NoSQL databases: a step to database scalability in web environment

Jaroslav Pokorny
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Praha, Czech Republic

Abstract

Purpose – The paper aims to focus on so-called NoSQL databases in the context of cloud computing.

Design/methodology/approach – 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.

Findings – Some characteristics like a data model and querying capabilities of NoSQL databases are discussed in more detail.

Originality/value – The paper shows vary different data models and query possibilities in a common terminology enabling comparison and categorization of NoSQL databases.

Keywords Cloud computing, NoSQL database, CAP theorem, Weak consistency, Horizontal scaling, Vertical scaling, Horizontal data distribution, Databases, Computing

Paper type Research paper

1. Introduction

In recent years with expansion of cloud computing (Armbrust *et al.*, 2010), problems of services that use internet and require big data come to forefront (data-intensive services). For companies like Amazon, Facebook and Google the web has emerged as a large, distributed data repository, whose processing by traditional DBMSs shown to be not sufficient. Instead of extending hardware capabilities of such rather centralized DBMSs, a more feasible solution has been accepted. Technically, it is a case of scaling via dynamic adding (removing) servers from the reasons increasing either data volume in a repository or the number of users of this repository. In this context, the big data problem is often discussed as well as solved on a technological level of web databases.

On the other hand, an environment containing enterprise and corporate databases has changed also towards big data. The consult company McKinsey & Company reports in Manyika *et al.* (2011), that average investing firm with fewer than 1,000 employees has 3.8 PBytes data stored and estimates its annual data growth rate of 40 percent. In the same time, other innovation occurs in area services – cloud databases, whose architecture and a way of data processing means other way of integration and, consequently, dealing with big data.

The cloud computing even seems to be the future architecture to support especially large-scale and data-intensive applications. According to Gantz and Reinsel (2011),

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

International Journal of Web Information Systems
Vol. 9 No. 1, 2013
pp. 69-82
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/17440081311316398

while cloud computing accounts for less than 2 percent of IT spending today, by 2015, nearly 20 percent of the information will be "touched" by a cloud computing service. Also as much as 10 percent of the data will be maintained in a cloud. 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. Obviously, on the other hand, there are certain requirements of applications considered, that cloud computing fulfils non-sufficiently.

It seem that feasible scaling is a key-point of cloud computing. 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 often cited 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 applications of Web 2.0 and in social networking, where horizontal scaling involves thousands nodes. It is not by chance, that NoSQL databases having the biggest impact on this category of software, originate from development laboratories of companies Google and Amazon.

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 Pokorný (2010) and Feuerlicht and Pokorny (2012). We focus also on new architectures of DBMS where scalability has a priority and describe a restricted functionality of such databases. Particularly, 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.

This paper is an updated and extended version of Pokorný (2011). 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 scalable databases. Section 4 is devoted to discussing database scalability. 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 DBMSs that continues even in line of traditional relational DBMS.

2. Cloud computing

While there are many different opinions of cloud computing, it is useful to start from a definition. We will refer this one introduced by the National Institute of Standards and Technology (NIST; Mell and Grance, 2009):

> Cloud computing is and model for enabling convenient, on – demand network access to and shared pool of configurable computing resources (e.g. network, servers, storage, applications and services) that can be rapidly provisioned and released with and minimal management effort or service provider interaction.

Five characteristics of cloud computing common to its providers are often stated:

1. on-demand self-service – provision of resources is done without interaction with user;
2. wide network access (typically to internet);
3. resource pooling (their size, location and structure are concealed to user);
4. rapid elasticity (provision and releasing of resources at any quantity are rapid and creates the illusion of unlimited scalability); and
5. measured service (the performance and usage of resources are automatically monitored, measured and optimized).

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; and
- 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 and metadata, moreover we can deal with various models coming from the data sources used for an application. A 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 some of these exclusively database issues in Section 3.

From the database point of view, we consider clouds providing a platform, i.e. the functionality platform-as-a-service (PaaS), where a database or DBMS is contained in the underlying infrastructure and can be even considered and used standalone (see Table II in Section 5), or they it is embedded into a broader service. A well-known example in this area is AppEngine of Google[1].

3. Transaction processing

One of the basic features of relational database technology is a transactional processing characterized by atomicity (A), consistency (C), isolation (I) and durability (D). Very shortly, these ACID properties mean “all or nothing”, “the result of each transaction are tables with legal data”, “transactions are independent”, “database survives system failures”, respectively. We will call database consistency in this sense by strong consistency. 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 in this case.

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 1990, 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:

(1) Consistency means that whenever data is written, everyone who reads from the database will always see the latest version of the data. The notion is different from that one used in ACID (Brewer, 2012).

(2) 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 through data sharing between various database nodes and replications.

(3) Partition tolerance means that the database still can be read from and written to when parts of it are completely inaccessible. Situations that would cause this appear, e.g. when the network link between a significant number of database nodes is interrupted. Partition tolerance can be achieved by mechanisms whereby writes destined for unreachable nodes are sent 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 (2000) and formally proved in Gilbert and Lynch (2002). 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. Usual DBMS 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 reminds 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 two-phased commit (2PC) protocol ensures C and A from ACID. In an unreliable system then, based on the CAP theorem, A cannot be guaranteed. For any A increasing it is necessary to relax C. Corporate cloud databases prefer C rather over A and P.

A database without strong consistency means, 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. If we abandon strong consistency, we can reach better availability which will highly improve database scalability. Such approach is appropriate for customer cloud databases. A nice example of this category is described in Brewer (2012). It concerns automated teller machines (ATM). In the design of an ATM, strong consistency would appear to be the logical choice, but in practice, A trumps C, but of course with a certain risk.

A recent transactional model uses, e.g. properties basically available, soft state, eventually consistent (BASE; Pritchett, 2008). The availability in BASE corresponds to availability in CAP theorem. 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 (after the inconsistency window closes) all accesses will return the last updated value. Availability in BASE is achieved through supporting partial failures without total system failure. Eventual consistency means that the system will become consistent after some time.

Up-to-now experiences with CAP theorem indicate that a design of a distributed system requires a deeper approach dependent on the application and technical conditions. For cloud computing with, e.g. of datacentre networking, failures in the network are minimized. Then it is possible to reach both C and P high with a high probability.

A more advanced solution of consistency can be found in DBMS CASSANDRA. A consistency is tunable there, i.e. its degree can be influenced by the application developer. For any given read or write operation, the client application decides how consistent the requested data should be. This enables to use CASSANDRA in applications with real time transaction processing.

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 processing application in cloud with massive data processing 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[2] 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 (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 (Dean and Ghemawat, 2008) developed in Google, and occurring especially in context of NoSQL databases. It is worth to mention that computing in such languages does not enable effective implementation of the relational operation join. Such architectures are suitable rather pro customer cloud computing.

Corporate cloud computing requires other approaches, 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 long been a limiting factor. A solution can be in solid-state drive (SSD) 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.

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. Instead of strong consistency NoSQL databases implement eventual consistency, whereby any changes are replicated to the entire database eventually, but in any given time. For example, Dynamo (DeCandia *et al.*, 2007) 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 even support ACID, the other eventual consistency (CASSANDRA, Dynamo), some, like SimpleDB, do not support transactions at all.

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. The source: http://nosql-database.org/ mentions even more than 122 NoSQL databases in this sense. For example, graph databases are actually network databases, whose edges and nodes serve to represent and store user data structured to sets of couples (key, value). Some representatives of this software tools are even no databases in traditional conception at all. Here we only focus on some of these NoSQL approaches as they are understood by sufficiently broad part of a database community (Section 5.1).

A part of NoSQL databases usually simplify or restrict overhead occurring in fully-functional RDBMS. On the other hand, their data is often organized into tables on a logical level and accessed only through primary key. NoSQL databases mostly 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 more nodes. Both horizontal and vertical distribution support horizontal scaling.

Another possibility is using replications. Despite of these restrictions, NoSQL databases enable to develop useful applications.

5.1 Data model

What is principal in classical approaches to databases – 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.

5.1.1 Kinds of data models. Most simple NoSQL databases called key-value stores (or big hash tables) contain a set of couples (key, value). A key is in principle the same as attribute name in relational databases of column name in SQL databases. In other words, 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 (e.g. 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.

In a more complex case, NoSQL database stores combinations of couples (key, value) collected into collections. Then we talk about column NoSQL databases. Some of these databases are composed from collections of couples (key, value) or, more generally, they look like semistructured documents or extendable records often equipped by indexes. New attributes (columns) can be added to these collections.

The most general models are called (rather inconveniently) document-oriented NoSQL databases. An example of such document is:

```
  {Name: "Jack",
  Address: "Maltézské nám. 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 JavaScript Object Notation (JSON)[3] format 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. We use here an intuitive notation coming out from the example and only reminding JSON.

5.1.2 Examples. We will present data models of two column NoSQL databases – CASSANDRA and BigTable.

In CASSANDRA[4] a database a triple `(name, value, time_stamp)` is called a column. For example, the expression:

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

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

```
Who: "person1", {Name:"Jack", Address:"Maltézské nám. 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 (Chang *et al.*, 2006) can be characterized as certain three dimensional sorted map (table), whose cells contain a value. Cells can also store multiple versions of data with timestamps. Cells are addressed by triples < `row_key, column, time_stamp` >.

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. A table in BigTable and key space in CASSANDRA mean in principle the same. A Bigtable database can contain more tables.

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 to them. Rows are ordered in lexicographic order by `row_key`. In web databases such a key is often a URL. If the reverse URL is used as the `row_key`, the column used for different attributes of the web page and the timestamp indicates from then the data is. The data this key points to is some content from the web page.

It is not too hard to imagine a two dimensional representation of such map (see Table I corresponding to example in Section 5.1.1). 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 can be 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`.

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 this approach.

Table I.
A table representation of a row in BigTable

			Column family Grandchildren					
Row key	Time stamp	Column name	Ch1	A1	Ch2	A2	Ch3	A3
http://ksi...	t1	"Jack"	"Claire"	7				
	t2	"Jack"	"Claire"	7	"Barbara"	6		
	t3	"Jack"	"Claire"	7	"Barbara"	6	"Magda"	3

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[5] 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 2,500). Operations join, aggregation and subquery embedding are not supported.

A broader subset of SQL called Google query language (GQL) is in the already mentioned AppEngine. Other very restricted variant of SQL is used in Hypertable[6]. This language including UPDATE and other statements is called hypertext query language (HQL).

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 special data structure, e.g. `list, set)`. 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 there. A procedural approach to querying is typical, e.g. for CouchDB[7].

There are also more user-oriented approaches to querying like, e.g. in the project Voldemort by using the JSON data type.

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. If necessary, the operation join can be implemented on the client side. Operation selection is in NoSQL databases often described on API level, even the code.

Thus, 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, e.g. adding search by key words, or even using a relational database for storing metadata about objects in NoSQL database. Such approach means nothing else than manual query programming, that can be appropriate for simple tasks and vice versa very time-consuming for others. There are also more user-oriented approaches to querying like, e.g. in the project Voldemort[8] by using the JSON data type.

5.3 Data storing

Relational databases are usually stored on disk or in a storage area within a network. Sets of database rows are transmitted into memory by a `SELECT` statement of the SQL language by operations of a stored procedure.

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

As an example of typical hierarchical storage we can mention 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[9] is called region identified by table name and start key. Rows in region are ordered in lexicographic order from start key to end key. CouchDB uses a B-tree for storing couples `(key, value)` in such way that they are sorted by key. Most of databases considered use indexes on unique keys or fields of any type (e.g. MongoDB[10]).

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)` $\rightarrow$ value. We will present how as DHT designed in the project Voldemort. Data is partitioned around a ring of nodes, and data from node K is replicated on nodes $K + 1, \ldots, K + n$, for a given n (so called consistent hashing).

Data in NoSQL databases are often stored in special file systems. As an example of data storage usable for "higher" systems we remind very popular activity of Amazon implemented in the file system single storage service (S3)[11]. Above mentioned SimpleDB is based on S3. S3 allows insert, read and delete objects of size do 5 TByte via a unique user-oriented key. S3 is most successful for multimedia objects and back-up. Such objects are typically large and seldom actualized.

The open source software Hadoop[12] has more general usability. It is based on the framework MapReduce for data processing and the distributed file system HDFS (Hadoop Distributed FileSystem). On the top of HDFS there is database HBase.

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. Such databases are called in memory databases (e.g. Redis[13]). NoSQL databases can reside one server, but more often are designed to work in cloud of servers. They are 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.

5.4 Architectures of NoSQL databases

Particular architectures of NoSQL databases use different possibilities of distribution, ensuring availability and access to data replication. Some of them support ACID, another ones eventual consistency (CASSANDRRA, Dynamo). The other, e.g. SimpleDB, do not support transactions at all.

Other important aspect particularly for cloud databases is their scalability. For example, the architecture of S3 allows infinite scalability and was also used for building fully-fledged database system with small objects and frequent updates (Brantner *et al.*, 2008) and for other NoSQL databases like, e.g. Dynamo.

Most of NoSQL databases employ asynchronous replication. This allows writes to complete more quickly since they do not depend on extra network traffic.

In Table II we present own summary of NoSQL databases together with their basic characteristics focused on a data model, a way of querying, a way of replicas processing (As – asynchronous, S – synchronous), and transactions possibilities (L – local transactions, N – no transactions). 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 opened and closed source NoSQL databases can be found in Cattell (2010) and Intersimone (2010), well maintained and structured

Name	Producer	Data model	Querying	Rep	Tr
Column oriented					
BigTable	Google	Set of couples (key, {value})	Selection (by combination of row, column, and time stamp ranges)	As + S	L
HBase	Apache	Groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)	As	L
Hypertable	Hypertable	Like BigTable	HQL	As	L
CASSANDRA	Apache (originally Facebook)	Columns, groups of columns corresponding to a key (supercolumns)	Simple selections on key, range queries, column or columns ranges	As	L
PNUTS (Cooper *et al.*, 2008)	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)	As	L
Key-valued					
SimpleDB	Amazon	Set of couples (key, {attribute}), where attribute is a couple (name, value)	Restricted SQL; select, delete, GetAttributes, and PutAttributes operations	As	N
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	As	N
Dynamo	Amazon	Like simple DB	Simple get operation and put in a context	As	N
Voldemort	LinkeId	Like simple DB	Similar to dynamo	As	N
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)	As	N
CouchDB	Couchbase	Document as a list of named (structured) items (JSON document)	Views via Javascript and MapReduce	As	N

Table II. Representatives of NoSQL databases

web pages are http://nosql-database.org/ and already mentioned DBPedias. A very detailed presentation of NoSQL databases can be found in the work (Strauch, 2011).

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. The exclusion is, e.g. the work (Meijer and Bierman, 2011) whose authors present a mathematical data model for the most common NoSQL databases, namely key-value relationships and demonstrate that this data model is the mathematical dual of SQL's relational data model of foreign key – primary key relationships.

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. The work (Leavitt, 2010) cites opinions of some proponents of successful and significant IT companies. They coincide in future of NoSQL in context of usage of various database tools in application-oriented way, their broader adoption primarily in specialized projects involving large unstructured distributed data with high requirements on scaling. Some voices are even more sceptic. An adoption of NoSQL data stores will hardly compete with relational databases that represent huge investments and mainly reliability and matured technology. According to the ReadWriteWeb blog post by Audrey Watters, 44 percent of enterprise users questioned had never heard of NoSQL and an additional 17 percent had no interest in year 2010.

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. Other architectures of cloud computing using horizontal scaling, preserving ACID and fault-tolerant database will obviously require other research. Such systems even occur in practice. The work (Cattell, 2010) provides a good introduction to scalable DBMS based on traditional architectures. For example, relational DBMS MySQL Cluster[14], VoltDB[15] and Clustrix[16] belong to this category.

These requirements are reflected in a new trend (from April 2011) denoting a next generation of highly scalable and elastic RDBMS as NewSQL databases. Here are some their properties:

- they are designed to scale out horizontally on shared nothing machines;
- still provide ACID guarantees;
- applications interact with the database primarily using SQL;
- the system employs a lock-free concurrency control scheme to avoid user shut down; and
- the system provides higher performance than available from the traditional systems.

Also hybrid systems with multiple data stores based generally on different principles are expected to be a trend in the future. For example, already mentioned Voldemort is hybrid with MySQL as one of storage backend. An interesting possibility exists with object-relational databases. Considering data from a NoSQL as semistructured data, it could be suitable to represent it as XML data in a XML typed column on a logical level and to access it by the SQL/XML language in hybrid approach. Clearly, such an approach will be beneficial especially for corporate (cloud) computing.

Notes

1. http://code.google.com/intl/cs/appengine/docs/whatisgoogleappengine.html
2. www.xtreemos.eu/
3. www.json.org/
4. http://cassandra.apache.org
5. http://aws.amazon.com/simpledb/
6. www.hypertable.com
7. http://couchdb.apache.org
8. http://project-voldemort.com
9. http://hbase.apache.org/
10. www.mongodb.org
11. http://aws.amazon.com/s3/
12. http://hadoop.apache.org/
13. http://redis.io/
14. www.mysql.com/products/cluster/
15. http://voltdb.com/
16. www.clustrix.com/

References

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010), "A view of cloud computing", *Communications of the ACM*, Vol. 53 No. 4, pp. 50-8.

Brantner, M., Florescu, D., Graf, D., Kossman, D. and Kraska, T. (2008), "Building and database on S3", *Proc. of ACM SIGMOD Conf. '08, Vancouver, Canada*, ACM, New York, NY, pp. 251-63.

Brewer, E.A. (2000), "Towards robust distributed systems", Invited Talk on PODC 2000, Portland, Oregon, 16-19 July.

Brewer, E.A. (2012), "CAP twelve years later: how the 'rules' have changed", *Computer*, Vol. 45 No. 2, pp. 22-9.

Cattell, R. (2010), "Scalable SQL and NoSQL data stores", *SIGMOD Record*, Vol. 39 No. 4, pp. 12-27.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. (2006), "Bigtable: a distributed storage system for structured data", *Proc. of 7th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI 06*), available at: 06?page=2">www.usenix.org/search/site/osdi'06?page=2 (accessed 30 July 2012).

Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D. and Yerneni, R. (2008), "PNUTS: Yahoo!'s hosted data serving platform", *PVLDB*, Vol. 1 No. 2, pp. 1277-88.

Dean, D. and Ghemawat, S. (2008), "MapReduce: simplified data processing on large clusters", *Communications the ACM*, Vol. 51 No. 1, pp. 107-13.

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", *SOSP'07, Stevenson, Washington, DC, USA, 14-17 October*, ACM, New York, NY, pp. 205-20.

Feuerlicht, G. and Pokorny, J. (2012), "Can relational DBMS scale-up to the cloud?", in Pooley, R.J., Coady, J., Linger, H., Barry, C. and Lang, M. (Eds), *Information Systems Development – Reflections, Challenges and New Directions*, Springer, Berlin.

Gantz, J. and Reinsel, D. (2011), "Extracting value from chaos", IDC iView, available at: http://idcdocserv.com/1142 (accessed 30 April 2012).

Gilbert, S. and Lynch, N. (2002), "Brewer's conjecture and the feasibility consistent, available, partition-tolerant web services", *Newsletter ACM SIGACT News*, Vol. 33 No. 2, pp. 51-9.

Intersimone, D. (2010), "The end of SQL and relational database? (Part 2 of 3)", *Computerworld*, 10 February, available at: http://blogs.computerworld.com/15556/the_end__sql_and_relational_database_part_2__3 (accessed 30 July 2012).

Leavitt, N. (2010), "Will NoSQL databases live up to their promise?", *Computer*, Vol. 43 No. 2, pp. 12-14.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. and Byers, A.-H. (2011), "Big data: the next frontier for innovation, competition, and productivity", McKinsey Global Institute, available at: www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation (accessed 30 July 2012).

Meijer, E. and Bierman, G. (2011), "A co-relational model of data for large shared data banks", *Queue – Programming Languages*, Vol. 9 No. 3, pp. 1-19.

Mell, P. and Grance, T. (2009), "The NIST definition of cloud computing", *National Institute of Standards and Technology*, Vol. 53 No. 6, p. 50.

Pokorný, J. (2010), "Databases in the 3rd millennium: trends and research directions", *Journal of Systems Integration*, Vol. 1 Nos 1/2, pp. 3-15.

Pokorný, J. (2011), "NoSQL databases: a step to database scalability in web environment", *Proc. of the 13th Int. Conf. on Information Integration and Web-Based Applications & Services (iiWAS) 2011, Ho Chi Minh City, Vietnam*, ACM, New York, NY, pp. 278-83.

Pritchett, D. (2008), "BASE: an ACID alternative", *ACM Queue*, May/June, pp. 48-55.

Strauch, Ch. (2011), "NoSQL databases", *Lecture Selected Topics on Software-Technology Ultra-Large Scale Sites*, Stuttgart Media University, p. 149, manuscript, available at: www.christof-strauch.de/nosqldbs.pdf (accessed 30 July 2012).

About the author

Jaroslav Pokorny received his PhD degree in theoretical cybernetics from Charles University, Prague, Czechoslovakia, in 1984. He is a Full Professor of Computer Science at the Faculty of Mathematics and Physics, Charles University, Prague. He is also a visiting Professor at the Faculty of Electrical Engineering of Czech Technical University, Prague. He has published more than 290 papers and books on data modelling, relational databases, query languages, XML technologies, and data organization. His current research interests include semi-structured data, web technologies, indexing methods, and social networks. He is a member of ACM and IEEE. He works also as the representative of the Czech Republic in IFIP. Jaroslav Pokorny can be contacted at: Pokorny@ksi.mff.cuni.cz