Kristof Kovacs
Software architect, consultant

Cassandra vs **MongoDB** vs **CouchDB** vs **Redis** vs **Riak** vs **HBase** vs **Couchbase** vs **OrientDB** vs **Aerospike** vs **Neo4j** vs **Hypertable** vs **ElasticSearch** vs **Accumulo** vs **VoltDB** vs **Scalaris** vs **RethinkDB** comparison

(Yes it's a long title, since people kept asking me to write about this and that too :) I do when it has a point.)

While SQL databases are insanely useful tools, their monopoly in the last decades is coming to an end. And it's just time: I can't even count the things that were forced into relational databases, but never really fitted them. (That being said, relational databases will always be the best for the stuff that has *relations*.)

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

In this light, here is a comparison of Open Source NOSQL databases Cassandra (http://cassandra.apache.org/), Mongodb (http://www.mongodb.org/), CouchDB (http://couchdb.apache.org/), Redis (http://redis.io/), Riak (http://basho.com/riak/), RethinkDB (http://rethinkdb.com/), Couchbase (ex-Membase) (http://www.couchbase.org/membase), Hypertable (http://hypertable.org), ElasticSearch (http://www.elasticsearch.org/), Accumulo (http://accumulo.apache.org/), VoltDB (http://voltdb.com/), Kyoto Tycoon (http://fallabs.com/kyototycoon/), Scalaris (https://code.google.com/p/scalaris/), OrientDB (http://www.orientechnologies.com/), Aerospike (http://www.aerospike.com/), Neo4j (http://neo4j.org/) and HBase (http://hbase.apache.org/):

The most popular ones

Redis (V3.2)

- **Written in:** C
- **Main point:** Blazing fast
- **License:** BSD
- **Protocol:** Telnet-like, binary safe
- Disk-backed in-memory database,
- Master-slave replication, automatic failover
- Simple values or data structures by keys
- but complex operations (http://redis.io/commands) like ZREVRANGEBYSCORE.
- INCR & co (good for rate limiting or statistics)
- Bit and bitfield operations (for example to implement bloom filters)
- 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)
- Lua scripting capabilities
- Has transactions
- Values can be set to expire (as in a cache)
- Pub/Sub lets you implement messaging
- GEO API to query by radius (!)

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

For example: To store real-time stock prices. Real-time analytics. Leaderboards. Real-time communication. And wherever you used memcached before.

Cassandra (2.0)

- **Written in:** Java
- **Main point:** Store *huge* datasets in "almost" SQL
- **License:** Apache
- **Protocol:** CQL3 & Thrift
- CQL3 is very similar to 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 ColumnFamilies, SuperColumns, etc.
- Querying by key, or key range (secondary indices are also available)

- Tunable trade-offs for distribution and replication (N, R, W)
- 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
- Very good and reliable cross-datacenter replication
- Distributed counter datatype.
- 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.

For example: Web analytics, to count hits by hour, by browser, by IP, etc. Transaction logging. Data collection from huge sensor arrays.

MongoDB (3.2)

- **Written in:** C++
- **Main point:** JSON document store
- **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
- Geospatial queries
- Multiple storage engines with different performance characteristics
- Performance over features
- Document validation
- Journaling
- Powerful aggregation framework
- On 32bit systems, limited to ~2.5Gb
- Text search integrated
- 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.

ElasticSearch (0.20.1)

- **Written in:** Java
- **Main point:** Advanced Search
- **License:** Apache
- **Protocol:** JSON over HTTP (Plugins: Thrift, memcached)
- Stores JSON documents
- Has versioning
- Parent and children documents
- Documents can time out
- Very versatile and sophisticated querying, scriptable
- Write consistency: one, quorum or all
- Sorting by score (!)
- Geo distance sorting
- Fuzzy searches (approximate date, etc) (!)
- Asynchronous replication
- Atomic, scripted updates (good for counters, etc)
- Can maintain automatic "stats groups" (good for debugging)

Best used: When you have objects with (flexible) fields, and you need "advanced search" functionality.

For example: A dating service that handles age difference, geographic location, tastes and dislikes, etc. Or a leaderboard system that depends on many variables.

Classic document and BigTable stores

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.

Accumulo (1.4)

- **Written in:** Java and C++
- **Main point:** A BigTable with Cell-level security
- **License:** Apache
- **Protocol:** Thrift
- Another BigTable clone, also runs of top of Hadoop
- Originally from the NSA
- Cell-level security
- Bigger rows than memory are allowed
- Keeps a memory map outside Java, in C++ STL
- Map/reduce using Hadoop's facitlities (ZooKeeper & co)
- Some server-side programming

Best used: If you need to restict access on the cell level.

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.

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.

Graph databases

OrientDB (2.0)

- **Written in:** Java
- **Main point:** Document-based graph database
- **License:** Apache 2.0
- **Protocol:** binary, HTTP REST/JSON, or Java API for embedding
- Has transactions, full ACID conformity
- Can be used both as a document and as a graph database (vertices with properties)
- Both nodes and relationships can have metadata
- Multi-master architecture
- Supports relationships between documents via persistent pointers (LINK, LINKSET, LINKMAP, LINKLIST field types)
- SQL-like query language (Note: no JOIN, but there are pointers)
- Web-based GUI (quite good-looking, self-contained)
- Inheritance between classes. Indexing of nodes and relationships
- User functions in SQL or JavaScript
- Sharding
- Advanced path-finding with multiple algorithms and Gremlin traversal language
- Advanced monitoring, online backups are commercially licensed

Best used: For graph-style, rich or complex, interconnected data.

For example: For searching routes in social relations, public transport links, road maps, or network topologies.

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
- Clustering, replication, caching, online backup, advanced monitoring and High Availability are commercially licensed

Best used: For graph-style, rich or complex, interconnected data.

For example: For searching routes in social relations, public transport links, road maps, or network topologies.

The "long tail"
(Not widely known, but definitely worthy ones)

Couchbase (ex-Membase) (2.0)

- **Written in:** Erlang & C
- **Main point:** Memcache 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).

Scalaris (0.5)

- **Written in:** Erlang
- **Main point:** Distributed P2P key-value store
- **License:** Apache
- **Protocol:** Proprietary & JSON-RPC
- In-memory (disk when using Tokyo Cabinet as a backend)
- Uses YAWS as a web server
- Has transactions (an adapted Paxos commit)
- Consistent, distributed write operations
- From CAP, values Consistency over Availability (in case of network partitioning, only the bigger partition works)

Best used: If you like Erlang and wanted to use Mnesia or DETS or ETS, but you need something that is accessible from more languages (and scales much better than ETS or DETS).

For example: In an Erlang-based system when you want to give access to the DB to Python, Ruby or Java programmers.

Aerospike (3.4.1)

- **Written in:** C
- **Main point:** Speed, SSD-optimized storage
- **License:** License: AGPL (Client: Apache)
- **Protocol:** Proprietary
- Cross-datacenter replication is commercially licensed
- Very fast access of data by key
- Uses SSD devices as a block device to store data (RAM + persistence also available)
- Automatic failover and automatic rebalancing of data when nodes or added or removed from cluster
- User Defined Functions in LUA
- Cluster management with Web GUI
- Has complex data types (lists and maps) as well as simple (integer, string, blob)
- Secondary indices
- Aggregation query model
- Data can be set to expire with a time-to-live (TTL)
- Large Data Types

Best used: Any application where low-latency data access, high concurrency support and high availability is a requirement.

For example: Storing massive amounts of profile data in online advertising or retail Web sites.