## cassandra-vs-mongodb-vs-couchdb-vs-redis

# Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Membase vs Neo4j comparison

While SQL databases are insanely useful tools, their monopoly 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](http://kkovacs.eu/resume) to choose the appropriate one for a project right at the beginning.

In this light, here is a comparison of [Cassandra](http://cassandra.apache.org/), [Mongodb](http://www.mongodb.org/), [CouchDB](http://couchdb.apache.org/), [Redis](http://redis.io/), [Riak](http://www.basho.com/Riak.html), [Membase](http://www.couchbase.org/membase), [Neo4j](http://neo4j.org/) and[HBase](http://hbase.apache.org/):

## 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 (auto failover with replica sets)
* Sharding built-in
* Queries are javascript expressions
* Run arbitrary javascript functions server-side
* Better update-in-place than CouchDB
* Uses memory mapped files for data storage
* Performance over features
* Journaling (with --journal) is best turned on
* On 32bit systems, limited to ~2.5Gb
* An empty database takes up 192Mb
* GridFS to store big data + metadata (not actually an FS)
* Has geospatial indexing

**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.

## Riak (V1.0)

* **Written in:** Erlang & C, some Javascript
* **Main point:** Fault tolerance
* **License:** Apache
* **Protocol:** HTTP/REST or custom binary
* Tunable trade-offs for distribution and replication (N, R, W)
* Pre- and post-commit hooks in JavaScript or Erlang, for validation and security.
* Map/reduce in JavaScript or Erlang
* Links & link walking: use it as a graph database
* Secondary indices: but only one at once
* Large object support (Luwak)
* Comes in "open source" and "enterprise" editions
* Full-text search, indexing, querying with Riak Search server (beta)
* In the process of migrating the storing backend from "Bitcask" to Google's "LevelDB"
* Masterless multi-site replication replication and SNMP monitoring are commercially licensed

**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. Could be used as a well-update-able web server.

## CouchDB (V1.1.1)

* **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](http://couchapp.org/) (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 (V2.4)

* **Written in:** C/C++
* **Main point:** Blazing fast
* **License:** BSD
* **Protocol:** Telnet-like
* Disk-backed in-memory database,
* Currently without disk-swap (VM and Diskstore were abandoned)
* Master-slave replication
* Simple values or hash tables by keys,
* but [complex operations](http://redis.io/commands) 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)
* Sorted sets (high score table, good for range queries)
* Redis has transactions (!)
* Values can be set to expire (as in a cache)
* Pub/Sub lets one implement messaging (!)

**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.

## HBase (V0.92.0)

* **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
* Cascading, hive, and pig source and sink modules
* 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:** Analysing log data.

## Neo4j (V1.5M02)

* **Written in:** Java
* **Main point:** Graph database - connected data
* **License:** GPL, some features AGPL/commercial
* **Protocol:** HTTP/REST (or embedding in Java)
* Standalone, or embeddable into Java applications
* Full ACID conformity (including durable data)
* Both nodes and relationships can have metadata
* Integrated pattern-matching-based query language ("Cypher")
* Also the "Gremlin" graph traversal language can be used
* Indexing of nodes and relationships
* Nice self-contained web admin
* Advanced path-finding with multiple algorithms
* Indexing of keys and relationships
* Optimized for reads
* Has transactions (in the Java API)
* Scriptable in Groovy
* Online backup, advanced monitoring and High Availability is AGPL/commercial licensed

**Best used:** For graph-style, rich or complex, interconnected data. Neo4j is quite different from the others in this sense.

**For example:** Social relations, public transport links, road maps, network topologies.

## 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
* Has secondary indices
* Writes are much faster than reads (!)
* Map/reduce possible with Apache Hadoop
* All nodes are similar, as opposed to Hadoop/HBase

**Best used:** 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 (though not necessarily for financial transactions, but these industries are much bigger than that.) Writes are faster than reads, so one natural niche is real time data analysis.

## Membase

* **Written in:** Erlang & C
* **Main point:** Memcache compatible, but with persistence and clustering
* **License:** Apache 2.0
* **Protocol:** memcached plus 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
* Very nice cluster-management web GUI
* Software upgrades without taking the DB offline
* Connection proxy for connection pooling and multiplexing (Moxi)

**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).

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