A MongoDB White Paper

Top 5 Considerations When Evaluating NoSQL Databases

February 2015

mongoDB

Table of Contents

Introduction

Relational databases have a long-standing position in most organizations, and for good reason. Relational databases underpin legacy applications that meet current business needs; they are supported by an extensive ecosystem of tools; and there is a large pool of labor qualified to implement and maintain these systems.

But companies are increasingly considering alternatives to legacy relational infrastructure. In some cases the motivation is technical — such as a need to scale or perform beyond the capabilities of their existing systems — while in other cases companies are driven by the desire to identify viable alternatives to expensive proprietary software. A third motivation is agility or speed of development, as companies look to adapt to the market more quickly and embrace agile development methodologies.

These drivers apply both to analytical and transactional applications. Companies are shifting workloads to Hadoop for their offline, analytical workloads, and they are building online, operational applications with a new class of data management technologies called "NoSQL", or "Not Only SQL", such as MongoDB.

Development teams have a strong say in the technology selection process. This community tends to find that the relational data model is not well aligned with the needs of their applications. Consider:

- Developers are working with new data types — structured, semi-structured, unstructured and polymorphic data — and massive volumes of it.
- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, even every day.
- Object-oriented programming is the norm, and developers need a data model that aligns with this approach, that is easy to use and that provides flexibility.
- Organizations are now turning to scale-out architectures using commodity servers and cloud computing instead of large monolithic architectures. NoSQL systems share several key characteristics. When compared to relational databases,

NoSQL systems are more scalable and provide superior performance. As companies evaluate NoSQL products, they should consider 5 critical dimensions to make the right choice for their applications and their businesses. In this Introduction 2 paper, we describe these dimensions and show why MongoDB is the most widely used NoSQL database in the market.

Data Model

The primary way in which NoSQL databases differ from relational databases is the data model. Although there are arguably dozens of NoSQL databases, they primarily fall into one of the following three categories:

Document Model

Whereas relational databases store data in rows and columns, document databases store data in documents. These documents typically use a structure that is like JSON (JavaScript Object Notation), a format popular among developers. Documents provide an intuitive and natural way to model data that is closely aligned with object-oriented programming — each document is effectively an object. Documents contain one or more fields, where each field contains a typed value, such as a string, date, binary or array. Rather than spreading out a record across multiple columns and tables, each record and its associated data are typically stored together in a single document. This simplifies data access and reduces or even eliminates the need for joins and complex transactions.

In a document database, the notion of a schema is dynamic: each document can contain different fields. This flexibility can be particularly helpful for modeling unstructured and polymorphic data. It also makes it easier to evolve an application during development, such as adding new fields. Additionally, document databases generally provide the query robustness that developers have come to expect from relational databases. In particular, data can be queried based on any fields in a document.

Applications: Document databases are general purpose, useful for a wide variety of applications due to the flexibility of the data model, the ability to query on any field and the natural mapping of the document data model to objects in modern programming languages.

Examples: MongoDB and CouchDB.

Graph Model

Graph databases use graph structures with nodes, edges and properties to represent data. In essence, data is modeled as a network of relationships between specific elements. While the graph model may be counter-intuitive and takes some time to understand, it can be used broadly for a number of applications. Its main appeal is that it makes it easier to model relationships between entities in an application.

Applications: Graph databases are useful in cases where relationships are core to the application, like social networks.

Examples: Neo4j and HyperGraphDB.

Key-Value and Wide Column Models

From a data model perspective, key-value stores are the most basic type of NoSQL database. Every item in the database is stored as an attribute name, or key, together with its value. The value, however, is entirely opaque to the system; data can only be queried by the key. This model can be useful for representing polymorphic and unstructured data, as the database does not enforce a set schema across key-value pairs.

Wide column stores, or column family stores, use a sparse, distributed multi-dimensional sorted map to store data. Each record can vary in the number of columns that are stored, and columns can be nested inside other columns called super columns. Columns can be grouped together for access in column families, or columns can be spread across multiple column families. Data is retrieved by primary key per column family.

Applications: Key value stores and wide column stores are useful for a narrow set of applications that only query data by a single key value. The appeal of these systems is their performance and scalability, which can be highly optimized due to the simplicity of the data access patterns.

Examples: Riak and Redis (Key-Value); HBase and Cassandra (Wide Column).

TAKEAWAYS

- All of these data models provide schema flexibility.
- The key-value and wide-column data model is opaque in the system - only the primary key can be queried.
- The document data model has the broadest applicability.
- The document data model is the most natural and most productive because it maps directly to objects in modern object-oriented languages.
- The wide column model provides more granular access to data than the key value model, but less flexibility than the document data model.

Query Model

Each application has its own query requirements. In some cases, it may be acceptable to have a very basic query model in which the application only accesses records based on a primary key. For most applications, however, it is important to have the ability to query based on several different values in each record. For instance, an application that stores data about customers may need to look up not only specific customers, but also specific companies, or customers by a certain deal size, or aggregations of customer types by zip code or state.

It is also common for applications to update records, including one or more individual fields. To satisfy these requirements, the database needs to be able to query data based on secondary indexes. In these cases, a document database may be the most appropriate solution.

Document Database

Document databases provide the ability to query on any field within a document. Some products, such as MongoDB, provide a rich set of indexing options to optimize a wide variety of queries, including compound indexes, sparse indexes, time to live (TTL) indexes, unique indexes, text indexes, geospatial indexes and others.

Furthermore, some of these products provide the ability to analyze data in place. MongoDB, for instance, provides both the Aggregation Framework for providing real-time analytics (along the lines of the SQL GROUP BY functionality), and a native MapReduce implementation for sophisticated analyses. Regarding updates, MongoDB provides find and modify capabilities so that values in documents can be updated in a single statement to the database rather than making multiple round trips.

Graph Database

These systems tend to provide rich query models where simple and complex relationships can be interrogated to make direct and indirect inferences about the data in the system. Relationship-type analysis tends to be very efficient in these systems, whereas other types of analysis may be less optimal.

Key Value and Wide Column Databases

These systems provide the ability to retrieve and update data based only on a primary key. For querying on other values, users are encouraged to maintain their own indexes. Some products provide limited support for secondary indexes, but with several caveats. To perform an update in these systems, two round trips may be necessary: first find the record, then update it. In these systems, the update may be implemented as a complete rewrite of the record whether a few bytes have changed or the entire record.

TAKEAWAYS

- The biggest difference between NoSQL systems lies in the ability to query data efficiently.
- Document databases provide the richest query functionality, which allows them to address a wide variety of applications.
- Key-value stores and wide column stores provide a single means of accessing data: by primary key. They offer very limited query functionality and may impose additional development costs and application-level requirements to provide more than the most basic query features.

Consistency Model

NoSQL systems typically maintain multiple copies of the data for availability and scalability purposes. In these architectures, there different guarantees regarding the consistency of the data across copies. NoSQL systems tend to be consistent or eventually consistent.

With a consistent system, writes by the application are immediately visible in subsequent queries. With an eventually consistent system writes are not immediately visible. As an example, when reflecting inventory levels for products in a product catalog, with a consistent system each query will see the current inventory as inventory levels are updated by the application, whereas with an eventually consistent system the inventory levels may not be accurate for a query at a given time, but will eventually become accurate. For this reason application code tends to be somewhat different for eventually consistent systems - rather than updating the inventory by taking the current inventory and subtracting one, for example, developers are encouraged to issue idempotent queries that explicitly set the inventory level.

TAKEAWAYS

- Most applications and development teams expect consistent systems.
- Different consistency models pose different trade-offs for applications in the areas of consistency and availability.
- MongoDB provides tunable consistency, defined at the query level.
- Eventually consistent systems provide some advantages for writes at the cost of making reads and updates more complex.

Consistent Systems

Each application has different requirements for data consistency. For many applications, it is imperative that the data be consistent at all times. As development teams have worked under a model of consistency with relational databases for decades, this approach is more natural and familiar. In other cases, eventual consistency is an acceptable trade-off for the flexibility it allows in the system's availability.

Document databases and graph databases can be consistent or eventually consistent. MongoDB provides tunable consistency. By default, data is consistent — all writes and reads go to the primary copy of the data. As an option, read queries can be issued against secondary copies where data is eventually consistent; the consistency choice is made at the query level.

Eventually Consistent Systems

With eventually consistent systems, there is a period of time in which all copies of the data are not synchronized. This may be acceptable for read-only applications and data stores that do not change often, like historical archives. By the same token, it may also be appropriate for high-write use cases in which the database is capturing information like logs, which will only be read at a later point in time. Key-value and wide column stores are typically eventually consistent.

Eventually consistent systems must be able to accommodate conflicting updates in individual records. Because writes can be applied to any copy of the data, it is possible and not uncommon for writes to conflict with one another. Some systems like Riak use vector clocks to determine the ordering of events and to ensure that the most recent operation wins in the case of a conflict. Other systems like CouchDB retain all conflicting values and allow the user to resolve the conflict. Another approach, followed by Cassandra, is simply to assume the greatest value is the correct one. For these reasons, writes tend to perform well in eventually consistent systems, but updates can involve trade-offs that complicate the application significantly.

APIs

There is no standard for interfacing with NoSQL systems. Each system presents different designs and capabilities for application development teams. The maturity of the API can have major implications for the time and cost required to develop and maintain the underlying NoSQL system.

Idiomatic Drivers

There are a number of popular programming languages, and each provides different paradigms for working with data and services. Idiomatic drivers are created by development teams that are experts in the given language and that know how programmers 5 prefer to work within that language. This approach can also benefit from its ability to leverage specific features in a programming language that might provide efficiencies for accessing and processing data.

For programmers, idiomatic drivers are easier to learn and use, and they reduce the onboarding time for teams to begin working with the underlying system. For example, idiomatic drivers provide direct interfaces to set and get documents or fields within documents. With other types of interfaces it may be necessary to retrieve and parse entire documents and navigate to specific values in order to set or get a field.

MongoDB supports idiomatic drivers in over a dozen languages: Java, .NET, Ruby, Node.js, Perl, Python, PHP, C, C++, Erlang, Javascript, Haskell and Scala. Other drivers are supported by the community.

Thrift or RESTful APIs

Some systems provide RESTful interfaces. This approach has the appeal of simplicity and familiarity, but it relies on the inherent latencies associated with HTTP. It also shifts the burden of building an interface to the developers; and this interface is likely to be inconsistent with the rest of their programming interfaces. Similarly, some systems provide a Thrift interface, a very low level paradigm that shifts the burden to developers to develop more abstract interfaces within their applications.

Pluggable Storage API

A pluggable storage API allows organizations to choose alternative storage engines optimized for different workloads. These APIs should be designed to support all the native features of the database so that teams leveraging an alternative hardware architecture are not forced to compromise on other features.

MongoDB 3.0 introduces a new storage engine API and two supported storage engines, both of which can coexist within a single MongoDB replica set, making it easy to evaluate and migrate between them:

- The default MMAPv1 engine, an improved version of the engine used in prior MongoDB releases.
- The new WiredTiger storage engine. For many applications, WiredTiger's more granular concurrency control and native compression will provide significant benefits in the areas of lower storage costs, greater hardware utilization, and more predictable performance.

Users can leverage the same MongoDB query language, data model, scaling, security and operational tooling across different applications, each powered by different pluggable MongoDB storage engines. Other engines are under development by MongoDB and members of the MongoDB ecosystem.

TAKEAWAYS

- The maturity and functionality of APIs vary significantly across NoSQL products.
- MongoDB's idiomatic drivers minimize onboarding time for new developers and simplify application development.
- Through the use of a pluggable storage architecture, MongoDB can be extended with new capabilities, and configured for optimal use of specific hardware architectures.

Commercial Support and Community Strength

Choosing a database is a major investment. Once an application has been built on a given database, it is costly and challenging to migrate it to a different database. Companies usually invest in a small number of core technologies so they can develop expertise, integrations and best practices that can be amortized across many projects. NoSQL systems are relatively new, and while there are many options in the market, a small number of products will stand the test of time.

Commercial Support

Users should consider the health of the company or project when evaluating a database. It is important not only that the product continues to exist, but also to evolve and to provide new features. Having a strong, experienced support organization capable of providing services globally is another relevant consideration.

TAKEAWAYS

- Commercial backing and support is an important part of evaluating NoSQL products.
- MongoDB has the largest commercial backing; the largest and most active community; support teams in New York, Palo Alto, Dublin, and Sydney; and extensive documentation.

Community Strength

There are significant advantages of having a strong community around a technology, particularly databases. A database with a strong community of users makes it easier to find and hire developers that are familiar with the product. It makes it easier to find information, documentation and code samples. It also helps organizations retain key technical talent. Lastly, a strong community encourages other technology vendors to develop integrations and to participate in the ecosystem.

Conclusion

As the technology landscape evolves, organizations increasingly find the need to evaluate new databases to support changing application and business requirements. The media hype around NoSQL databases and the commensurate lack of clarity in the market makes it important for organizations to understand the differences between the available solutions. As discussed in this paper, key criteria to consider when evaluating these technologies are the data model, query model, consistency model and APIs, as well as commercial support and community strength. Many organizations find that document databases such as MongoDB are best suited to meet these criteria, though we encourage technology decision makers to evaluate these considerations for themselves.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than a third of the Fortune 100. We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Management Service (MMS) is the easiest way to run MongoDB in the cloud. It makes MongoDB the system you worry about the least and like managing the most.

Production Support helps keep your system up and running and gives you peace of mind. MongoDB engineers help you with production issues and any aspect of your project.

Development Support helps you get up and running quickly. It gives you a complete package of software and services for the early stages of your project.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.org)
MongoDB Enterprise Download (mongodb.com/download)