Analyzing and Comparison of NoSQL DBMS

Anna Kuzochkina, Mariya Shirokopetleva, Zoia Dudar
Department of software engineering
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
anna.kuzochkina@nure.ua, marija.shirokopetleva@nure.ua, zoia.dudar@nure.ua

***Abstract* – In this article we described the main characteristics and types of NoSQL technology while approaching different aspects that highly contribute to the use of those systems. Also, three different types of application were considered and based on them several NoSQL databases were analyzed. According to the results, we summarize which databases are more preferable to solve specific type of problem.**

Keywords – NoSQL DBMS; multiplicative convolution; criteria; normalizing factors; weighting factors; Pareto set; analyzing; comparison; decision making methods; additive convolution;

I. Introduction

NoSQL databases are the non-relational databases with a possibility of scaling optimization for applying of data models without uniform diagram. NoSQL databases were widely adopted as they simplify designing and operating and provide fail safety.

NoSQL removes all restrictions of a relational model (labor-consuming horizontal scaling, inapplicable productivity) and makes accessing data easier. Such kind of databases use approach of creating the specific structure on the fly, in this way lifting limits of tight couplings and offering different access types to specific data.

The purpose of creating NoSQL DBMS was to provide the user with additional functionality that was not allowed by using RDBMS.

II. Math Model

Essence of multicriterion tasks of making decision: variants-"candidates" are compared on two or more to the criteria, to find an optimal variant (or one of optimal, if different “candidates” divide the “first place”) [1].

We will define now, what relation of Pareto. A variant *x* is better than variant *y* on the relation of Pareto (further: $x > y$), if *x* even on one criterion better, than *y*, and on other criteria not worse, than *y*. This relation is transitive (if $x > y$ and $y > z$, then $x > z$), irreflexive (*x* is impossible $> x$), asymmetric (it is impossible simultaneously $x > y$ and $y > x$), i.e. this strict ordering relation. We notice that between a certain pair (x, y) not always it is possible to set the relation of Pareto. Such relation will not set, if each of pair in something better, and in something worse than "partner".

The variant *x* is named an optimal Pareto decision, if there is not such variant of *y*, that $y > x$ by Pareto. The great number of such decisions was called a set of Pareto.

Therefore, it is required to accept some decision, choosing one of possible variants as “optimal” and there are a few criteria of efficiency of decision. For taking of multi criterion task to the one criterion, we will use multiplicative convolution.

Multiplication of the criteria is multiplicative convolution [2]. In this case, it is possible, before multiplying the criteria, to raise them to a power the greater, the greater the importance given to the criterion. Obviously, the multiplicative convolution is justified if the criteria are nonnegative – otherwise the rule “minus to minus gives plus” will play us a bad joke, making a “good” convolution of two obviously bad criteria. However, if only one of the criteria takes negative values, such paradoxes do not arise, and we can use the multiplicative convolution. It should also be considered that if one of the criteria is zero, then the multiplicative convolution is zero. In general, in the multiplicative convolution the greatest influence is exerted by those criteria that have low values for a given object.

The multiplicative criterion follows the principle of fair relative compensation. It consists of the following. We will assume for simplicity that all the partial criteria are “good”. Let us assume that by changing some values of alternatives, we are trying to increase some particular criteria. Usually, because of the problem of the conflict between the LPP (linear programming problem), an enhancement of some "good" partial criteria leads to a reduction in other "good" particular criteria. The problem arises in finding a compromise between increasing one moreover, decreasing other criteria.

So, we have *m* alternatives and *n* criteria. Then the decision matrix $n \times m$ is built. Values for alternative are populated in one column and refer to *j* index, values for criterion are populated in one row and refer to *i* index.

Multiplicative convolution with normalizing factors is following the next law

$$Z = max \prod_{i=1}^{m} \alpha_i x_{ij} \quad (1)$$

where α_i – normalizing factors, x_{ij} – alternative values for each criterion in decision matrix $n \times m$ (Table I).

Normalizing factors are calculated according to the (2):

$$\alpha_i = \frac{1}{\sum_{j=1}^{n} x_{ij}} \quad (2)$$

The multiplicative convolution is based on the postulate: “a low score of at least one criterion entails a low value of the utility function”.

TABLE I. DECISION MATRIX

	Alternative 1		Alternative m
Criterion 1	x_{11}	…	x_{1m}
	…	…	…
Criterion n	x_{1n}	…	x_{nm}

Weight (weighting factors) of the criteria – the subtle place in the problem of the criterial ordering of alternatives. Often, weight is assigned based on the intuitive idea of the comparative importance of the criteria. Nevertheless, studies show that a person (an expert) cannot directly assign the correct numerical weight to the criteria.

Soviet mathematician Vladislav Podinovsky is the founder of a scientific approach to the problem of the importance of criteria.

Let us have n criteria. The most important alternative is assigned *m* points, next in importance – *m-1* points, the last – one point. The estimates obtained are divided by *(1 + 2 + ... + m)* – so that the sum of total weight is equal to one.

Then, multiplicative convolution with normalizing and weighting factors is described in (3).

$$Z = max \prod_{i=1}^{m} w_i \alpha_i x_{ij} \qquad (3)$$

where α_i – normalizing factors, w_i – weighting factors, x_{ij} – alternative values for each criterion in decision matrix $n \times m$.

Weighting factors are calculated according to the (4):

$$w_i = \frac{k}{\sum_{i=1}^{m} m} \qquad (4)$$

where k – is assigned by the expert separately for each alternative.

III. Analyzing And Comparison

Databases is the logical simulated storages intended for different data types. DBMS is special applications or a set of libraries for operation with different databases.

SQL databases are a method to group all data stores available with use of a structured query language as main (and most often) a method of communication with them, it means that he requires that the database supported structures, which are general for such systems, like "tables", "columns", "lines", "relations", etc. NoSQL appeared to avoid use of the SQL standards [3].

NoSQL is an overall definition term for a class of not relational data storages which compete with relational databases and have a higher warranty of ACID [4]. Such databases may not require the predefined schemas of tables and avoid union operations. This term was popularized at the beginning of 2009.

A. Analyzing NoSQL storage

Usually the choice of this or that NoSQL-solution begins with a data model choice. Some decisions support several different models, but usually select the following families: document, key-value and column storages.

Storages of key-value type. Key-value stores are the simplest kind of database, being, in fact, an associative array – each value is assigned its own unique key. Such DB stores data in the form of couples a key-value. In certain cases, arrays, lists, sets (sets of unique values), etc. can be used as values. Usually they realize the minimum set of operations (to set, read value, etc.) [5]. Typical application of these decisions – caching of data for increase overall performance of application (for example, results of requests to more difficult systems) and counters. Sometimes they are applied as the intermediate link to the systems of logging or collection of statistics. At the same time if the task is slightly more difficult, the key-value model will be insufficient for such decision or it will be required to manually develop a set of operations at the level of application. Also, some storages do not allow to hold bigger data volume, than the size of a random-access memory on one node of a cluster. They have no unloading function "not located" data on a disk. Key-value databases are not very suitable as a complete replacement for relational databases, but they have been used as object caches because there are no links between cached objects of different users, only the access speed to the cache is important, and so the ability to quickly scale the system.

Examples of DBMS: Oracle NoSQL Database, Amazon DynamoDB, Scalaris, Tokyo Cabinet, MemcacheDB, Voldemort, Redis.

Extensible record store. This is an enhanced version of the key-value databases. It is a two-dimensional array, where each row, which represents key, contains several key-value couples referred to it [6]. Distributed storage stores data by grouping them into columns. The rows and columns of such a model form a discharged matrix, the relationships between the data are not as stringent as in the relational model. Both keys and columns can be used as keys. The model allows to store and use very large volumes of unstructured data of various types. Such databases are usually used when there are not enough simple key-value pairs. Typically, for web indexing and other tasks that involve huge amounts of data.

Examples of DBMS: Apache HBase, Apache Cassandra, Apache Accumulo, SimpleDB.

Document storage. Such storage structure is similar to a key-value, but only the structured documents like JSON can be stored as value. Document-oriented databases support hierarchical structures with much greater nesting than "key-value" and distributed repositories. This allows you to describe the arbitrarily complex structure of the document and write it to the database. But such databases have a serious drawback: in case of requesting specific fields of this document, the whole document comes back, which negatively affects the performance of software applications for working with the database. Despite quite big functionality and ability of data access on one key, such DBMS has a bunch of the problems. For example, in case of access to one document you receive the whole one in response even if you need only one field. That affects productivity.

Thus, it can be concluded that document-oriented databases will find their application in tasks where orderly storage of information is required, but there are not many relationships between the data and you do not need to constantly collect statistics on them.

Examples of DBMS: Couchbase Server, CouchDB, MarkLogic, MongoDB, eXist.

B. Overall comparison of specific DBMS

A few mainstream NoSQL databases are considered to be preferred to other NoSQL alternatives. A few of these databases were considered – Aerospike, Cassandra, CouchDB, MongoDB. Particular attention is paid to the principal attribute-based evaluation of NoSQL databases, described in [7]. In Table II chosen databases are introduced, itemizing their attributes. Each NoSQL database is described according to key characteristics: category, positioning in the context of the CAP theorem, consistency guarantees and configurability, durability guarantees and configurability, querying possibilities and mechanisms, concurrency control mechanisms, partitioning schemes and the existence of native partitioning and replication.

TABLE II. SUMMARY TABLE WITH CHARACTERISTICS OF THE SELECTED NOSQL DATABASES

Charac-teristics	**Databases**			
	Aerospike	***Cassandra***	***CouchDB***	***MongoDB***
Category	Key-Value	Column-Store	Document-Store	Document-Store
CAP	AP	AP/CP	CP	CP
Consisten-cy	Configurable (several options)	Configurable (several options)	Eventual Consistency	Configurable (several options)
Durability	Notified written to replica nodes	Configurable (several options)	Configurable (several options)	Configurable (several options)
Querying	Internal API	Internal API, SQL like (CQL)	Internal API MapReduce	Internal API, MapReduce, complex query support
Concurren-cy Control	Read-commited isolation level (support for optimistic concurrency control)	MVCC	MVCC (application can select Optimistic or Pessimistic locking)	Master-slave with multi-granularity locking
Partitio-ning Scheme	Proprietary (Paxos based)	Consistent Hashing	Consistent Hashing	Consistent Hashing
Native Partitio-ning	Yes	Yes	Yes	Yes
Replica-tion	Async	Async	Async	Async

Table II summarizes the features of the NoSQL DBMS. Each NoSQL database is described according to key characteristics:

- category;
- positioning in the context of the CAP theorem;
- consistency guarantees and configurability;
- durability guarantees and configurability;
- querying possibilities and mechanisms;
- concurrency control mechanisms;
- partitioning schemes;
- the existence of native partitioning.

We studied the literature [8] according to the mentioned DBMS and identified which influence each DB has on the chosen attributes.

Each section explores the NoSQL literature on a given quality attribute, drawing conclusions regarding all the evaluated NoSQL databases. This information is then summarized in table II. Each column in table II represents a NoSQL database, and each row one of the studied software engineering quality attributes. A 5-point scale ranging from "Great for this quality attribute" (is 5) to "Bad for this quality attribute" (is 1) is presented. This allows for a direct comparison among databases. We should highlight that there are more quality attributes that should be focused on, which we intend to do in future work, and, thus, that this table does not intend to show that "one database is better than another", but, rather, that some database is better for a particular use case scenario where these attributes are needed.

As we can see all our alternatives are optimal by Pareto and our set of Pareto consists of all four alternatives.

Let us rewrite table II in the numeric equivalent (table III).

TABLE III. DECISION MATRIX FOR NOSQL DBMS

Criteria	**Alternatives**				
	A_1	A_2	A_3	A_4	***Normali-zing factors, α***
C_1	10	9	10	5	1/34
C_2	8	9	7	10	1/34
C_3	5	8	8	8	1/29
C_4	7	6	7	7	1/27
C_5	8	5	8	10	1/31
C_6	9	4	8	8	1/29
C_7	5	8	4	7	1/24
C_8	8	7	6	6	1/27
C_9	10	8	7	5	1/30
C_{10}	4	7	7	4	1/22
C_{11}	7	9	8	7	1/31
Result of multipli-cative convolu-tion with normali-zing factors	20.93 E-08	20.34 E-08	21.88 E-08	12.21 E-08	

Therefore, as we decided to choose alternative with the greatest values for all criteria, the best option will be with the greatest result of multiplicative convolution with normalizing factors. In our case this is A_3, that is stand for CouchDB.

C. Comparing DBMS in specific situations

No one of those data stores is best for all uses. A client's prioritization of highlights will be diverse relying upon the application due to the needed kind of scalability. Therefore, during this section we consider some samples of applications that work well with the various data store classes.

However, as the results do not consider the specifics of the concrete problem let us review applicability of each DBMS for solving specific problems according to the table III. Let us consider three different situations mentioned in [9].

1) You have a simple application of similar objects and the only purpose to filter them on one criterion. As an example, suppose you have a web application that does many RDBMS queries to create a tailored page when a user logs in.

2) You have a simple application with multiple different kinds of objects, where you need to search/filter objects based on several criteria. You can tolerate an "eventually consistent" model with limited atomicity and isolation.

3) You have a simple application for storing customer information and you want to partition your data both horizontally and vertically:

you might want to separate the rarely-changed "core" customer information such as customer addresses and email addresses in one place and put certain frequently-updated customer information in a different place, to improve performance.

We designate the weighting factors for all criteria according to each of described situation (table IV). Let us take six major criteria and set them coefficient from six to one due to the importance of each criterion. After that we divide each criterion on the total sum of all coefficients. "–" means we won't consider current criterion in total result.

TABLE IV. WEIGHTING FACTORS

Criteria	**Situations**		
	Weighting factors for situation #1	***Weighting factors for situation #2***	***Weighting factors for situation #3***
C_1	6/51	–	5/51
C_2	5/51	5/51	6/51
C_3	–	4/51	1/51
C_4	1/51	–	–
C_5	4/51	6/51	–
C_6	2/51	1/51	–
C_7	–	3/51	–
C_8	–	1/17	–
C_9	3/51	–	4/51
C_{10}	–	–	2/51
C_{11}	–	–	3/51

The result of multiplicative convolution with normalizing and weighting factors for different situations is present in table V.

TABLE V. RESULT OF MULTIPLICATIVE CONVOLUTION

Result of multiplicative convolution with normalizing and weighting factors	**Alternatives**			
	A_1	***A_2***	***A_3***	***A_4***
Situation #1	**40.21 E-10**	7.75 E-10	21.89 E-10	13.96 E-10
Situation #2	16.84 E-10	11.79 E-10	12.57 E-10	**39.29 E-10**
Situation #3	13.70 E-10	**39.98 E-10**	26.87 E-10	6.85 E-10

As we can see, for the first situation suits A_1 – Aerospike. Aerospike is really fast DB. And we almost managed to reach million operations per second (on data in memory). It takes several seconds to execute RDBMS queries, and then user's data is rarely changed. Then the user's tailored page is stored as a single object in a key-value store, represented in a manner that's efficient to send in response to browser requests, and index these objects by user ID. If we store these objects persistently, then we may be able to avoid many RDBMS queries, reconstructing the objects only when a user's data is updated. This allow us to save time on querying database. Also, Aerospike works well for SSD, especially if we consider that it does not use caching of data in memory, and on each request addresses a disk directly [10]. Means, it is really possible to place a large number of data in Aerospike (so far there will be enough disks, but not the RAM).

For the second situation suits A_4 – MongoDB. MongoDB is quite slow on write action but is rather fast on reading. That might be good in the case when there is no need to know if the data was updated in the past minute, and it would be quite unlikely for two records to be updated at the same time. It is also necessary to remember that MongoDB have a global lock on reading/writing that can deliver problems in case of very high loading. So, if you require the data be up-to-date and atomically consistent, e.g. if you want to lock out logins after three incorrect attempts, then you need to consider other alternatives, or use a mechanism such as quorum-read to get the latest data. In general MongoDB – a good DB for a web.

For the third situation suits A_2 – Cassandra. Cassandra is the only DB which writes quicker than reads. It is because the record in it successfully ends (in the fastest option) right after logging (on a disk). And reading requires checking, several disk readings, a choice of the latest record. Although this kind of horizontal/vertical partitioning could be done on top of a document store by creating multiple collections for multiple dimensions, the partitioning is most easily achieved with an extensible record store like Cassandra. This increases development and design phases. is perfectly suits for the application which has multiple kinds of objects, with lookups based on any field.

IV. CONCLUSIONS

Considering all mentioned situations, we can conclude the following scenarios:

1. Cassandra – the scalable open-source NoSQL DB perfectly suitable for tasks of control big quantity of the unstructured, semi-structured and structured data sets in several data-centers or in a cloud. Cassandra provides the continuous accessibility, the linear scalability. It can be used on several servers without uniform point of a failure. Using Cassandra, it is possibly to efficiently process a petabyte of data and thousands of parallel user operations per second.

2. CouchDB - the document-oriented database management system which is not requiring the description of the data scheme. It gives great accessibility abilities (combined with great stabilization and recuperation times), making it a decent contender for circumstances where failover will undoubtedly happen.

3. Aerospike experiences information loss issues influencing its durability in synchronous mode. On the other hand, it has a possibility of a clustering. At the same time the system uniformly distributes data on cluster nodes, competently using the hardware resources. If cluster are in the different data-centers system automatically understands it and uses in case of load distribution. Aerospike operates with a concept of strict consistency. Allows to maintain consistency of data on different nodes of a cluster.

4. MongoDB offers the document-oriented data model. Thanks to this MongoDB works quicker, has better scalability, it is easier to use it. All MongoDB system can represent not only one database which is on one physical server. The storage system of data represents a set of replicas. In this set there is the main node and also there can be a set of secondary nodes. Absence of the tough database scheme and regard to this the need to recreate this considerably in case of the slightest change of the concept of data storage, facilitates operation with DB and their further scaling. Besides, time of developers is saved.

Finally, we can draw the following conclusions and advices in which situations each of the NoSQL DBMS will be preferable and will show its best side.

Key-value can be used for:

- caching – fast and frequent data storage for future use;
- queue – supporting of lists, sets and queues;
- live update of information – applications using states.

Column-store DBMS are better for:

- storage of unstructured, non-destructible data – if you need to store large amounts of data for a long time, such databases will be very good at the task;
- scaling – as planned, such databases are easily scaled. They easily cope with any amount of data.
- Document-oriented DBMS are preferable for:
- data which should be stored not in lines and columns, but in the form of JSON-like documents which model is tree, not table;
- the integrity only at the level of separate records (but not at the level of communications in between them);
- vertical scalability – it supports not only huge clusters, but also portable devices (netbooks, smartphones and so forth);

REFERENCES

[1] S. Belinschi, "A note on regularity for free convolutions", *Annales de l'Institut Henri Poincare (B) Probability and Statistics,* vol. 42, no. 5, pp. 635-648, 2006.

[2] I. Chernega, P. Galindo and A. Zagorodnyuk, "A multiplicative convolution on the spectra of algebras of symmetric analytic functions", *Revista Matemática Complutense*, vol. 27, no. 2, pp. 575-585, 2013.

[3] S. Rautmare and D. M. Bhalerao, “MySQL and NoSQL database comparison for IoT application”, *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, 2016, pp. 235-238.

[4] C. Wu, Q. Zhu, Y. Zhang, Z. Du, X. Ye, H. Qin, and Y. Zhou, "A NoSQL–SQL Hybrid Organization and Management Approach for Real-Time Geospatial Data: A Case Study of Public Security Video Surveillance", *ISPRS International Journal of Geo-Information*, vol. 6, no. 1, pp. 21, 2017.

[5] D. Alves Florencio, D. Ricardo Freitas de Oliveira, E. Laisa Soares Xavier Freitas and F. da Fonseca de Souza, "Which Fits Better? A Comparative Analysis about NoSQL Key-Value Databases", *IEEE Latin America Transactions*, vol. 15, no. 11, pp. 2251-2256, 2017.

[6] G. Bathla, R. Rani and H. Aggarwal, "Comparative study of NoSQL databases for big data storage", *International Journal of Engineering & Technology*, vol. 7, no. 26, pp. 83, 2018.

[7] S. Freire, D. Teodoro, F. Wei-Kleiner, E. Sundvall, D. Karlsson and P. Lambrix, "Comparing the Performance of NoSQL Approaches for Managing Archetype-Based Electronic Health Record Data", *PLOS ONE*, vol. 11, no. 3, pp. e0150069, 2016.

[8] J. Lourenço, B. Cabral, P. Carreiro, M. Vieira and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation", *Journal of Big Data*, vol. 2, no. 1, pp. 18, 2015.

[9] R. Cattell, "Scalable SQL and NoSQL data stores", *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12, 2011.

[10] R. Sánchez-de-Madariaga, A. Muñoz, A. Castro, O. Moreno and M. Pascual, "Executing Complexity-Increasing Queries in Relational (MySQL) and NoSQL (MongoDB and EXist) Size-Growing ISO/EN 13606 Standardized EHR Databases", *Journal of Visualized Experiments*, no. 133, pp. e57439, 2018.