[Cassandra](http://cassandra.apache.org/), [Mongodb](http://www.mongodb.org/), , [Hypertable](http://hypertable.org/) and [HBase](http://hbase.apache.org/):

**The most popular ones**

MongoDB (2.2)

* **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
* Data center 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.

habiendo columnas predefinidas realmente te retiene.

**Clones of Google's Bigtable**

Cassandra (1.2)

* **Written in:** Java
* **Main point:** Best of BigTable and Dynamo
* **License:** Apache
* **Protocol:** Thrift & custom binary CQL3
* Tunable trade-offs for distribution and replication (N, R, W)
* Querying by column, range of keys (Requires indices on anything that you want to search on)
* BigTable-like features: columns, column families
* Can be used as a distributed hash-table, with an "SQL-like" language, CQL (but no JOIN!)
* Data can have expiration (set on INSERT)
* 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
* Cross-datacenter replication

**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 data analysis.

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
* 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.

## Hypertable (0.9.6.5)

* **Written in:** C++
* **Main point:** A faster, smaller HBase
* **License:** GPL 2.0
* **Protocol:** Thrift, C++ library, or HQL shell
* Implements Google's BigTable design
* Run on Hadoop's HDFS
* Uses its own, "SQL-like" language, HQL
* Can search by key, by cell, or for values in column families.
* Search can be limited to key/column ranges.
* Sponsored by Baidu
* Retains the last N historical values
* Tables are in namespaces
* Map/reduce with Hadoop

**Best used:** If you need a better HBase.

**For example:** Same as HBase, since it's basically a replacement: Search engines. Analysing log data. Any place where scanning huge, two-dimensional join-less tables are a requirement.

## CouchDB (V1.2)

* 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)

**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.