IBM Analytics
White paper

April 2016

Why NoSQL?

Your database options in the new non-relational world

Contents

New types of apps are generating new types of data

The continual increase in web, mobile and IoT applications, alongside emerging trends shifting online consumer behavior and new classes of data, is causing developers to reevaluate how their data is stored and managed. Today's applications require a database that is capable of providing a scalable, flexible solution to efficiently and safely manage the massive flow of data to and from a global user base.

Developers and IT alike are finding it difficult, and sometimes even impossible, to quickly incorporate all of this data into the relational model while dynamically scaling to maintain the performance levels users demand. This is causing many to look at NoSQL databases for the flexibility they offer, and is a big reason why the global NoSQL market is forecasted to nearly double and reach USD3.4 billion in 2020[1].

A brief history of NoSQL

NoSQL, also known as *not only SQL* or *non-relational* databases, were specifically introduced to handle the rise in data types, data access, and data availability needs brought on by the dot-com boom.

Going back 20 years or so, when application architects and developers needed a data store for their applications, they were choosing among various relational databases. In fact, relational databases have been the *de facto* choice since the 1970s, as they have been the sole option available for both developers (because SQL is taught in computer science programs everywhere) and infrastructure teams (because of the well-understood tooling and predictable operational characteristics of RDBMSs). Some of the most popular are Oracle, MySQL, SQL Server and IBM® DB2®.

However, when Internet applications and companies started exploding during the late 90s to early 2000s, applications went from serving thousands of internal employees within companies to having millions of users on the public Internet. For these applications, performance and availability were paramount. The new problem of high availability at large scale drove companies like Google, Facebook and Amazon to create new technologies. Thankfully, they documented their efforts, released white papers, and open sourced their technology for the Internet community to continue building upon. By the late 2000s, several new non-relational database technologies had emerged, and NoSQL was the name that stuck to describe them all.

NoSQL's roots in open source

Many NoSQL databases have roots in the open source community. This heritage has been fundamental for their ever-increasing popularity and usage. You will often see companies that provide a commercial version of a database with services and support for the technology, while at the same time participating in the communities of their open source counterparts. Examples of these relationships include Datastax for Apache Cassandra, IBM Cloudant® for Apache CouchDB, and even MongoDB with an open source version of its MongoDB software.

Types of NoSQL databases

NoSQL is simply the term used to describe a family of databases that are all non-relational. It is generally agreed that there are four types of NoSQL databases, and the technologies, data types and uses cases vary wildly among them:

Key-value stores

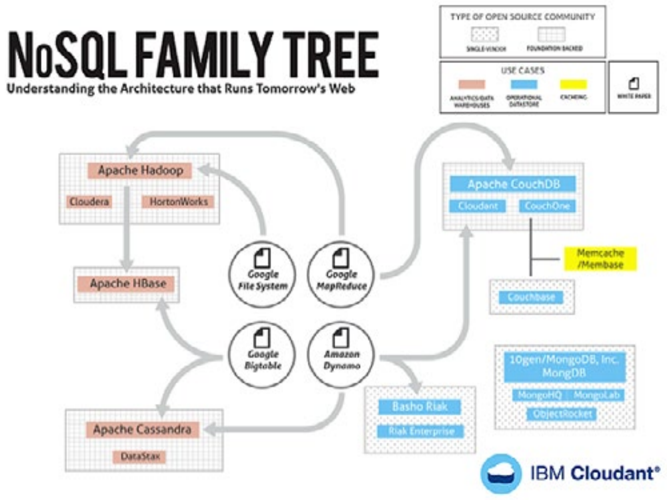
These databases pair keys to values, like a hash table in computer programming. A good analogy for a key-value store is a file system. The path acts as the key and the contents act as the file. There are often no "fields" to update; rather, the entire value other than the key must be updated if changes are to be made. This simplicity scales well; however, it can limit the complexity of queries and other advanced features available in key-value stores. *Examples of key-value stores include MemcacheD, Redis, etcd and Riak.*

Graph stores

Graph data stores excel at dealing with interconnected data. Graph databases consist of connections (called *edges* in graph DB terms) between nodes. Both nodes and their edges can store additional properties, such as key-value pairs. The strength of a graph database is in traversing the connections between nodes. Graph databases, however, generally require all data to fit on one machine, limiting their scalability. *Examples of graph stores include Neo4j and Titan, and an open source technology that is a de facto platform for graphs is Apache TinkerPop.*

Column stores

Relational databases store all of the data in a particular table's row together on disk, which makes retrieval of a particular row fast. Column-family databases generally serialize all the values of a particular column together on disk, which makes retrieval



of aggregated data on a specific attribute fast. This is valuable in data warehousing and analytics scenarios where you might run range queries over a specific value. *Examples of column stores include Hbase and Cassandra.*

Document stores

Document databases store records as *documents*, where a document can generally be thought of as a grouping of key-value pairs. Keys are always strings and values can be stored as strings, numerics, booleans, arrays and other nested, key-value pairs. Values can be nested to arbitrary depths. Popular syntaxes for document DBs include XML and JavaScript Object Notation (JSON). *Examples of document stores include MongoDB, CouchDB, Cloudant and MarkLogic.*

Why consider NoSQL?

The various NoSQL databases available today differ quite a bit, but there are common threads uniting them: flexibility, scalability, availability, lower costs and special capabilities.

Flexibility

Schema flexibility and intuitive data structures are key features that attract developers working in agile development cycles, and most NoSQL databases accommodate these qualities.

Schema updates and their associated downtime are generally not required, unlike in a relational database. NoSQL's flexibility is popular for supporting agile development practices by eliminating the complexity of database schema changes to support changing codebases.

Scalability

Scale refers to both the size of the data as well as the concurrent users acting on that data. NoSQL databases are typically more specialized to various use cases involving scale and can be much simpler for developing related application functions than relational databases.

Many NoSQL databases are built to scale horizontally and spread their data across a cluster of servers more easily than their relational counterparts. Relational database performance suffers when queries execute JOINs across shards, whereas a NoSQL database can avoid JOINs altogether and maintain high performance.

Availability

Downtime results in lost revenue, lost customers, and frustrated users. Thankfully, some NoSQL databases offer new or different replication architectures that can make different types of NoSQL databases more highly available for writes in addition to reads. This means that if one or more database servers, or *nodes*, goes down, the other nodes in the system are able to continue operations without data loss.

Lower operational cost

The open source roots of NoSQL make its entry point an obvious incentive, and it is common to hear of NoSQL adopters cutting significant costs versus their existing databases while still receiving the same or better performance and functionality. Historically, large relational databases have run on expensive machines and mainframes. The distributed nature of NoSQL databases means that they can be deployed and operated on clusters of servers in cloud architectures, equating to lower upfront hardware ependitures.

The Appeal of NoSQL JSON document stores

- JSON schema can be evolved rapidly without the need for intervention
- It only contains six kinds of values, so it is easy to implement and use
- JSON is self-describing and easy to understand
- It helps enable performance and scalability gains

Example: NoSQL JSON document store versus a RDBMS

Records in a NoSQL document store may be fully denormalized into individual JSON documents (store a record and all its related data together) versus the RDBMS approach of decomposing relations into separate tables (separate records into logical tables to better enforce consistency).

The benefit here is that by using NoSQL, you can get and put objects at once without JOINing data from separate tables. The JOINs of a relational database are absolute scalability killers for clustered databases.

Specialized capabilities

Not all NoSQL databases are created equal. To offer incentive and added integration to their users, many NoSQL providers include various specialized capabilities. Examples include specific indexing and querying capabilities for geospatial data, integrated full-text indexing for search, automatic data replication and synchronization features and application-friendly RESTful web APIs.

Summary

The database you pick for your next application matters now more than ever. Thankfully, you don't have to pick just one. Today's applications are expected to run non-stop and must efficiently manage continuously growing amounts of multi-structured data in order to do so. This has caused NoSQL to grow from a buzzword to a serious consideration for every database, from small shops to the enterprise.

While NoSQL databases share some common qualities, such as being non-relational and generally easy to scale, there are many unique differentiators to consider when deciding upon the correct NoSQL solution for your unique data needs. The right NoSQL database can act as a viable alternative to relational databases or can be utilized in a complementary fashion along with existing systems.

The requirement for efficient data delivery is our future, and you should consider NoSQL technology when planning any application development project that involves scale, different varieties of data or large amounts of potential users.

Getting started with IBM Cloud Data Services

IBM Cloud Data Services provides developers, data science professionals, and analytic architects and with a broad portfolio of composable, integrated data services covering content, data and analytics. The open, self-service Cloud Data Service offerings speed up time to market and improve uptime and deliver higher value, backed by IBM technology leadership. For information about how IBM Cloud Data Services is changing the way services are created for and delivered to developers, follow us on Twitter at @IBMcloudant, and visit **http://ibm.biz/clouddataservices**.

1 www.marketresearchmedia.com/?p=568

KUW12354-USEN-02