NoSQL Databases: a step to database scalability in Web environment

Jaroslav Pokorny
Charles University,
Faculty of Mathematics and Physics,
Malostranske n. 25, 118 00 Praha 1
Czech Republic
+420-221914265
pokorny@ksi.mff.cuni.cz

ABSTRACT

The paper is focused on so called NoSQL databases. In context of cloud computing, architectures and basic features of these databases are studied, particularly their horizontal scalability and concurrency model, that is mostly weaker than ACID transactions in relational SQL-like database systems. Some characteristics like a data model and querying capabilities are discussed in more detail. The paper also contains an overview of some representatives of NoSQL databases.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed databases;

General Terms

Algorithms, Languages.

Keywords

cloud computing, NoSQL database, CAP theorem, weak consistency, horizontal scaling, vertical scaling, horizontal data distribution.

1. INTRODUCTION

In recent years with expansion of cloud computing [1], problems of services that use Internet and require big data come to forefront (*data-intensive services*). The cloud computing even seems to be the future architecture to support especially large-scale and data-intensive applications. Switching from traditional custom-tailored middleware for business applications to cloud computing has a massive influence on the management of an application's life-cycle. On the other hand, there are certain requirements of applications, that cloud computing fulfils non-sufficiently.

For years a development of information systems has relied on *vertical scaling* (called also *scale-up*), i.e. investments into new and expensive big servers. Unfortunately, this approach using architecture shared-nothing requires higher level of skills and it is not reliable in some cases. A redistribution data on the fly can cause decreasing system performance. Database partitioning across multiple cheap machines added dynamically, so called *horizontal scaling* (also *scale-out*), can apparently ensure scalability in a more effective and cheaper way. Than to accommodate current DBMS for horizontal scaling, it seems, that today's NoSQL databases designed for cheap hardware and using also the architecture shared-nothing, can be in some cases even better solution. Besides cloud computing NoSQL databases assert oneself in Web 2.0 applications and in social networking, where horizontal scaling involves thousands nodes.

To achieve horizontal scaling, NoSQL databases had to relax some usual database characteristics. This is related, e.g., to restrictions of the relational data model and usual demands of transaction processing. The goal of this paper is to discuss the restrictions inhibiting scaling today's databases and to attempt to extend the considerations about trends in databases described in [13] and [9]. We also focus on NoSQL databases and discuss their pros and cons. A question is how just this software can ensure feasible development of cloud computing.

First, in Section 2 we introduce basic characteristics of cloud computing, as they are generally accepted today. Section 3 summarizes transactional problems that are critical for scalability of databases that is discussed in Section 4. Section 5 mentions specialized data stores, as they occurred in the last decade, and it focuses on their one variant - NoSQL databases. Finally, we summarize observations about NoSQL databases and mention another development of scalable DBMS that continues in line of traditional relational DBMS.

2. CLOUD COMPUTING

Cloud computing is indivisibly connected with other technologies, like grid computing, SOA and virtualization, which also occur separately. In this paper we are interested in technological problems of cloud computing related to provider and/or user. Issues specific to providers primarily include:

- data consistency,
- availability,
- predicable performance,
- scalable and high performance storage.

From the user point of view it is appropriate to mention a data model. Unlike to enterprise systems, where the data model is relatively well-defined, in cloud computing we meet more data models, e.g., for structured and unstructured data, multimedia, metadata, etc., moreover with models coming from external data sources. Cloud computing architecture should provide possibilities to combine these models. Other user related problems include security and encryption, interoperability and consistency guaranteed by using transactions. We will address these exclusively database issues in Section 3.

From the database point of view, we consider clouds providing a platform, i.e. the functionality PaaS (Platform-as-a-Service), where a database is contained in the underlying infrastructure. These databases can be considered and used standalone, or they are embedded into a broader service.

3. TRANSACTION PROCESSING

One of the basic features of database technology is a transactional processing characterized by *atomicity* (A), *consistency* (C), *isolation* (I) and *durability* (D), i.e. so called *ACID properties*. In practice, relational databases always have been fully ACID-compliant. Though, the database practice also shows that ACID transactions are required only in certain use cases. For example, databases in banks and stock markets always must give correct data. Consequently, business applications also demand that the cloud database be ACID compliant. In cloud computing we then usually talk about *corporate cloud databases* and conceive consistency in this sense of *strong consistency*.

Databases that do not implement ACID fully can be only *eventually consistent*. In principle, if we give up some consistency, we can gain more availability and greatly improve scalability of the database. Such approach can be suitable rather for *consumer cloud databases*. It reminds data stores for documents from 90ties, where infrequent occurrences of conflicts during updates were not so important in comparison to contributions of distribution and replication.

In contrast with ACID properties consider now the triple of requirements including consistency (C), availability (A) and partitioning tolerance (P), shortly *CAP properties*.

- *Consistency* means that whenever data is written, everyone who reads from the database will always see the latest version of the data.
- *Availability* means that we always can expect that each operation terminates in an intended response. High availability usually is accomplished through large numbers of physical servers acting as a single database with data shared between various database nodes and data replications.
- *Partition tolerance* means that the database still can be read from and written to when parts of it are completely inaccessible. Situations that cause this appear, e.g., when the network link between a significant number of database nodes is interrupted. Partition tolerance can be achieved by sending writes destined for unreachable nodes to nodes that are still accessible. Then, when the failed nodes come back, they receive the writes they missed.

There is the CAP theorem, also called Brewer's theorem, formulated by Brewer in [3] and formally proved in [10]. The CAP theorem states, that for any system sharing data it is impossible to guarantee simultaneously all of these three properties. Particularly, in Web applications based on horizontal scaling strategy it is necessary to decide between C and A. Usually DBMSs prefer C over A and P.

There are two directions in deciding whether C or A. One of them requires strong consistency as a core property and tries to maximize availability. The advantage of strong consistency, that is ACID transactions, means to develop applications and to manage data services in more simple way. On the other hand, complex application logic has to be implemented, which detects and resolves inconsistency. The second direction prioritizes availability and tries to maximize consistency. Priority of availability has rather economic justification. Unavailability of a service can imply financial losses.

Remind that existence of usual 2PC protocol ensures consistency and atomicity from ACID. Then, based on the CAP theorem, availability cannot be always guaranteed and for its increasing it is necessary to relax consistency, i.e., when the data is written, not everyone, who reads something from the database, will see correct data; this is usually called *eventual consistency* or *weak consistency*. Such transactional model uses, e.g., properties BASE (Basically Available, Soft state, Eventually Consistent) [15]. An application works basically all the time (basically available), does not have to be consistent all the time (soft state) but the storage system guarantees that if no new updates are made to the object eventually all accesses will return the last updated value. Availability in BASE is achieved through supporting partial failures without total system failure.

4. DATABASE SCALABILITY

Dynamic scalability as one of the core principles of cloud computing has proven to be a particularly essential problem for databases. Top level Web sites are distinguished by massive scalability, low latency, the ability to grow the capacity of the database on demand and an easier programming model. These and other features current RDBMS just do not provide in a cost-effective way. Relational databases (traditionally) reside on one server, which can be scaled by adding more processors, more memory and external storage. Relational database residing on multiple servers usually uses replications to keep database synchronization.

One of fundamental requirements for massive data processing in cloud is a platform for a support of database scalability. Popular relational database like Oracle have a great expressivity, but it is difficult to scale them up by increasing the number of computers instead of a single database server. Often it is necessary to go yet lower, i.e. to the operation system. A relevant example offers on Linux based operating system XtreemOS[1] for grids.

In the last decade, a new family of scalable DBMS has been developed, namely NoSQL databases discussed in Section 5. These systems scale nearly linearly with the number of servers used. This is possible due to the use of data partitioning. Technically, the method of *distributed hash tables* (Distributed Hash Table - DHT) is often used, in which couples (`key`, `value`) are hashed into buckets – partial storage spaces, each from that placed in one network node.

Horizontal data distribution enables to divide computation into concurrently processed tasks. It is obviously not easily realizable for arbitrary algorithm and arbitrary programming language. Complexity of tasks for data processing is minimized using specialized programming languages, e.g., MapReduce [8] developed in Google, and occurring especially in context of SQL databases. It is worth to mention that computing in such languages does not enable effective implementation of the

[1] http://www.xtreemos.eu/

relational operation join. Such architectures are suitable rather pro customer cloud computing.

Corporate cloud computing requires other approaches to scalability, i.e., not only NoSQL database. Some reserves are in RDBS alone. The argument that RDBMS “don't scale” is not always true. The largest RDBMS installations routinely deal with huge traffic and PBytes of data. Such databases require much memory and processing power. Traditional spinning-platter disk drive has been a limiting factor for a long time. A solution can be in SSD (solid-state drive) data storage technology. SSD storages are 100 times faster in random read/writes than the best disks on the market (up to 50,000 random writes per second). Such storages improve shared-disk database architecture that can be ideal for corporate cloud databases. Such architecture eliminates the need to partition data.

5. NOSQL DATABASES

The term *NoSQL database* was chosen for a loosely specified class of non-relational data stores. Such databases (mostly) do not use SQL as their query language. The term NoSQL is therefore confusing and in the database community is interpreted rather as „not only SQL“. Sometimes the term *postrelational* is used for these data stores. In broad sense, this database category also includes XML databases, graph databases or document databases and object databases. Some representatives of this software tools are even no databases in traditional conception at all.

A part of NoSQL databases usually simplify or restrict overhead occurring in fully-functional RDBMS. Their data is also often organized into tables and accessed only through primary key. NoSQL databases also do not support operations join and `ORDER BY`. The reason is that partitioning row data is done horizontally. The loss is also relevant in the case when full RDBMS is used on each node. If necessary, the join operation can be implemented at client side. Obviously, data can be partitioned also vertically, i.e. each part of a record is in one of two nodes. Another possibility is using replications. Despite of these restrictions, NoSQL databases enable to develop useful applications. We will show how features known in traditional databases degrade in these databases.

5.1 Data model

That what is principal in classical approaches to databases, i.e. a (logic) data model, is in particular approaches to NoSQL databases described rather intuitively, without any formal fundamentals. The terminology used is also very diverse and a difference between conceptual and database view of data is mostly blurred. We do not consider here graph databases, XML databases, and others sometimes included into NoSQL category.

5.1.1 Kinds of data models

NoSQL database store most often combinations of couples (`key`, `value`), where `key` is that what is called attribute name in relational databases or column name in SQL databases. Alternatively we speak directly about attributes and about columns and, consequently, about *column NoSQL databases*. Some of these databases are composed from collections of couples (`key`, `value`) or, more generally, they look like *semistructured documents* equipped by indexes. Sometimes, they are also called (rather inconveniently) *document-oriented* NoSQL databases. An example of such document is

```
Name:"Jack",
Address:"Maltézské n. 25, 118 00 Praha 1“
Grandchildren: {Claire: "7", Barbara: "6",
"Magda: "3", "Kirsten: "1", "Otis: "3",
Richard: "1"}
```

The value of, e.g., `Grandchildren: Barbara` is 6 (or 6 years in more “user-oriented” interpretation).

The JSON (JavaScript Object Notation) format[2] is usually used to presentation of such data structures. JSON is a binary and typed data model which supports the data types list, map, date, Boolean as well as numbers of different precision. In above example, we have used only JSON-like notation.

More simple NoSQL databases, called *key-value stores* (or *big hash tables*), contain a set of couples (`key`, `value`). A database is a set of named values. A key uniquely identifies a value (typically string, but also a pointer, where the value is stored) and this value can be structured or completely unstructured (typically BLOB). The approach key-value reminds simple abstractions as file systems or hash tables (DHT), which enables efficient lookups. However, it is essential here, that couples (`key`, `value`) can be of different types. In terms of relational data model – they may not “come“ from the same table. Though very efficient and scalable, the disadvantage of too simple data models can be essential for such databases. On the other hand, `NULL` values are not necessary, since in all cases these databases are schema-less.

5.1.2 Examples

In database CASSANDRA[3] a triple (*name*, *value*, *time_stamp*) is called a *column*. For example, the expression

```
{Name:"Jack", Address:"Maltézské n. 25, 118
00 Praha 1"}
```

represents two such columns (no time stamps are presented). A *supercolumn* has no time stamp, contains a number of columns and creates a higher named unit, e.g.,

```
Who: "person1", {Name:"Jack",
Address:"Maltézské n. 25, 118 00 Praha 1"}
```

Column family is (surprisingly) a named structure containing unbounded number of rows. Each of them has a key (raw name). *Rows* are composed from columns or supercolumns. A higher unit containing the previous structures is a *key space*, which is usually named after the application. An interesting feature is the possibility to specify ordering in a row (by column names as well as by columns in supercolumns).

The data model used in BigTable [4] can be characterized as three dimensional sorted map (*table*), whose *cells* contain a *value*. Cells are addressed by triples <*row_key*, *column*, *time_stamp*>. Cells can store multiple versions of data with timestamps.

On the API level, triples serve for lookup as well as for operations `INSERT` and `DELETE`. One or more columns are in BigTable associated in named *column families* (other notion than in CASSANDRA). A column is then addressed by *column_family:qualifier*, e.g., `Grandchildren:Barbara`. There are a fixed number of column families; the family can contain for each row a different number of columns.

[2] http://www.json.org/

[3] http://cassandra.apache.org

Time stamps model time and serve to distinguishing data versions. Documents contained in table rows are of different size, it is possible to add other data. Rows are ordered in lexicographic order by `row_key`. A Bigtable database can contain more tables.

It is not too hard to imagine a two dimensional representation of such map. To each row there is a table with so many rows, how many time stamps are used for the row. The table will have so many columns as it is the number of column families. Due to that column families are of different size for each row, it is possible to view such data as a table of sparse data. On a physical level, a vertical data distribution is used, where column families of one table are stored on different nodes. For example, column families *Customer*, *Customer_account*, *Login_information* can be placed on three nodes and conceived as three tables interconnected over *Customer_ID*. A table in BigTable and key space in CASSANDRA mean in principle the same.

An advantage of these and similar databases is richer data model in comparison with the simple approach (`key`, `value`). Data with such model fall rather to category of semistructured data. Column names actually represent tags assigned to values. For example, user's profiles, information about a product, web content (blogs, wiki, and messages), etc., are appropriate applications of column-oriented approach.

5.2 Querying

Querying in NoSQL databases is their fewest elaborated part. One of possibilities of querying NoSQL databases is (for somebody paradoxically) a restricted SQL dialect. For example, in system SimpleDB[4] the `SELECT` statement has the following syntax:

```
SELECT output_list
FROM domain_name
[WHERE expression] [sorting] [LIMIT limit]
```

where `output_list` can be: *, `itemName()`, `count(*)`, `list_of_attributes`, where `itemName()` is used for obtaining the item name. `domain_name` determines the domain, from which data should be searched. In `expression` we can use =, <=, <, > =, `LIKE`, `NOT LIKE`, `BETWEEN`, `IS NULL`, `IS NOT NULL`, etc., `sorting` the results by a particular attribute in ascending or descending order, `limit` restricts output size (default 100, maximally 2500). Operations join, aggregation, and subquery embedding are not supported.

A broader subset of SQL called GQL (Google Query Language) is used in AppEngine. Other very restricted variant of SQL is used in Hypertable[5]. This language including UPDATE and other statements is called HQL (Hypertext Query Language).

A typical API for NoSQL databases contains operations like `get(key)`, i.e., extract the value given a key, `put(key, value)` (create or update the value given its key), `delete(key)` (remove the key and its associated value), `execute(key, operations, parameters)` (invoke an operation to the value given the key, which is a data structure like list, set, etc.). More structured databases like CASSANDRA use the general form of access to data `get(keyspace, column family, row_key)`. A returned value is typically a tuple. A procedural approach to querying is typical for document-oriented CouchDB[6]. Being based on JSON-like format a more user-oriented querying can be achieved.

Due to the horizontal data distribution NoSQL databases do not support database operations join and `ORDER BY`. This restriction is actually also in the case, when fully-functional DBMS is in each node. Querying and update operations come down mostly on access through a key over a simple API (e.g., by key hashing). It seems that a development of query possibilities is left on the client. Such approach means nothing else than manual query programming, that can be appropriate for simple tasks and vice versa very time-consuming for others.

5.3 Data storing

Some (but not all) NoSQL databases are designed in such way, that for speed increasing their data is placed in memory and stored on disk after closing the work with a database or for back-ups (e.g., Redis[7]). NoSQL databases can reside one server, but more often are designed to work in cloud of servers. They can be equipped also by distributed indexes. Because the workload can be spread over many computers, we can conceive NoSQL databases as a special type of non-relational distributed DBMSs.

Physical data model is again multi-level. A database looks physically as a set interconnected tables (e.g., in a hierarchy) that are really stored in a file environment on disk. NoSQL databases use also column-oriented database techniques, which with a key associate a set of column groups. Such groups are stored on different machines. The column approach also enables simple adding information (vertical scaling) and a data compression.

An example of typical hierarchical storage is the physical level in BigTable. A table is split into so called *tablets*, each of them contains rows of some range (in accordance to given ordering). Tablets do not overlap. A tablet is identified by the table name and end key of the range. The same data structure in HBase[8] is called *region* identified by table name and start key.

CASSANDRA uses DHT for partitioning data on particular servers in the key space. Such a DHT is, e.g., organized around a ring of nodes with a possibility of dynamization by adding a new node between two nodes merging neighbouring nodes. A user of API can manipulate with DHT again by means of operations `put(key, value)` and `get(key)` → `value`. For example, in the project Voldemort[9] data is partitioned around a ring of nodes, and data from node K is replicated on nodes K+1,…,K+*n*, for a given *n* (so called *consistent hashing*).

5.4 Transaction Processing in NoSQL DB

We have mentioned that being relational and ACID is not necessary for some use cases. Moreover, it can add unnecessary overhead. Even, to reach strong consistency has not to be possible for these databases. Thus, a fixation of partition tolerance requires weaker forms of consistency or lower availability (see BASE properties in Section 3). In the case, that databases focus on A and P, they may dispense with C.

[4] http://aws.amazon.com/simpledb/

[5] http://hypertable.org

[6] http://couchdb.apache.org

[7] http://redis.io/

[8] http://hbase.apache.org/

[9] http://project-voldemort.com

Instead of strong consistency NoSQL database implement eventual consistency, whereby any changes are replicated to the entire database eventually, but in any given time. For example, Dynamo [7] provides availability and partition tolerance at the expense of consistency. This means, that a single node or group of nodes may not have the latest data. Such database then achieves low latency, high throughput that makes the web site more responsive for users.

Particular architectures use various possibilities of data distribution, ensuring availability and access to data replications. Some of them support ACID, the other eventual consistency (CASSANDRA, Dynamo), some, like SimpleDB, do not support transactions at all.

5.5 Architectures of NoSQL DB

As an example we mention a very popular Amazon's activity materialized in the data store Single Storage Service (S3)[10]. S3 enables to write, read, and delete objects of size up to 5 TBytes via a unique user-oriented key. Also SimpleDB is based on S3. S3 is most successful for multimedia objects and backups. Such objects are typically large and rarely updated. The architecture of S3 allows infinite scalability and was also used for building fully-fledged database system with small objects and frequent updates [2] and for other NoSQL databases like, e.g., Dynamo.

In Table 1 we present our own summary of NoSQL databases together with their basic characteristics focused on a data model and a way of querying. The expression `{value}` denotes a set of values. Some of these projects are more mature than others, but each of them is trying to solve similar problems. A list of various open and closed source NoSQL databases can be found in [4] and [11], well maintained and structured Web pages are http://nosql-database.org/ and so called NoSQLPedia[11]. A very detailed presentation of NoSQL databases can be found in the work [16].

With respect to differences among NoSQL databases it does not seem that a unified query standard will be developed. An associated theory of NoSQL databases is also missing. An exclusion is, e.g., the work [13] whose authors present a mathematical data model for the most common NoSQL databases.

6. CONCLUSIONS

We have presented various approaches to NoSQL databases, namely features of their models and possibilities of querying with emphasis on their use in cloud architectures. For now NoSQL databases are still far from advanced database technologies and they will not replace traditional relational DBMS. In work [12] Leavitt cites opinions of some proponents of significant IT companies. They coincide in future of NoSQL in context of usage of various database tools in application-oriented way and their broader adoption primarily in specialized projects involving large unstructured distributed data with high requirements on scaling. On the other hand, an adoption of NoSQL data stores will hardly compete with relational databases that represent huge investments and mainly reliability and matured technology.

We have shown that due to horizontal scaling it is not possible to reach simply ACID properties. However, it does not mean, that any cloud computing agrees to give up the preservation of these properties. Such systems even occur in practice. The work [4] provides a good introduction to scalable DBMS based on traditional architectures. For example, relational DBMS MySQL Cluster[12], VoltDB[13], and Clustrix[14] belong to this category.

Further, hybrid systems with multiple data stores based generally on different principles are expected to be a trend in the future. For example, the Project Voldemort is hybrid with MySQL as one of storage backend. An interesting possibility exists with object-relational databases. Considering data from a NoSQL database as semistructured data, we can represent it as XML data in a XML typed relation column and to access its values through SQL/XML language. Such an approach will be beneficial especially for corporate (cloud) computing. Other architectures for cloud computing using horizontal scaling, preserving ACID and fault-tolerant database require obviously other research.

7. ACKNOWLEDGMENTS

This research has been partially supported by the grants of GACR No. 201/09/0990 and No. P202/10/0573.

8. REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M. 2000. A View of Cloud Computing. *Comm. of the ACM*, 53, 4 (Apr. 2010), 50-58. DOI= http://doi.acm.org/10.1145/1721654.1721672

[2] Brantner, M., Florescu, D., Graf, D., Kossman, D., Kraska, T. 2000. Building a Database na S3. In *Proceedings of SIGMOD '08* (Vencouver, Canada, June 9-12, 2008), 251-263. DOI= http://doi.acm.org/10.1145/1376616.1376645

[3] Brewer, E.A. 2000. Towards Robust Distributed Systems. Invited talk on PODC 2000, July 16-19 2000, Portland, Oregon (2000).

[4] Cattell, R. 2010. Scalable SQL and NoSQL Data Stores. *SIGMOD Record*, 39, 4 (Dec. 2010), 12-27. DOI= http://doi.acm.org/10.1145/1978915.1978919

[5] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, AND., Gruber, R. E. 2006. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008) DOI = http://doi.acm.org/10.1145/1365815.1365816

[6] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., Yerneni, R. 2008. PNUTS: Yahoo!'s hosted data serving platform. *J. PVLDB* 1(2):1277–1288 (2008). DOI= http://dx.doi.org/10.1145/1454159.1454167

[7] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. 2007. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of SOSP'07* (Stevenson, Washington, USA October 14–17, 2007) , ACM, 205-220. DOI= http://doi.acm.org/10.1145/1323293.1294281

[10] http://aws.amazon.com/s3/

[11] http://nosqlpedia.com/wiki/Survey_distributed_databases#top

[12] http://www.mysql.com/products/cluster/

[13] http://voltdb.com/

[14] http://www.clustrix.com/

Table 1. Representatives of NoSQL databases

Name	Producer	Data model	Querying
column-oriented			
BigTable	Google	set of couples (key, {value})	selection (by combination of row, column, and time stamp ranges)
HBase	Apache	groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)
Hypertable	Hypertable	like BigTable	HQL (Hypertext Query Language)
CASSANDRA	Apache (originally Facebook)	columns, groups of columns corresponding to a key (supercolumns)	simple selections on key, range queries, column or columns ranges
PNUTS [5]	Yahoo	(hashed or ordered) tables, typed arrays, flexible schema	selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)
key-value			
SimpleDB	Amazon	set of couples (key, {attribute}), where attribute is a couple (name, value)	restricted SQL; select, delete, GetAttributes, and PutAttributes operations
Redis	Salvatore Sanfilippo	set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value	primitive operations for each value type
Dynamo	Amazon	like S3	simple get operation and put in a context
Voldemort	LinkeId	based on the Dynamo model	similar to Dynamo
document-based			
MongoDB	10gen	object-structured documents stored in collections; each object has a primary key called ObjectId	manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)
CouchDB	Couchbase	document as a list of named (structured) items (JSON document)	by key and key range, views via Javascript and MapReduce

[8] Dean, D., Ghemawat, S. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Comm.of the ACM* 51, 1 (January 2008) 107-113. DOI= http://doi.acm.org/10.1145/1327452.1327492

[9] Feuerlicht, G., Pokorny, J. 2011. Can Relational DBMS Scale-up to the Cloud? In *Proceedings of ISD 2011, (Edinbourgh, GB, August 24-26)*, to appear.

[10] Gilbert, S., Lynch, N. 2002. Brewer's conjecture and the feasibility consistent, available, partition-tolerant web services. Newsletter ACM SIGACT News, 33, 2 (2002) 51-59. DOI= http://doi.acm.org/10.1145/ 10.1145/564585.564601

[11] Intersimone, D. 2010. The end of SQL and relational database? (Part 2 of 3). Computerworld, February 10, 2010, http://blogs.computerworld.com/15556/the_end__sql_and_relational_database_part_2__3.

[12] Leavitt, N. 2010. Will NoSQL Databases Live Up to Their Promise? *Computer*, 43, 2 (2010), 12-14. DOI= http://doi.acm.org/10.1109/MC.2010.58

[13] Meijer, E., Bierman, G. 2011. A co-Relational Model of Data for Large Shared Data Banks*.* *Comm.of the ACM*, 54, 3 (2011), 49-58. DOI= http://*doi*.ieeecomputersociety.org/10.1145/1924421.1924436

[14] Pokorný, J. 2010. Databases in the 3rd Millennium: Trends and Research Directions. J. of Syst. Integration, Vol 1, No 1-2 (2010) 3-15.

[15] Pritchett, D. 2008. BASE: An Acid Alternative. *ACM Queue*, (May/June, 2008), 48-55. DOI= http://doi.acm.org/10.1145/1394127.1394128

[16] Strauch, Ch. 2011. NoSQL Databases. Lecture Selected Topics on Software-Technology Ultra-Large Scale Sites, Manuscript. Stuttgart Media University, 2011, 149 p., http://www.christof-strauch.de/nosqldbs.pdf