

Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase comparison

While SQL databases are insanely useful tools, their tyranny of ~15 years is coming to an end. And it was just time: I can't even count the things that were forced into relational databases, but never really fitted them.

But the differences between "NoSQL" databases are much bigger than it ever was between one SQL database and another. This means that it is a bigger responsibility on [software architects](#) to choose the appropriate one for a project right at the beginning.

In this light, here is a comparison of [Cassandra](#), [Mongodb](#), [CouchDB](#), [Redis](#), [Riak](#) and [HBase](#):

CouchDB

- **Written in:** Erlang
- **Main point:** DB consistency, ease of use
- **License:** Apache
- **Protocol:** HTTP/REST
- Bi-directional (!) replication,
- continuous or ad-hoc,
- with conflict detection,
- thus, master-master replication. (!)
- MVCC - write operations do not block reads
- Previous versions of documents are available
- Crash-only (reliable) design
- Needs compacting from time to time
- Views: embedded map/reduce
- Formatting views: lists & shows
- Server-side document validation possible
- Authentication possible
- Real-time updates via `_changes` (!)
- Attachment handling
- thus, [CouchApps](#) (standalone js apps)
- jQuery library included

Best used: For accumulating, occasionally changing data, on which

pre-defined queries are to be run. Places where versioning is important.

For example: CRM, CMS systems. Master-master replication is an especially interesting feature, allowing easy multi-site deployments.

Redis

- **Written in:** C/C++
- **Main point:** Blazing fast
- **License:** BSD
- **Protocol:** Telnet-like
- Disk-backed in-memory database,
- but since 2.0, it can swap to disk.
- Master-slave replication
- Simple keys and values,
- but [complex operations](#) like ZREVRANGEBYSCORE
- INCR & co (good for rate limiting or statistics)
- Has sets (also union/diff/inter)
- Has lists (also a queue; blocking pop)
- Has hashes (objects of multiple fields)
- Of all these databases, only Redis does transactions (!)
- Values can be set to expire (as in a cache)
- Sorted sets (high score table, good for range queries)
- Pub/Sub and WATCH on data changes (!)

Best used: For rapidly changing data with a foreseeable database size (should fit mostly in memory).

For example: Stock prices. Analytics. Real-time data collection. Real-time communication.

MongoDB

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

- Performance over features
- After crash, it needs to repair tables

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 all things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

Cassandra

- **Written in:** Java
- **Main point:** Best of BigTable and Dynamo
- **License:** Apache
- **Protocol:** Custom, binary (Thrift)
- Tunable trade-offs for distribution and replication (N, R, W)
- Querying by column, range of keys
- BigTable-like features: columns, column families
- Writes are much faster than reads (!)
- Map/reduce possible with Apache Hadoop
- I admit being a bit biased against it, because of the bloat and complexity it has partly because of Java (configuration, seeing exceptions, etc)

Best used: If you're in love with BigTable. :) When you write more than you read (logging). If every component of the system must be in Java. ("No one gets fired for choosing Apache's stuff.")

For example: Banking, financial industry

Riak

- **Written in:** Erlang & C, some Javascript
- **Main point:** Fault tolerance
- **License:** Apache
- **Protocol:** HTTP/REST
- Tunable trade-offs for distribution and replication (N, R, W)
- Pre- and post-commit hooks,
• for validation and security.
- Built-in full-text search
- Map/reduce in javascript or Erlang
- Comes in "open source" and "enterprise" editions

Best used: If you want something Cassandra-like (Dynamo-like), but no way

you're gonna deal with the bloat and complexity. If you need very good single-site scalability, availability and fault-tolerance, but you're ready to pay for multi-site replication.

For example: Point-of-sales data collection. Factory control systems. Places where even seconds of downtime hurt.

HBase

(With the help of ghshephard)

- **Written in:** Java
- **Main point:** Billions of rows X millions of columns
- **License:** Apache
- **Protocol:** HTTP/REST (also Thrift)
- Modeled after BigTable
- 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
- Cascading, hive, and pig source and sink modules
- Jruby-based (JIRB) shell
- No single point of failure
- Rolling restart for configuration changes and minor upgrades
- Random access performance is like MySQL

Best used: Use it when you need random, realtime read/write access to your Big Data.

For example: Facebook Messaging Database (more general example coming soon)

Of course, all systems have much more features than what's listed here. I only wanted to list the key points that I base my decisions on. Also, development of all are very fast, so things are bound to change. I'll do my best to keep this list updated.

-- Kristof