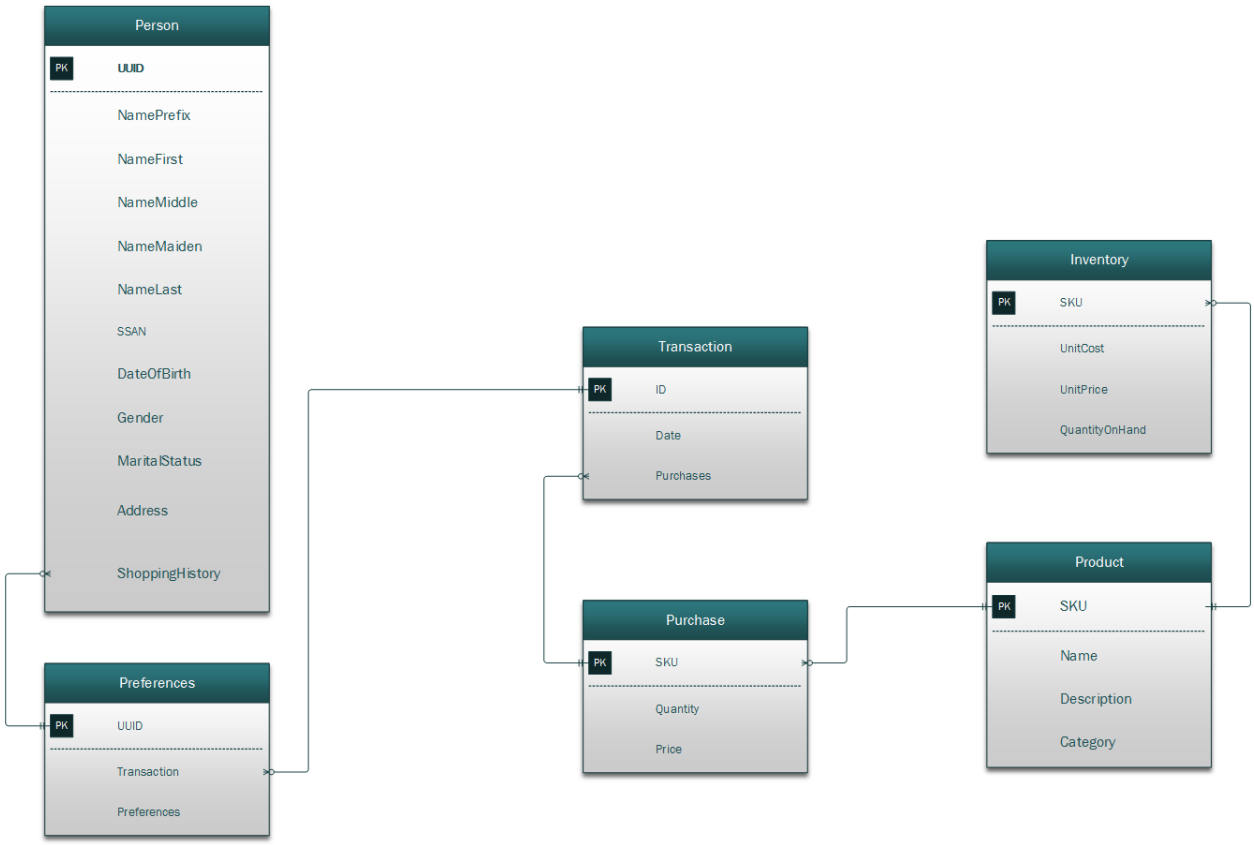

Figure 1Point of Sale ERD

With some slight modification, this same data can be easily represented in a column-family model in a data store such as Apache HBase. Rather than relations between tables, customer data can be

stored in one column family and transaction data in another column family. This would allow for easy retrieval of customer data, and easy association of transactions with a customer without the need for joins, as all data pertaining to that customer and his/her purchases would be maintained in one row in the datastore. Similarly, product and inventory data can be stored in a separate table in related column families.

Since most distributed NoSQL column family stores (HBase, Accumulo, Cassandra, Bigtable) provide the MapReduce programming framework as a method of iterating over stored data, many types of descriptive and predictive analytics can be performed over this data. Such applications as market-basket analysis, customer demographic analysis, just-in-time inventory, and many other applications are possible over very large volumes of data.

5. Conclusion

Although relational databases will always be the go-to solution for reliable, consistent data storage, NoSQL data stores are becoming more and more useful for scalable, highly available storage of unstructured, dynamic data. NoSQL databases provide some advantages in the modeling of data in a way that more closely resembles the way applications use data without the rigid structure and complex joining logic of relational databases. (Sadalage & Fowler, 2013)

NoSQL databases require some tradeoffs in data consistency and availability, but provide lateral scaling at a much lower cost than the vertical scaling of monolithic relational database systems. The data models provided by NoSQL databases, and the MapReduce programming framework supported by many NoSQL implementations, lend themselves to massively scaled analytics over large volumes of dynamic data.

References

Aerospike. (2104). *What Is A NoSQL Key-Value Store*. Retrieved from Aerospike: http://www.aerospike.com/what-is-a-nosql-key-value-store/

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., . . . Gruber, R. E. (2006). *Bigtable: A Distributed Storage System for Structured Data.* Retrieved from Google Research: http://static.googleusercontent.com/media/research.google.com/en/us/archive/bigtable-osdi06.pdf

MongoDB. (2014). *NoSQL Databases Explained*. Retrieved from MongoDB Web Site: http://www.mongodb.com/learn/nosql

NoSQL Archive. (2014). *NoSQL Databases*. Retrieved from NoSQL Archive Web Site: http://nosql-database.org/

Sadalage, P. J., & Fowler, M. (2013). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Upper Saddle River, NJ: Pearson Education, Inc.