IRSTI 20.17.17

[1]G.T. Balakayeva, [1*]D.K. Darkenbayev, [2]Chris Phillips

[1]Faculty of Information Technologies, al-Farabi Kazakh National University, Almaty, Kazakhstan
[2]University of Newcastle upon Tyne, Newcastle, Great Britain
[*]e-mail: dauren.kadyrovich@gmail.com

Investigation of technologies of processing of big data

Abstract. An overview of the technologies and methods is presented, on the basis of which the authors of this article model the processing of a large amount of data, developing a web application. In particular, it is proposed to combine models to improve the efficiency of processing large amounts of data. Large data set before traditional storage systems and processing a new challenge. This article analyzes possible methods their decisions, limitations that do not allow to do it effectively, and also provides an overview of three modern approaches to working with large data: NoSQL and real-time event flow processing. Analysis of large data requires the use of technology and the means to implement highly productive computing. The main factors of the problem are, first of all, the complexity and the second physical volume of the information collection. It should be noted that the actual processing of data includes the construction of the algorithm and the time for its description and debugging. Unique data collections require the development of unique algorithms, which increases the total processing time by an order of magnitude.
Key words: BigData, DataMining, modeling of large data processing, NoSQL, data analysis, modeling, analysis.

Introduction

One of the topical tasks of many fields of science and technology is the task of processing large amounts of data, for example. The use of effective technologies of processing of large data allows enterprises to take a new level of work in such areas as: improving the quality of service, product development, risk management, security, cost optimization. The processing large amounts of data is relevant for geoinformatics, aerospace imagery obtained from remote sensing of the Earth, bioinformatics – analysis and ordering of genomic and proteomic information, etc.

The information value of large data is obvious, the proof of the topic is a set of tasks that can be solved by analyzing information flows of large data:

- forecast of the outflow of customers – based on analysis of data from call centers, technical support services and website traffic;
- creation of predictive models
- detection of fraud in real time
- risk analysis
- construction of situation rooms
- operative analytical processing, etc.

To solve the problem of large amounts of data, a special version of NoSQL databases was developed (http://www.nosql-database.org). A comparison of the properties of relational databases and NoSQL is presented in the table below [1,2].

Table 1 – Comparison of relational base data and NoSQL

Relational databases	NoSQL databases
Complex data relationships	Very simple relationship
Scalability	Arbitrary scheme; unstructured data
Static memory	Distributed Processing
Universal properties and functions	The memory is scaled together with the computing resources
	The system is application and developer oriented

NoSQL database features

There are not many common characteristics for all NoSQL, since many different systems are hidden under the NoSQL label. Many characteristics are peculiar only to certain NoSQL databases, this we will certainly mention in the listing.

1. Do not use SQL

This refers to ANSI SQL DML, since many databases try to use query languages similar to the well-known favorite syntax, but it was not possible to fully implement it, and it is unlikely to succeed. Although there are rumors that startups are trying to implement SQL, for example, in Hadup.

2. Unstructured (schemaless)

The sense is that in NoSQL databases, unlike relational databases, the data structure is not regulated (or poorly typed if analogies are made with programming languages) – you can add an arbitrary field in a separate line or document without first declaring the structure of the entire table. Thus, if there is a need to change the data model, the only sufficient action is to reflect the change in the application code.

For example, when renaming a field in MongoDB:

```
BasicDBObject order = newBasicDBObject();
order.put("date", orderDate); // this field was a long time ago
order.put("totalSum", total); // we used to simply "sum"
```

If we change the logic of the application, then we expect the new field also when reading. But because there is no data schema, the totalSum field is missing from other existing Order objects. In this situation, there are two options for further action. The first is to bypass all documents and update this field in all existing documents. Due to the amount of data, this process occurs without any locks (comparable to the alter table rename column), so during an update, existing data can be read by other processes during the update. Therefore, the second option – checking in the application code – is inevitable:

```
BasicDBObject order = newBasicDBObject();
Double totalSum = order.getDouble("sum"); // This is an old model
if (totalSum == null)
totalSum = order.getDouble("totalSum"); // This is an updated model
```
[3].

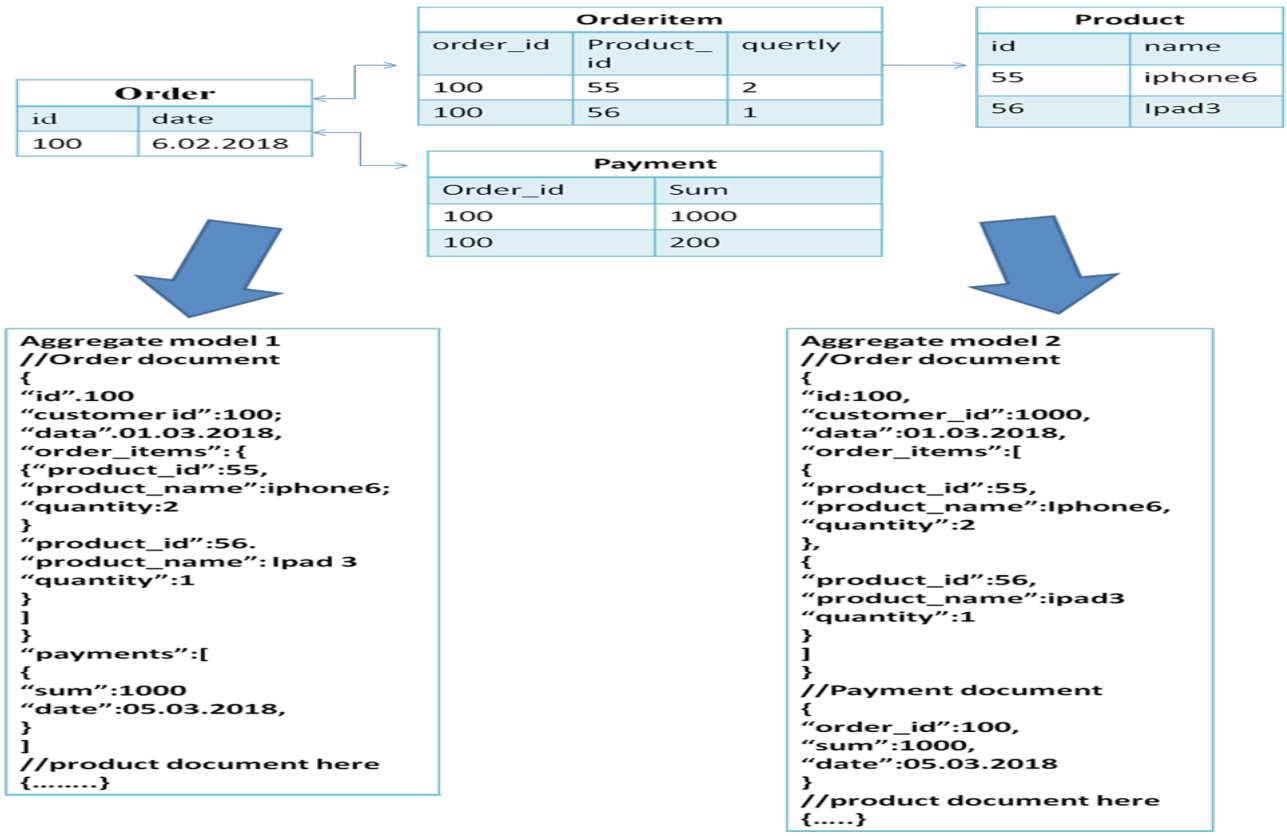
And already with the re-recording we will write this field into the database in a new format. A pleasant consequence of the lack of a scheme is the effectiveness of working with sparse data. If there is a date_published field in one document, and not in the second one, then no date_published field for the second field will be created. This is, In principle, logical, but less obvious example – the column-family NoSQL database, which uses familiar concepts of tables columns. However, due to the absence of the schema, the columns are not declared declaratively and can be changed added during the user session of working with the database. This makes it possible in particular to use dynamic columns to implement lists [4]. The unstructured scheme has its drawbacks – in addition to the above overhead in the application code when changing the data model – the absence of all possible restrictions from the database (not null, unique, check constraint, etc.), plus additional difficulties in understanding and controlling the structure data in parallel work with the database of different projects (there are no dictionaries on the side of the database). However, in a rapidly changing modern world such flexibility is still an advantage. An example is Twitter, which five years ago, together with a tweet, stored only a little extra information (time, Twitter handle and a few more meta information bytes), but now in addition to the message itself, a few more kilobytes of metadata are stored in the database [5].

Representation of data in the form of aggregates

Unlike the relational model, which preserves the logical business entity of the application in various physical tables for normalization purposes, the NoSQL repositories operate with these entities as with integral objects.

In this example, the aggregates for the standard conceptual relational model of e-commerce "order-order items-payments-product" are demonstrated. In both cases, the order is combined with the positions into one logical object, each position holding a reference to the product and some of its attributes, for example, the name (such a denormalization is necessary so as not to request the product object when retrieving the order – the main rule of distributed systems is the minimum "Joins" between objects). In one aggregate, payments are combined with an order and are an integral part of the object, in another aggregate they are placed in a separate object. This demonstrates the main rule of designing data structures in NoSQL databases – it must comply with the application requirements and be optimized to the most frequent requests. If payments are regularly

withdrawn along with the order – it makes sense to include them in a common object, but if many requests work only with payments – then it is better to put them in a separate entity. Many will argue that working with large, often denormalized, objects is fraught with numerous problems when trying to access arbitrary data requests, when the requests do not fit into the structure of the aggregates [6].

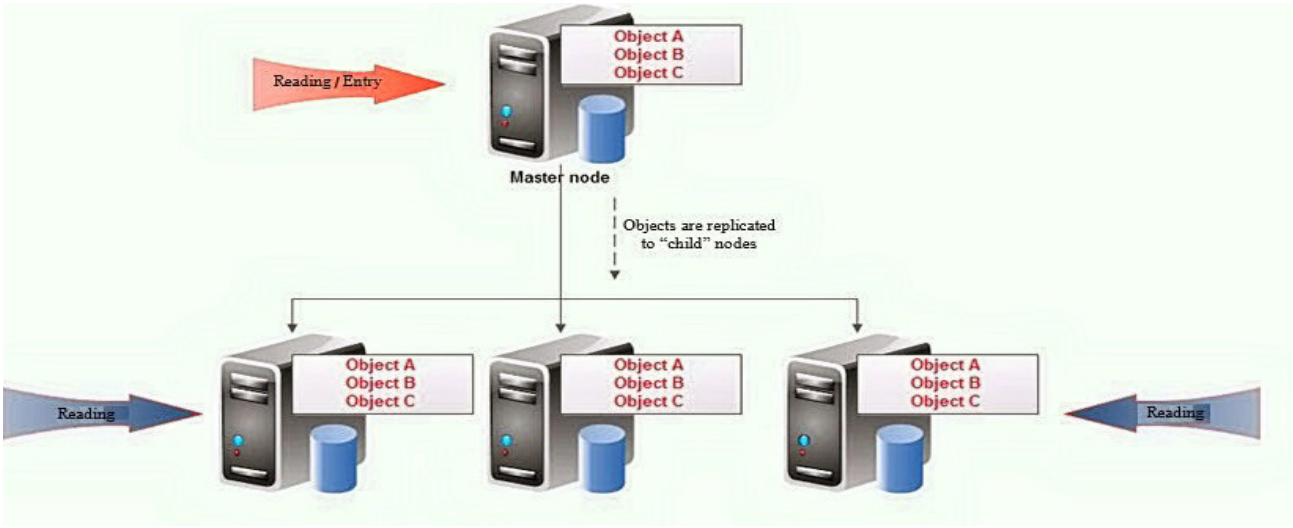


Distributed systems, without shared resources (share nothing)

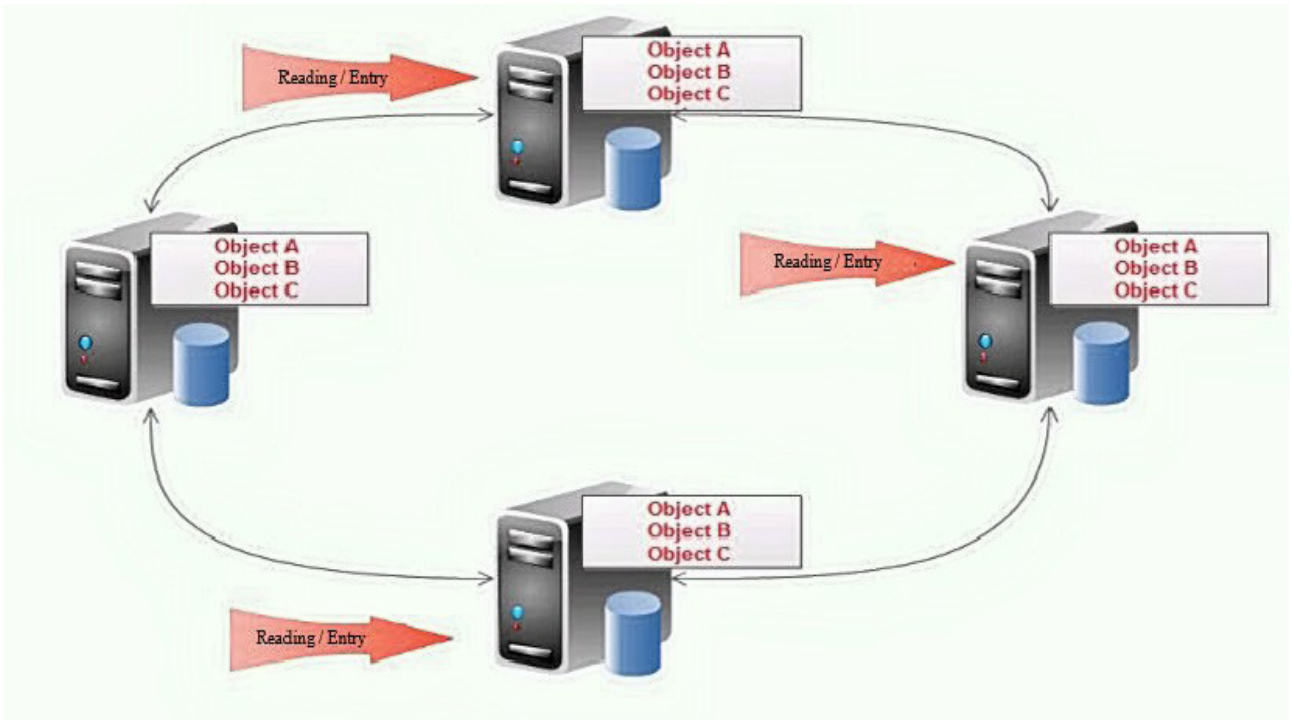
Again, this does not apply to the database graph, whose structure by definition is poorly distributed over remote nodes. This, perhaps, is the main leitmotif of the development of NoSQL databases. With the avalanche-like growth of information in the world and the need to process it in a reasonable time, the problem of vertical scalability has risen – the speed of the processor has stopped at 3.5 GHz, the speed of reading from the disk is also growing at a slow pace, plus the price of a powerful server is always greater than the total price of several simple servers. In this situation, conventional relational databases, even clustered on an array of disks, can not solve the problem of speed, scalability, and bandwidth. The only way out of this situation is horizontal scaling, when several independent servers are connected by a fast network and each owns / processes only part of the data and / or only part of the read-update requests. In such an architecture, to increase the storage capacity (capacity, response time, bandwidth), you only need to add a new server to the cluster – that's all. The procedures of shading, replication, provision of fault tolerance (the result will be obtained even if one or more servers have ceased to respond), the NoSQL database itself handles the redistribution of data in case of adding a node. Briefly I will present the main properties of distributed NoSQL databases:

Replication – copy data to other nodes during the upgrade. Allows you both to achieve greater scalability, and increase the availability and security of data. It is accepted to subdivide into two types:

Master-slave:

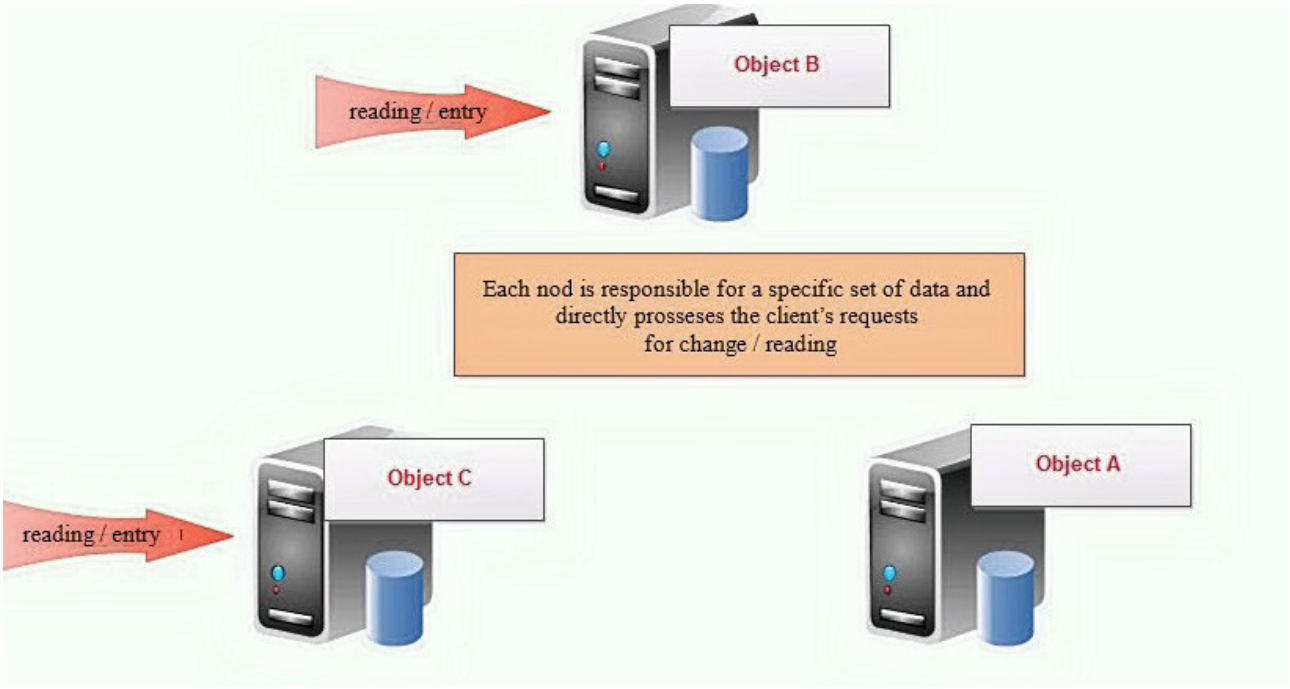


and peer-to-peer:



The first type assumes good read scalability (it can occur from any node), but an unscaled entry (only in the master node). Also there are subtleties with ensuring constant accessibility (in case of the master falling either manually or automatically in its place one of the remaining nodes is assigned). For the second type of replication, it is assumed that all nodes are equal and can serve both read and write requests.

Sharding – the division of data into nodes:



Sharding was often used as a "crutch" for relational databases in order to increase speed and throughput: a user application partitioned data into several independent databases and, when prompted by the user, accessed a specific database. In NoSQL databases, shading, like replication, is automatically produced by the database itself and the user application is separate from these complex mechanisms [7].

NoSQL databases technology

NoSQL databases technology (for example, Cassandra) is not intended to replace relational databases, but rather it helps to solve problems when the amount of data becomes too large. NoSQL often uses clusters of low-cost standard servers. This solution allows you to reduce the cost per gigabyte per second several times [8]. NoSQL databases continue to gain popularity. If five years ago they were not taken seriously , now the situation has radically changed. NoSQL databases become not just competitive, they are already leaders in projects requiring high performance. The state of the non-relational database technology has been investigated and several NoSQL databases have been categorized with respect to them: consistency, data models, replication and fulfillment of query capabilities. There are quite a few different models and functional systems for NoSQL databases [9]:

1. Warehouse key (usually store data in memory)

The key-value store works with key-value data, for example, as a dictionary. There is no place for structure or connections. After connecting to the server (for example, Redis), the application can set the key and its value, and subsequently receive these data on request. Such DBMSs are usually used to quickly store basic data, and sometimes not so basic, if you calculate the costs of the processor and memory. They are usually very fast, workable or easily scalable (it's good to use such databases for storing sessions, cache, counters, visits, etc.). As in the case of relational DBMSs, there are many open source products. Of the most popular we note memcached (and related memcachedb and membase), Voldemort, Redis and Riak .

2. Distributed storage (Column-oriented) – Cassandra, HBase, etc. (designed for very large amounts of data) . These databases work just fine by creating collections of one or more key-value pairs that in total correspond to one record. Unlike traditional tables in relational models, these databases do not require a preliminary description of the data structure. In general, distributed storage is nothing more than a two-dimensional array, where each key (record) contains one or more key-value pairs tied to it. Such a system allows you to store and use large amounts of unstructured data (one entry with a large amount of additional information). Such databases

are usually used when there are not enough simple key-value pairs, and you need to store a large amount of records with different information.

3. Document-oriented DBMS – MongoDB, Couchbase, etc. (designed to store hierarchical data structures – documents)

These databases allow much more nesting and complexity of the data structure. (for example, a document embedded in a document embedded in a document). Documents remove the nesting constraints of the first and second levels of the key-value type in distributed storages. In general, you can describe an arbitrarily complex data structure as a document and save it in such a database. Recently, in connection with the development of the Internet, search engines, social networks and highly loaded services are actively developing, which must handle large amounts of information and answer a huge number of requests. This requires not only a maximum consideration of the specifics of the information being processed, but also the transition to distributed computing. No server of any size is capable of providing the required performance [10].

Conclusion

The authors of this article in their work on the development of Web applications for processing large amounts of data take into account the above features of the functioning of databases. Including, there are three basic requirements for heavily loaded applications [11]:

• Lots of data: the largest of web applications handle data volumes of the order more than those intended for managing relational databases;

• Huge number of users: numbered in millions, access to systems simultaneously and constantly;

• Complex data: Typically, these applications are not simple processing of tabular data that can be found in many commercial and business applications.

The relational database technologies that have dominated the IT industry since 1980 began to show their weaknesses in the transition to web scales in these three aspects, so a growing number of people began to look for an alternative. Such an alternative became NoSQL database. In the advantages of using MongoDB, such as deep query capability, simple scalability, document-based storage, we made sure of the process of working on the web application “Electronic library for students, teachers and researchers” using MongoDB, NodeJS, PhpStorm [12]. The results of this research are used by the authors of this article to further simulate the processing of large amounts of data and develop a Web application.

References

1. From SQL to NoSQL and back [Electronic resource]. – Access mode: http://www.osp.ru/os/2012/02/13014127//, 2014. – P. 114-118.

2. Balakayeva G.T., Nurlybayeva K. Simulation of Large Data Processing for Smarter Decision Making. AWER Procedia Information Technology & Computer Science, 3rd World Conference on Information Technology (WCIT-2012). – 2013. – Vol. (03). – P.1253-1257

3. Redmont E., Uilson R. Seven Database in Seven Weeks: A Guid to Modern Databases and the NoSQL Movement. Publishing house “DMK Press”. – 2018. – P. 418-420. ISBN: 978-5-97028-455-2.

4. Pramod J., Sadalage, Martin Fowler. NoSQL Distilled. Publishing house “Moskva, Sankt-Peterburg-Kiev”. – 2013. – P. 135-140.

5. Gaurav Vaish .Getting Started with NoSQL. Packt books-Packt Publishing. – 2013. – P. 127-137. ISBN: 978-1849694988

6. Dan Sullivan. NoSQL for mere Mortals. Published by John Wiley&Sons, Inc.2014. – P. 321-324.

7. Adam Fowler. No SQL for Dummies. – 2015. – P. 231-237. ISBN-13: 978-1118905746

8. Guy Harisson. Next Generation Data bases. – 2016. – P. 245-247. ISBN-13: 978-1-48-42-1329-2.

9. Neal Leavitt. Will NoSQL Databases Live Up to Their Promise? Computer, 43:12–14, February 2010. – P. 128-132.

10. Gantz John, Reinsel David. The digital universe in 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East. URL: http:// www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf (Date duration 10.07.2014)

11. Balakayeva G.T., Melisova A.E. Modeling large data with NoSQL. Nauka i Studia. – 2017. – Vol. 8(169). – P. 35-37.

12. Stefan Edlich. NoSQL Databases, Available at http://nosql-database. – 2011. – P.105-107.