# NoSQL for Scale, Agility and ROI

www.mongodb.com

# NoSQL for Scale, Agility and ROI

www.mongodb.com

## What is NoSQL

NoSQL encompasses a wide variety of different database technologies and were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs. **NoSQL's predecessor, relational databases, were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of cheap storage and processing power that can be leveraged via NoSQL**.

## NoSQL Database Types

**Since "NoSQL" just means non-relational and not SQL, there are many different ways to implement NoSQL technology. Generally, NoSQL databases include the following designs:**

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- **Graph stores** are used to store information about networks, such as social connections. Graph stores include Neo4J and HyperGraphDB.
- **Key-value stores** are the

simplest NoSQL databases. Every single item in the database is stored as an attribute name, or key, together with its value. Examples of key-value stores are Riak and Voldemort. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.

- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

NoSQL systems share several key characteristics. When compared to relational databases, NoSQL systems are more scalable and provide superior

performance. The NoSQL data model addresses several issues that relational model is not designed to address:

* Large volumes of structured, semi-structured, and unstructured data
* Agile sprints, quick iteration, and frequent code pushes
* Object-oriented programming that is easy to use and flexible
* Efficient, scale-out architecture over large, expensive monolithic architecture

**Dynamic Schemas**

Relational databases require that schemas be defined before you can add data. For example, you might want to store data about your customers such as phone numbers, first and last name,

address, city and state – a SQL database needs to know this in advance.

This fits poorly with agile development approaches, because each time you complete new features, the schema of your database often needs to change. So if you decide, a few iterations into development, that you'd like to store customers' favorite items in addition to their addresses and phone numbers, you'll need to add that column to the database, and then migrate the entire database to the new schema.

If the database is large, this is a very slow process that involves significant downtime. If you are frequently changing the data your application stores – because you are iterating rapidly – this downtime may also be frequent. There's also no

way, using a relational database, to effectively address data that's completely unstructured or unknown in advance.

NoSQL databases are built to allow the insertion of data without a predefined schema. That makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed.

**Auto-sharding, replication, and integrated caching**

Because of the way they are structured, relational databases usually scale vertically – a single server has to

host the entire database to ensure reliability and continuous availability of data. This gets expensive quickly, places limits on scale, and creates a relatively small number of failure points for database infrastructure. The solution is to scale horizontally, by adding servers instead of concentrating more capacity in a single server. "Sharding" a database across many server instances can be achieved with SQL databases, but usually is accomplished through SANs and other complex arrangements for making hardware act as a single server. *NoSQL databases, on the other hand, usually support auto-sharding, meaning that they natively and automatically spread data across an arbitrary number of servers, without requiring the application to even*

*be aware of the composition of the server pool*. Data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.

Cloud computing makes this significantly easier, with providers such as Amazon Web Services providing virtually unlimited capacity on demand, and taking care of all the necessary database administration tasks. Developers no longer need to construct complex, expensive platforms to support their applications, and can concentrate on writing application code. Commodity servers can provide the same processing and storage capabilities as a single high-end server for a fraction of the price.

Most NoSQL databases also support automatic replication, meaning that you get high availability and disaster recovery without involving separate applications to manage these tasks. The storage environment is essentially virtualized from the developer's perspective.

Many NoSQL database technologies have excellent integrated caching capabilities, keeping frequently-used data in system memory as much as possible and removing the need for a separate caching layer that must be maintained.

When relational databases were introduced into the 1970s, data schemas were fairly simple and straightforward, and it made sense to conceive objects as sets of relationships. For example, an article object might be related to a

category (an object), a tag (another object), a comment (another object), and so on. Because relationships between different types of data were specified in the database schema, these relational databases could be queried with a standard Structured Query Language, or SQL. *But the environment for data, as well as programming, has changed since the development of the SQL database:*

- **The emergence of cloud computing** has brought deployment and storage costs down dramatically, but only if data can be spread across multiple servers easily without disruption. In a complex SQL database, this is difficult because many queries require multiple large

tables to be joined together to provide a response. Executing distributed joins is a very complex problem in relational databases.

- **The need to store unstructured data**, such as social media posts and multimedia, has grown rapidly. SQL databases are extremely efficient at storing structured information, and workarounds or compromises are necessary for storing and querying unstructured data.
- **Agile development methods** mean that the database schema needs to change rapidly as demands evolve. SQL databases require their structure to be specified in advance, which means any changes to the

information schema require time-consuming ALTER statements to be run on a table.

In response to these changes, new ways of storing data (e.g. NoSQL databases) have emerged that allow data to be grouped together more naturally and logically, and that loosen the restrictions on database schema. One of the most popular ways of storing data is a document data model, where each record and its associated data is thought of as a “document”. In a document database, such as MongoDB, everything related to a database object is encapsulated together. Storing data in this way has the following advantages:

- **Documents are independent units** which makes performance better (related data is read contiguously off disk) and makes it easier to distribute data across multiple servers while preserving its locality.
- **Application logic is easier to write.** *You don't have to translate between objects in your application and SQL queries, you can just turn the object model directly into a document*.
- **Unstructured data can be stored easily**, since a document contains whatever keys and values the application logic requires. In addition, costly migrations are avoided since the database does not

need to know its information schema in advance.

Document databases generally have very powerful query engines and indexing features that make it easy and fast to execute many different optimized queries. **The strength of a document database's query language is an important differentiator between these databases**.

Relational databases were never designed to cope with the scale and agility challenges that face modern applications – and aren't built to take advantage of cheap storage and processing power that's available today through the cloud. Relational database vendors have developed two main

technical approaches to address these shortcomings:

**manual sharding**

Tables are broken up into smaller physical tables and spread across multiple servers. Because the database does not provide this ability natively, development teams take on the work of deploying multiple relational databases across a number of machines. Data is stored in each database instance autonomously. Application code is developed to distribute the data, distribute queries, and aggregate the results of data across all of the database instances. Additional code must be developed to handle resource failures, to perform joins across the

different databases, for data rebalancing, replication, and other requirements. Furthermore, many benefits of the relational database, such as transactional integrity, are compromised or eliminated when employing manual sharding.

**distributed cache**

A number of products provide a caching tier for database systems. These systems can improve read performance substantially, but they do not improve write performance, and they add complexity to system deployments. If your application is dominated by reads then a distributed cache should probably be considered, but if your application is dominated by writes or if you have a

relatively even mix of reads and writes, then a distributed cache may not improve the overall experience of your end users.

NoSQL databases have emerged in response to these challenges and in response to the new opportunities provided by low-cost commodity hardware and cloud-based deployment environments - and natively support the modern application deployment environment, reducing the need for developers to maintain separate caching layers or write and maintain sharding code.

**Comparison: NoSQL vs. SQL**

| | **SQL Databases** | **NoSQL Databases** |
|---|---|---|

| | | |
|---|---|---|
| Types | One type (SQL database) with minor variations | Many different types including key-value stores, document databases, wide-column stores, and graph databases |
| Development | Developed in 1970s to deal with first | Developed in 2000s to deal with limitations of SQL databases, particularly |

| | | |
|---|---|---|
| History | wave of data storage applications | concerning scale, replication and unstructured data storage |
| Examples | MySQL, Postgres, Oracle Database | **MongoDB, Cassandra, HBase, Neo4j** |
| | Individual records (e.g., "employees") are stored as rows in tables, with each column storing a | Varies based on NoSQL database type |

| | | |
|---|---|---|
| | specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. | For example, key-value stores function similarly to SQL databases, bu have only two columns ("key" and "value"), with |
| Data Storage Model | Separate data types are stored in separate tables, and then joined together when more complex queries are | more complex information sometimes stored within the "value" columns. Document |

executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables

databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically

| | | |
|---|---|---|
| | together to get all the information necessary. | |
| Schemas | Structure and data types are fixed in advance. To store information about a new data item, the entire database must be | Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some |

| | |
|---|---|
| altered, during which time the database must be taken offline. | databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically. |
| Vertically, meaning a single server must be made increasingly powerful in order to deal with | Horizontally, meaning that to add capacity, a database administrator can simply |

| | | |
|---|---|---|
| Scaling | increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required. | add more commodity servers or cloud instances. The NoSQL database automatically spreads data across servers as necessary |
| Development Model | Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle | Open-source |

| | | |
|---|---|---|
| | Database) | |
| Supports Transactions | Yes, updates can be configured to complete entirely or not at all | In certain circumstances and at certain levels (e.g., document level vs. database level) |
| Data Manipulation | Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE… | **Through object-oriented APIs** |

| | | |
|---|---|---|
| Consistency | Can be configured for strong consistency | Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra) |

MongoDB is the leading NoSQL database. Designed for how we build and run applications today, MongoDB (named from *humongous*," meaning "extremely large") empowers organizations to be more agile and

scalable. It enables new types of applications, better customer experience, faster time to market and lower costs for organizations of all sizes.

MongoDB is built for scalability, performance and high availability. Auto-sharding allows MongoDB to scale from single server deployments to large, complex multi-data center architectures. Leveraging native caching and RAM, MongoDB provides high performance for both reads and writes. Built-in replication with automated failover enables enterprise-grade reliability and operational flexibility. MongoDB also provides native, idiomatic drivers for all popular programming languages and frameworks to make development natural.

**Leading organizations relying on MongoDB:**

Choosing a Big Data technology is a strategic and long-term investment. Given that technologies with large and growing communities like Linux tend to endure and foster the best third-party application ecosystems, organizations evaluating Big Data projects should consider which technologies have the most momentum and adoption. MongoDB is the fastest-growing community in Big Data based on a wide array of metrics. And while MongoDB is not the largest Big Data community by every measure, MongoDB's large base and meteoric trajectory position it as essential Big Data infrastructure for years

to come.

**Nearly 3x as many Google searches as the next most prevalent NoSQL technology**

**Skill adoption greater than that of the next 3 NoSQL databases combined on LinkedIn**

**Served as the #1 big data source overall in 2012 according to Jaspersoft's Big Data Index**

"Big Data" describes data sets so large and complex they are impractical to manage with traditional software tools. It relates to data creation, storage, retrieval and analysis that is remarkable in terms

of volume, velocity, and variety. Big Data means new opportunities for organizations to create business value — and extract it. Enterprises can save money, grow revenue, and achieve many other business objectives, in any vertical.

The MongoDB NoSQL database can underpin many Big Data systems, not only as a real-time, operational data store but in offline capacities as well. With MongoDB, organizations are serving more data, more users, more insight with greater ease — and creating more value worldwide.

Read more...

# Big Data Explained

The three pillars of big data are volume, velocity, and variety.

- **Volume.** A typical PC might have had 10 gigabytes of storage in 2000. Today, Facebook ingests 500 terabytes of new data every day; a Boeing 737 will generate 240 terabytes of flight data during a single flight across the US; the proliferation of smart phones, the data they create and consume; sensors embedded into everyday objects will soon result in billions of new, constantly-updated data feeds containing environmental, location, and other information, including

video.

- **Velocity.** Clickstreams and ad impressions capture user behavior at millions of events per second; high-frequency stock trading algorithms reflect market changes within microseconds; machine to machine processes exchange data between billions of devices; infrastructure and sensors generate massive log data in real-time; on-line gaming systems support millions of concurrent users, each producing multiple inputs per second.
- **Variety.** Big Data data isn't just numbers, dates, and strings. Big Data is also geospatial data, 3D data, audio and video, and unstructured text, including log files and social

media. Traditional database systems were designed to address smaller volumes of structured data, fewer updates or a predictable, consistent data structure. Traditional database systems are also designed to operate on a single server, making increased capacity expensive and finite. As applications have evolved to serve large volumes of users, and as application development practices have become agile, the traditional use of the relational database has become a liability for many companies rather than an enabling factor in their business. Big Data technologies, such as MongoDB, solve these problems and provide companies with the means to create

tremendous business value.

## Big Data and the Enterprise

Big Data solutions are providing organizations across all verticals with new opportunities to increase ~~revenue, lower costs, and meet their unique business objectives. For example, organizations can:~~

- ~~**Build new applications:** Big data might allow a company to collect billions of real-time data points on its products, resources, or customers – and then repackage that data instantaneously to optimize customer experience or resource utilization. For example, a major US~~

city is using MongoDB to cut crime and improve municipal services by collecting and analyzing geospatial data in real-time from over 30 different departments.

- **Improve the effectiveness and lower the cost of existing applications:** Big data technologies can replace highly-customized, expensive legacy systems with a standard solution that runs on commodity hardware. And because many big data technologies are open source, they can be implemented far more cheaply than proprietary technologies. For example, by migrating its reference data management application to MongoDB, a Tier 1 bank

dramatically reduced the license and hardware costs associated with the proprietary relational database it previously ran, while also bringing its application into better compliance with regulatory requirements.

- **Realize new sources of competitive advantage:** Big data can help businesses act more nimbly, allowing them to adapt to changes faster than their competitors. For example, MongoDB allowed one of the largest Human Capital Management (HCM) solution providers to rapidly build mobile applications that integrated data from a wide variety of disparate sources.

- **Increase customer loyalty:** Increasing the amount of data shared within the organization – and the speed with which it is updated – allows businesses and other organizations to more rapidly and accurately respond to customer demand. For example, a top 5 global insurance provider, MetLife, used MongoDB to quickly consolidate customer information from over 70 different sources and provide it in a single, rapidly-updated view.

## Combining Operational and Analytical Technologies; Using Hadoop

New technologies like NoSQL, MPP databases, and Hadoop have emerged to

address Big Data challenges and to enable new types of products and services to be delivered by the business. One of the most common ways companies are leveraging the capabilities of both systems is by integrating a NoSQL database such as MongoDB with Hadoop. The connection is easily made by existing APIs and allows analysts and data scientists to perform complex, retroactive queries for analysis and insights while maintaining the efficiency and ease-of-use of a NoSQL database.

## MongoDB and Big Data

With MongoDB, organizations are serving more data, more users, more insight with greater ease — and creating

~~more value worldwide.~~

To learn more about the criteria you should consider when selecting a Big Data technology and see examples of how organizations are using Big Data solutions, read our free whitepaper *Big Data: Examples and Guidelines for the Enterprise Decision Maker*.

~~Cloud computing refers to a broad set of computing and software products that are sold as a service, managed by a 3rd-party provider and delivered over a network. Infrastructure-as-a-Service (IaaS) is a flavor of cloud computing in which on-demand processing, storage or network resources are provided to the customer. Sold on-demand with limited or no upfront investment for the end-user,~~

~~consumption is readily scalable to accommodate spikes in usage. MongoDB complements the horizontal scaling and agility afforded by cloud computing.~~

Read more...

## Cloud Computing Explained

Cloud computing refers to a broad set of computing and software products that are sold as a service, managed by a 3rd-party provider and delivered over a network. Infrastructure-as-a-Service (IaaS) is a flavor of cloud computing in which on-demand processing, storage or network resources are provided to the

customer. Sold on-demand with limited or no upfront investment for the end-user, consumption is readily scalable to accommodate spikes in usage. To realize full economies of scale and cost advantages, the provider delivers the services using a multi-tenant, virtualized platform. Customers pay only for the capacity that is actually used (like a utility), as opposed to self-hosting, where the user pays for system capacity it is are used or not.

As compared to self-hosting, IaaS is:

- **Inexpensive.** To self-host an application, one has to pay for enough resources to handle peak load on an application, at all times. Amazon discovered that before

launching its cloud offering it was using only about 10% of its server capacity the vast majority of the time [[^1]]. And because cloud providers make large investments in hardware, and use that hardware more effectively, they cover their costs more easily.

- **Tailored.**There’s no need to invest in the next-largest server to provide more resources for an application - bandwidth, processing and storage capability can be added in relatively small increments. Small applications can be run for very little cost by taking advantage of spare capacity.
- **Elastic.** Computing resources can easily be added and released as

needed. And because clouds are typically designed to host millions of applications and users, one can also add an effectively limitless amount of capacity, making it much easier to deal with unexpected traffic spikes.

- **Reliable.** With the cloud, it's easy and inexpensive to have servers in multiple geographic locations, allowing content to be served locally to users, and also allowing for better disaster recovery and business continuity. Clouds, and applications that run on them, are also built such that individual server instances can come in and out of existence and applications will continue to run. If the application is properly

configured, it will not go down unless the entire cloud goes down, too.

Overall, cloud computing provides better agility and scalability, together with lower costs and faster time to market. However, it does require that applications be engineered to take advantage of this new infrastructure; applications built for the cloud need to be able to scale by adding more servers, for example, instead of adding capacity to existing servers.

## MongoDB and Cloud Computing

MongoDB is built for the cloud. MongoDB's native scale-out architecture,

~~enabled by "sharding," aligns well with the horizontal scaling and agility afforded by cloud computing. Sharding automatically distributes data evenly across multi-node clusters and balances queries across them. In addition, MongoDB automatically manages sets of redundant servers, called "replica sets," to maintain availability and data integrity even if individual cloud instances are taken offline. To ensure high availability, for instance, users can spin up multiple members of a replica set as individual cloud instances across different availability zones and/or data centers.~~

Agile software development includes a set of software development methods focused on an iterative approach

to building software (*as opposed to software development methods that focus on rigorous planning and scheduling in advance*). Agile development means that tasks are broken into small pieces, and evaluated and built on as they are completed, with users and other stakeholders able to frequently see small, completed results. ~~NoSQL databases such as MongoDB support this paradigm, allowing business to achieve a faster time to market and improve developer productivity.~~

Read more...

## Agile Development Explained

Classically, the approach to software design and system development has been planned in advance, and done in a strict sequence. This has the advantage of predictably delivering a project on time, and letting stakeholders know in advance what to expect with minimal further input needed.

However, ultimately the wa**terfall method may result in more overh**ead, as initial specifications can't respond to findings as software is built - for example, a new need emerging, an unexpected interaction occurring, or system architecture changes that can be taken advantage of. Once a stage is done in the waterfall method, making future changes is very c*ostly, since it involves reevaluating the entire project and*

*signing off a new set of specifications or requirements before further work can proceed*.

Agile development does several things. It allows projects to adapt quickly, since changes are expected in advance. It minimizes risk, since users and other stakeholders are able to frequently evaluate results and change their desires. It also accelerates development, since superfluous requirements can be dropped early in the development cycle. Ideally, in an agile environment, even between official iterations the changes of many developers might be merged into the product and released nightly, or even continuously.

~~Facebook, for example, changes its codebase many times daily. It also~~

~~emphasizes keeping software working throughout the development process. Continuous attention to high-quality software, and usable design, helps ensure a higher-quality product. At regular intervals, the agile developer team can also tune and adjust its behavior to improve future work~~. It also allows easier completion of much more complex products, with smaller teams that function as independent cells. Having a launchable product at the end of each small work unit also makes sure that bugs are eliminated quickly and early, making future development more efficient, and increasing flexibility at design and implementation time.

But new products are necessary to take advantage of agile development - for

example, continuous integration tools, automated testing, and flexible databases. SQL databases, which require a schema defined upfront and subsequent (and costly) database migrations as schemas change, are more difficult to use with agile methods and impossible to use in a continuously integrated environment without significant additional engineering.

Databases that allow the storage of unstructured data, such as the NoSQL databases, support agile development by freeing developers from upfront and evolving schema specification.

**MongoDB and Agile Development**

MongoDB has dynamic schemas,

which can evolve as applications evolve without requiring expensive migrations. In addition, **MongoDB stores data using JSON-like documents, which map naturally to object-oriented programming.** MongoDB also includes automatic horizontal scaling, native replication and automated failover, rich queries with full index support, and more, delivering faster time to market and higher developer productivity.

## Selecting the Right Database

Selecting the right database for your application and goals is important. As you consider alternatives to legacy infrastructures, you may have several motivations: to scale or perform beyond

the capabilities of your existing system, identify viable alternatives to expensive proprietary software, or increase the speed and agility of development. When selecting the right database for your business and application, there are five important dimensions to consider.

**Read our free whitepaper *Top 5 Considerations When Evaluating NoSQL Databases* and learn about:**

- Selecting the appropriate data model: document, key-value & wide column, or graph model
- The pros and cons of consistent and eventually consistent systems
- Why idiomatic drivers minimize onboarding time for new developers

and simplify application development

Download Your Copy

# Disclaimer

This information was automatically retrieved on 2013-09-12T13:27:42+00:00 from:

http://www.mongodb.com/learn/nosql

and automatically parsed, clean up and interpreted by dotEPUB.com. It is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. This e-book is not an authoritative source: please, visit the original webpage.

Help improve the dotEPUB code and report issues at:

http://code.google.com/p/dotepub/

dotEPUB’s cleanup process is partially based on a modified old version of Readability (© by arc90).

(v. 0.8.8)