# NoSQL Databases and When to Use Them

## Cassandra, Hbase, Mongo-DB & Couchbase

---





---

## Contents

# 1. Introduction

The aim of this whitepaper is to investigate the suitability of NoSQL as a data source. NoSQL databases are a compelling alternative to relational databases because of their capability to store and process large amounts of loosely structured data at fast speeds. The cost effectiveness of such solutions and the proliferation of new data sources have led to widespread adoption of NoSQL

## 1.1. What is NoSQL

NoSQL databases were created to address the limitations of relational database management systems. NoSQL seeks to break away from the traditional structure of relational databases, and enable developers to implement models in ways that more closely fit the data flow needs of their system. This means that NoSQL databases can be implemented in ways that traditional relational databases could never be structured.A NoSQL database environment is, simply put, a non-relational and largely distributed database system that enables rapid, ad-hoc organization and analysis of extremely high-volume, disparate data types. NoSQL databases are sometimes referred to as cloud databases, non-relational databases, Big Data databases and a myriad of other terms and were developed in response to the sheer volume of data being generated, stored and analyzed by modern users (user-generated data) and their applications (machine-generated data).

In general, NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today's modern business applications. A very flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology.

From a business standpoint, considering a NoSQL or 'Big Data' environment has been shown to provide a clear competitive advantage in numerous industries. In the 'age of data', this is compelling information as a great saying about the importance of data is summed up with the following "if your data isn't growing then neither is your business".

## 2. Cassandra

Cassandra is a massively scalable open source NoSQL database. Cassandra is perfect for managing large amounts of structured, semi-structured, and unstructured data across multiple data centers and the cloud. Originally created for Facebook, Cassandra is designed to have peer-to-peer symmetric nodes, instead of master or named nodes, to ensure there can never be a single point of failure (SPoF). Cassandra automatically partitions data across all the nodes in the database cluster, but the administrator has the power to determine what data will be replicated and how many copies of the data will be created.

### 2.1. When should we use Cassandra?

Being a part of NoSQL family Cassandra offers solution for problem where your requirement is to have very heavy write system and you want to have quite responsive reporting system on top of that stored data. Consider use case of Web analytics where log data is stored for each request and you want to built analytical platform around it to count hits by hour, by browser, by IP, etc in real time manner. Cassandra Scales linearly with massive write.Cassandra can be integrated with Hadoop, Hive and Apache Spark for batch Processing.

### 2.2. Some more key features

- ★ Written in: Java
- ★ Main point: Store huge data sets in "almost" SQL
- ★ License: Apache
- ★ Protocol: CQL3 & Thrift
- ★ CQL3 is very similar SQL, but with some limitations that come from the scalability (most notably: no JOINs, no aggregate functions.)
- ★ CQL3 is now the official interface. Don't look at Thrift, unless you're working on a legacy app. This way, you can live without understanding Column Families, Super Columns, etc.
- ★ Querying by key, or key range (secondary indices are also available)
- ★ Tunable trade-offs for distribution and replication (N, R, W)

- ★ Writes can be much faster than reads (when reads are disk-bound).
- ★ Map/reduce possible with Apache Hadoop.
- ★ All nodes are similar, as opposed to Hadoop/HBase.
- ★ Very good and reliable cross-datacenter replication.
- ★ Distributed counter data type.
- ★ You can write triggers in Java.
- ★ Best used: When you need to store data so huge that it doesn't fit on server, but still want a friendly familiar interface to it.
- ★ Example: Web analytics, to count hits by hour, by browser, by IP, etc.

### 2.3. When should we not use Cassandra?

Cassandra is typically not the choice for transactional data that needs per-transaction commit/rollback capabilities. Note that Cassandra does have atomic transactional abilities on a per row/insert basis (but with no rollback capabilities).

## 3. Hbase

Hbase is a column-oriented database management system which runs on top of HDFS. About 45% of Hadoop users today are estimated to be using HBase. Hbase is not a relational data store, and it does not support structured query language like SQL.In Hbase, a master node regulates the cluster and region servers to store portions of the tables and operates the work on the data.

### 3.1 When should we use Hbase?

- ★ High capacity storage system
- ★ Distributed design to cater large tables
- ★ Column-Oriented Stores
- ★ Horizontally Scalable
- ★ High performance & Availability

## 3.2. Some more key features

★ Written in: Java

★ Main point: Billions of rows X millions of columns

★ License: Apache

★ Protocol: HTTP/REST (also Thrift)

★ Modeled after Google's BigTable

★ Uses Hadoop's HDFS as storage

★ Map/reduce with Hadoop

★ Query predicate push down via server side scan and get filters

★ Optimizations for real time queries.

★ A high performance Thrift gateway.

★ HTTP supports XML, Protobuf, and binary.

★ Jruby-based (JIRB) shell.

★ Rolling restart for configuration changes and minor upgrades.

★ Random access performance is like MySQL.

★ A cluster consists of several different types of nodes.

★ Best used: Hadoop is probably still the best way to run Map/Reduce jobs on huge datasets. Best if you use the Hadoop/HDFS stack already.

★ For example: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.

## 3.3. When should we not use Hbase?

★ New data only needs to be appended

★ Batch processing instead of random reads

★ Complicated access patterns (such as joins)

★ Full ANSI SQL support required

★ A single node can deal with the volume and the velocity of the complete data set

# 4. MongoDB

Mongo DB is a No SQL product supported by 10 gen.Mongo DB stores documents in “Collections”. Collections are analogous to tables. Mongo DB stores data in memory, so it is hugely memory hungry. For redundancy, data is stored to disk and this makes it a high IO system as well (depending on the load and the amount of data). MongoDB supports dynamic schema design, allowing the documents in a collection to have different fields and structures. The database uses a document storage and data interchange format called BSON, which provides a binary representation of JSON-like documents.

## 4.2. When should we use MongoDB?

★ MongoDB by default prefers high insert rate over transaction safety. If you need to load tons of data lines with a low business value for each one, MongoDB should fit.
★ If you need to partition and shard your database, MongoDB has a built in easy solution for that.
★ MongoDB has built in special functions, so finding relevant data from specific locations is fast and accurate.
★ If you don't have a DBA, and you don't want to normalize your data and do joins, you should consider MongoDB.

## 4.2. Some more key features

★ Written in: C++
★ Main point: Retains some friendly properties of SQL. (Query, index)
★ License: AGPL (Drivers: Apache)
★ Protocol: Custom, binary (BSON)
★ Master/slave replication (auto failover with replica sets)
★ Sharding built-in
★ Queries are javascript expressions
★ Run arbitrary javascript functions server-side
★ Uses memory mapped files for data storage

- ★ Performance over features
- ★ Journaling (with --journal) is best turned on
- ★ On 32bit systems, limited to ~2.5Gb
- ★ Text search integrated
- ★ GridFS to store big data + metadata (not actually an FS)
- ★ Has geospatial indexing
- ★ Data centre aware
- ★ Best used: If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks.
- ★ For example: For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

### 4.3. When should we not use MongoDB?

- ★ Multi-Object Transactions: MongoDB only supports ACID transactions for a single document.
- ★ SQL: SQL is well-known and a lot of people know how to write very complex queries to do lots of things. This knowledge is transferrable across a lot of implementations where MongoDB's queries language are specific to it.
- ★ Strong ACID guarantees: MongoDB allows for things like inconsistent reads which is fine in some applications, but not in all.
- ★ Traditional BI: A lot of very powerful tools exist that allow for OLAP and other strong BI applications and those run against traditional SQL database.

## 5. Couchbase

Couchbase is an open source, document-oriented database that has a flexible data model, is performant, is scalable, and is suitable for applications like the one in our use case that needs to shift its relational database data into a structured JSON document.

### 5.1. When should we use Couchbase?

- ★ Flexible Data Model
- ★ Easy Scalability
- ★ Consistent High Performance
- ★ Always Online
- ★ cross-datacenter replication
- ★ auto-failover

### 5.2. Some more key features

- ★ Written in: Erlang & C
- ★ Main point: Memcached compatible, but with persistence and clustering
- ★ License: Apache
- ★ Protocol: memcached + extensions
- ★ Very fast (200k+/sec) access of data by key
- ★ Persistence to disk
- ★ All nodes are identical (master-master replication)
- ★ Provides memcached-style in-memory caching buckets, too
- ★ Write de-duplication to reduce IO
- ★ Friendly cluster-management web GUI
- ★ Connection proxy for connection pooling and multiplexing (Moxi)
- ★ Incremental map/reduce
- ★ Cross-datacenter replication

★ Best used: Any application where low-latency data access, high
★ concurrency support and high availability is a requirement.
★ For example: Low-latency use-cases like ad targeting or highly-concurrent
★ web apps like online gaming (e.g. Zynga).

## 5.3. When should we not use Couchbase?

★ When you ask Couchbase to save a document for you, it immediately allocates it to RAM, and adds it to a write queue. On a simplistic level, the write queue churns through its list of documents and persists them to disk. Holding data in RAM and adding to the write queue has a number of advantages, such as the ability to perform fast writes and to query this data immediately. However, there is not currently any way in Couchbase to know if a document has been persisted to disk.
★ The other problem with this approach is that you may lose data if your system goes down whilst holding a large write queue. Whilst the amount of RAM you hold is important for good Couchbase performance, you need to ensure that disk I/O is quick enough to keep the write queue low. Generally, the more nodes you have, the better your I/O performance since increasing the number of servers will increase your overall I/O bandwidth. If your application needs high write rates, you may need to think about clustering a larger number of less powerful servers.
★ You will need to write a view (think of this as pulling together data from different sources) with a map function (in javascript) for every query you want to run in Couchbase.

# 6. Comparison

## 6.1. Comparison NOSQL DataBases

| Name | cassandra | Hbase | MongoDB | Couchbase |
|---|---|---|---|---|
| Description | Wide-column store based on ideas of BigTable and Dynamo DB | Wide-column store based on Apache Hadoop and on concepts of BigTable Database model Document store | One of the most popular document stores | JSON-based document store derived from CouchDB with a Memcached-compatible interface |
| Database model | Wide column store | Wide column store | Document store | Document store |
| Developer | Apache Software Foundation | Apache Software Foundation | MongoDB,inc | Couchbase, Inc. |
| License | Opensource | Opensource | Opensource | Opensource |
| Database as a Service (DBaaS) | No | No | No | No |

| Implementation language | Java | Java | C++ | C, C++, Go and Erlang |
|---|---|---|---|---|
| Server operating systems | BSD<br>Linux<br>OS X<br>Windows | Linux<br>Unix<br>Windows | Linux<br>OS X<br>Solaris<br>Windows | Linux<br>OS X<br>Windows |
| Typing | Yes | No | Yes | Yes |
| XML support | Yes | No | No | No |
| Secondary indexes | Yes | NO | YES | YES |
| SQL | No | No | No | No |
| Mapreduce | Yes | Yes | Yes | Yes |
| APIs and other access methods | Proprietary protocol | Java API<br>RESTful HTTP API<br>Thrif | proprietary protocol using JSON | Memcached protocol<br>RESTful HTTP API |
| Replication methods | This is due to cassandra is designed as a peer- to- peer | selectable replication factor | Master-slave replication | Master-master replication<br>Master-slave replication |
| User concepts | Access rights for users can be defined per object | Access Control Lists (ACL) | Access rights for users and roles | Memcached protocol<br>RESTful HTTP API |

Thank you