Communication papers of the Federated Conference on Computer Science and Information Systems, pp. 177–182
DOI: 10.15439/2017F195
ISSN 2300-5963 ACSIS, Vol. 13

Business Intelligence Platform for Big Data based on Scalable Distributed Two-Layer Data Store

Adam Krechowicz
Kielce University of Technology, Poland
Email: a.krechowicz@tu.kielce.pl

Stanisław Deniziak
Kielce University of Technology Poland
Email: s.deniziak@tu.kielce.pl

***Abstract*—Data mining is one of the main business intelligence technique. The volume of Big Data expands in such a way that the classical Business Intelligence methods need to be redefined. First, the organization of large data sets requires new distributed database architectures. Second, it is necessary to develop distributed data processing models that provide a high degree of scalability.In this paper we introduce fully scalable BI platform that is suitable for the most common data processing issues. The platform is based on our scalable distributed two-layer data store, which is competitive to existing NoSQL distributed data base systems. We show examples and experimental results showing advantages of our approach.**

I. Introduction

BUSINESS Intelligence (BI) is very important branch of modern analytic in many companies. Analysing gathered data (like information about company's clients) can be a huge source of essential knowledge. Because of that companies can offer their services on higher level and can multiply their profits. There are many existing BI tools available to use out of the box. Some of them, which offers basic functionality, are even available free of charge. The prices of others, which offer advanced techniques of data processing, can be very high.

From the basic principles of statistics we know that to achieve more accurate data analysis, the set of analysed data should be really big. Unfortunately processing that huge data sets is still a great challenge. The popularity of Big Data nowadays is a reflection of this trend. There is still a wide pressure on developing efficient algorithms that are suitable for processing huge data sets.

Scalable Distributed Two-Layer Data Store (SD2DS) [1] is a very powerful data store that was developed in our research team. The ease of its scalability gave us the opportunity to develop an efficient BI platform that can process data distributed on nodes in the cluster. In this paper we present the prototype of such a system.

This paper is organized as follows. Firstly, we present the existing BI platforms for data analysis. Then, we discuss the demand of creating this novel platform. In the next section we introduce the basic concept of SD2DS. Next, we present key architectural elements of our platform which is evaluated in the next section. The paper ends with conclusions.

II. Related Work

The Business Intelligence have a long history. For many years the basic tools were developed as stand alone systems that run on a standalone computer. For many people the first choice to create some kind of data analysis is to use Microsoft Excel or its open-source equivalent like Libre Office or Open Office. Among many other tools the QlikView [2] is worth highlighting. It allows to create advanced and responsive data analysis. That kind of programs are suitable for most typical applications.

On the other hand, there are many commercial platforms provided by the companies such as Oracle [3], SAP [4] or IBM [5]. They offer BI tools as a part of their whole enterprise management systems. In the vast majority they are oriented as client–server systems. In most cases they offer custom designed solution for specific needs.

Contemporary BI tools use advanced data mining techniques, expert systems and computational intelligence methods, but still simple methods for analysing large data sets are needed. Distributed storage and processing is required to ensure the required performance of Big Data analysis.

The truly distributed data analysis can be developed based on the framework such as Hadoop [6] which is the open source implementation of a very popular Google MapReduce [7] system. On the contrary to the previously presented tools, it can really process big data. There are many existing scenarios for BI based on Hadoop solutions [8], [9], [10]. The Hive [11] processing engine can be considered as one of the most recognizable examples of such systems.

III. Motivation

All of the systems presented in the previous section have their limitations. The stand alone systems are obviously not suitable for huge data sets. For example Libre Office allows to load only a finite number of data. In the case of the QlikView system, authors of this paper suffered themselves the "Out of Memory" error while processing huge data sets many times.

Commercial products are, in vast majority, based on the typical relational data bases which have their serious limitations. Those limitations are well known and are widely discussed [12], [13]. They were one of the main reason for expansion of NoSQL data storages.

The MapReduce framework tends to eliminate these drawbacks. However, it does not provide a complete tool to work with. It rather provide environment for designing custom solutions [7]. Their efficiency is strongly correlated with the programmers skills due to the fact that designing such

distributed algorithms is not intuitive in comparison with programming single process programs.

All described above problems concerning processing big data were the driving force behind designing distributed and efficient tool for Business Intelligence. As a base of our platform we have chosen Scalable Distributed 2-Layer Data Store because it is a very effective architecture for storing really huge data.

IV. Scalable Distributed Two-Layer Data Store

The Scalable Distributed Two-Layer Data Store (SD2DS) is an efficient NoSQL key-value database, which has been designed by the authors for the last couple of years. Its main goal is to store huge data sets that are too big to be efficiently stored by typical relational databases. It allows to run in distributed environment and is capable to scale its storage capacity almost infinitely.

On the base of the SD2DS the very mature standard of Distributed Linear Hashing (LH^*) [14] was utilized. It organizes the data in so called *buckets* which are located on nodes in a cluster. Each bucket is responsible for storing some set of data portions (*components*). All of the buckets are distributed on a nodes in a cluster. The main advantage of the LH^* is the ease of addressing the components without any central directory while preserving full data scalability. Each component consists of two parts: *headers* and *bodies*. The component body is the data itself while the component header consists of metadata that allows to manage the bodies. The LH^* mechanism is responsible for managing the first layer of the store which is responsible for storing headers. The bodies are stored in the second layer of the store. That layer division was firstly introduced to eliminate the main drawback of the single layer store, namely the need of constantly reorganizing the whole structure [15].

The example architecture of SD2DS is presented in Fig. 1. The first layer consists of the set of headers while the second layer consists of the set of bodies. If a client wants to access specific component (component number 2 in the figure) it first needs to access specific header in the first layer. The appropriate first layer bucket, that contains desired header, is found based on the $h(key)$ function according to the LH^* scheme. The locator inside header is used then to access the corresponding body. When a client wants to insert data item into SD2DS it first needs to insert appropriate data header. The first layer is responsible for allocating the space for the body in the second layer. This indirect access gave us opportunity to introduce additional functionality like throughput scalability [16] or fault tolerance.

The Fig. 2 and Fig. 3 present the time of accessing the data in SD2DS in comparison to the most recognizable representatives of the NoSQL systems: MongoDB [17] and MemCached [18]. The overall evaluation was presented in [1] and [19]. The MongoDB and MemCached was chosen because they have many similarities with SD2DS. First of all, all of those systems were developed using C++ language and allowed to store the data in distributed environment. Secondly, they all allow to extensively use main memory of machines to speed up the all operations. Both figures 2 and 3 present the results of getting the data portions in the relation to the number of clients that simultaneously operate on the system. Figure 2 presents the results of getting data portions of 5MiB while figure 3 presents the results of getting 10MiB data portions.

In all of these two cases the best results were achieved for SD2DS. The times for MongoDB are strongly correlated to the number of the *mongos* elements. The mongos elements are responsible for properly addressing the data portions in MongoDB system. Because the clients does not know the exact location of the data it has to direct their queries to the central element that is aware of the current configuration of the system [17]. The SD2DS is free of that kind of drawback. Our SD2DS proved to have better performance even in comparison with the MemCached system. The MemCached is optimised mostly for the small data portions. It does not do best with the data portions that are greater that 1MiB. It also does not work well under the heavy clients load when it just rejects clients after specified time [20].

In its basic form SD2DS stores the data in raw format which does not use any schema at all. Because of that its basic application was storing multimedia data [21], [22], [23] like photos or videos. We also successfully utilized our data store for gathering data from advanced Internet of Things system [24] and used it as a base for anonymous storing the data in the Cloud environment [25], [26].

V. Architecture of SD2DS-based BI platform

In this section we presented the essential parts of the architecture of our BI platform. The main goal to face with was to develop an efficient processing model that can process the data effectively in the distributed environment. Due to the fact that in our previous work we mostly used raw portion of the data we also needed to develop a simple yet effective data model to begin with.

A. Data model

In the original conception of SD2DS bodies were just a block of data that have no structural form at all. Hence, in our previous work, we used it mostly for large data portions like high resolution photos and videos. To efficiently store structural data we needed to develop a method of storing a set of structured data into a single body. We decided to introduce simple data schema in which the whole set of data is divided into *records*. Each record was then divided into *dimensions*. Because of the nature of the most data and also simplicity we assumed that all records had the same dimensions. Additionally we assumed that the number of dimensions (n) was constant to all records. In that form all of the bodies could consist of a separable subset of all records (so called *block*). The Table I illustrate this concept.

This data model gave us opportunity to fast access value of any dimension (i) in record (j) within specified block (b) simply by utilizing the following equation:

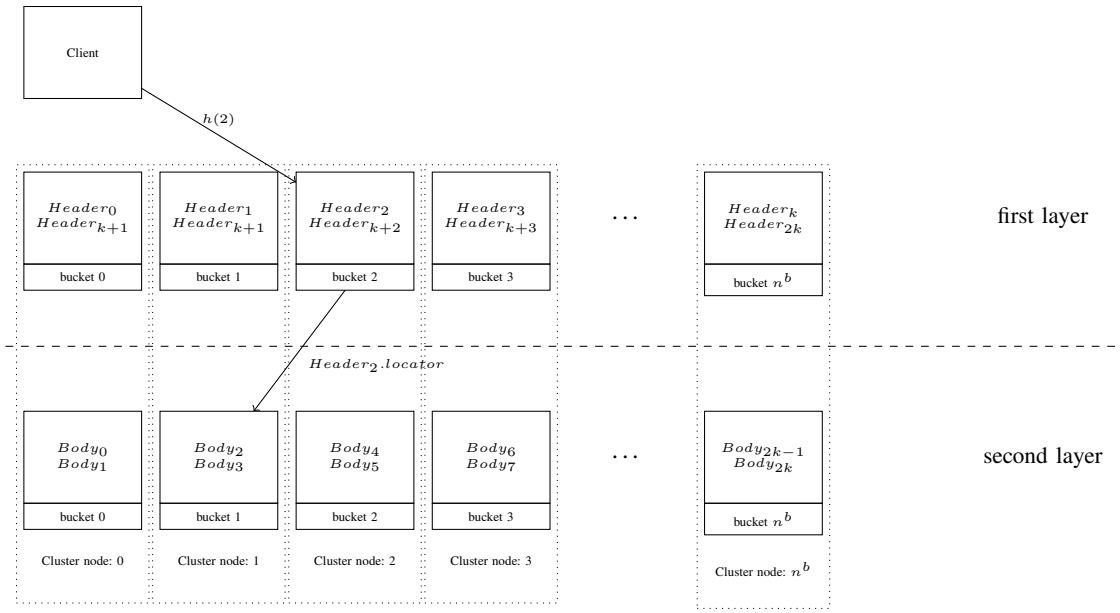

Fig. 1. Architecture of SD2DS

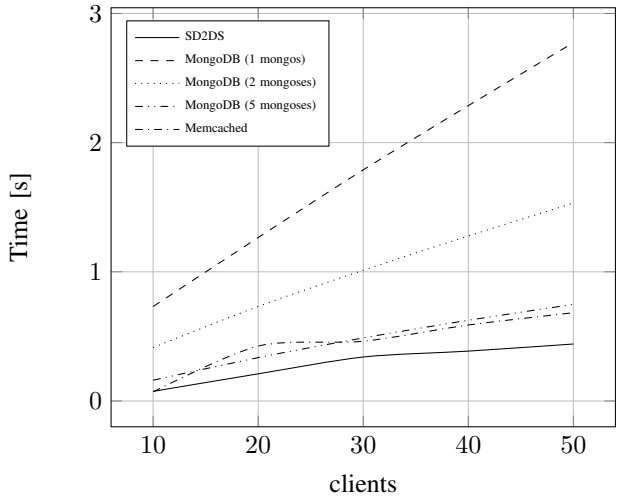

Fig. 2. Time comparison of getting components of fixed (5MiB) size [19]

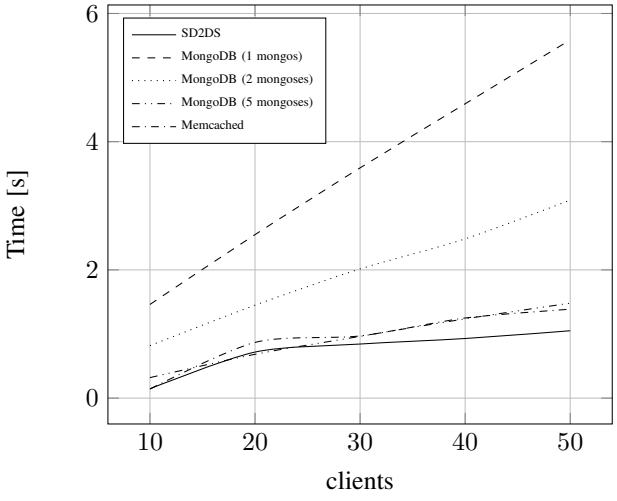

Fig. 3. Time comparison of getting components of fixed (10MiB) size [19]

TABLE I
DATA MODEL IN SD2DS

$dimension_1$	$dimension_2$	...	$dimension_n$
$value_{1,1}$	$value_{2,1}$	...	$value_{n,1}$
$value_{1,2}$	$value_{2,2}$	...	$value_{n,2}$
...	...	...	...
$value_{1,m}$	$value_{2,m}$	...	$value_{n,m}$

$$value(i, j, b) = block_b[j * n + i] \quad (1)$$

where:

- b – block number
- j – record number in b–th block
- i – dimension number in j–th record in b–th block

The main advantage of this model is that it can access each value within the block in constant time regardless of the number of dimensions and the number of records within block.

B. Processing model

The natural distribution of blocks within buckets creates a great opportunity to distribute the processing on all buckets within SD2DS. All buckets are responsible for processing blocks that are stored within. Because all bodies are accessible through the first layer there are used to aggregate the results given from the second layer. The partially aggregated results

from the first layer are then fully aggregated by the client. This creates the processing model which requires to define two functions:

- $f_b(b, \ldots)$ – Block function
- $f_a(x, y)$ – Aggregate function

The $f_b(b, \ldots)$ function is responsible for processing specified block of data (b). Depending of the nature of the processing it may require additional parameters. The results of all $f_b(b, \ldots)$ functions are then used as a parameters of the $f_a(x, y)$ functions which aggregates the partial results. The $f_b(b, \ldots)$ function is executed on the second layer buckets while $f_a(x, y)$ is executed both on the first layer buckets and by clients. This creates a processing model similar to one introduced in MapReduce framework [7].

The results of both $f_b(b, \ldots)$ and $f_a(x, y)$ has to produce the result in the same form. Additionally aggregate function should work with only one parameter in such a way that:

$$f_a(f_b(b_1), null) = f_a(null, f_b(b_1)) = f_b(b_1) \tag{2}$$

Additionally, because results of different blocks can be executed in different order the following two constraints should be satisfied:

$$f_a(f_b(b_1), f_b(b_2)) = f_a(f_b(b_2), f_b(b_1)) \tag{3}$$

$$\begin{aligned} &f_a(f_a(f_b(b_1), f_b(b_2)), f_b(b_3)) = \\ =&f_a(f_a(f_b(b_2), f_b(b_3)), f_b(b_1)) = \\ =&f_a(f_a(f_b(b_3), f_b(b_1)), f_b(b_2)) \end{aligned} \tag{4}$$

The sample architecture and the execution model of defined functions are presented in Figure 4. In this example SD2DS consists of three first layer buckets and three second layer buckets. Each second layer bucket consists of three blocks (bodies). First second layer bucket consists of blocks b_1, b_4 and b_7, second bucket consists of blocks b_2, b_5 and b_8 while the third bucket consists of blocks b_3, b_6 and b_9. To accomplish processing, each second layer bucket executes $f_b(b, \ldots)$ function on all blocks for which it is responsible. Then the results of block functions are passed to appropriate first layer bucket which executes the $f_a(x, y)$ to aggregate the results from second layer buckets. Next, the results from all first layer buckets are transferred to the client which executes again $f_a(x, y)$ functions for each intermediate results.

C. Example: Finding Maximum Value

For better understanding of the role of the $f_b(b, \ldots)$ and $f_a(x, y)$ functions we present an example calculation of the maximum value of the i-th dimension in all records. The algorithm 1 presents the block function for finding max value while algorithm 2 presents the aggregate function.

The function $f_b^{max}(b, i)$, like all other block function requires the b parameter which indicates the block that operates on. Additional parameter i determines the dimension on which the maximum value is searched. In its basic form it searches on all records within the blocks but can easily be modified to process only specific records that follow user defined constraints.

Algorithm 1 $f_b^{max}(b, i)$ – Block function for finding maximum value

```
result ← null;
r ← 0;
while r ≤ m do
    if result = null ∨ result < value(i, r, b) then
        result ← value(i, r, b);
    end if
    r ← r + 1;
end while
return result
```

Algorithm 2 $f_a^{max}(x, y)$ – Aggregation function for finding maximum value

```
if x > y then
    return x
else
    return y
end if
```

The $f_a^{max}(x, y)$ function determine the maximum value from the two output of block functions, from other aggregate functions or from one block and one aggregate function. It simply determines the maximum value from two intermediate values. Its simplicity is very important because it is also executed by the client for every intermediate results obtained by the all first layer buckets. It is also important to not change the first layer buckets which are responsible for serving other clients requests.

VI. Use Case

To evaluate our platform we designed a special implementation that gave us an opportunity to visualise the exchange rates of different currencies in relation to the "polish złoty" (PLN). The data about the exchange rates was taken from the [27]. Although the data set was relatively small (approx. 250 records per year) it allowed us to evaluate the correctness of the calculation by comparing it with different standalone tools. The data consisted of 38 dimensions. Each record represented the exchange rates of 35 different currencies for the specified day. The 3 additional dimension was used to represent the date (one dimension per year, month and day). The user interface of the prepared tool is presented in figure 5. It allowed to create visualization data aggregated in SD2DS environment.

The figures 6–8 present the results of executing different queries of the different subsets of the whole data set. The figure 6 shows the average exchange rate of United State Dollar in the months of the year 2014. The figure 7 presents the maximal exchange rate of Great British Pound that was obtained in the evaluated years. At last figure 8 presents the average exchange rate of Euro of specified days in each month of the year 2012.

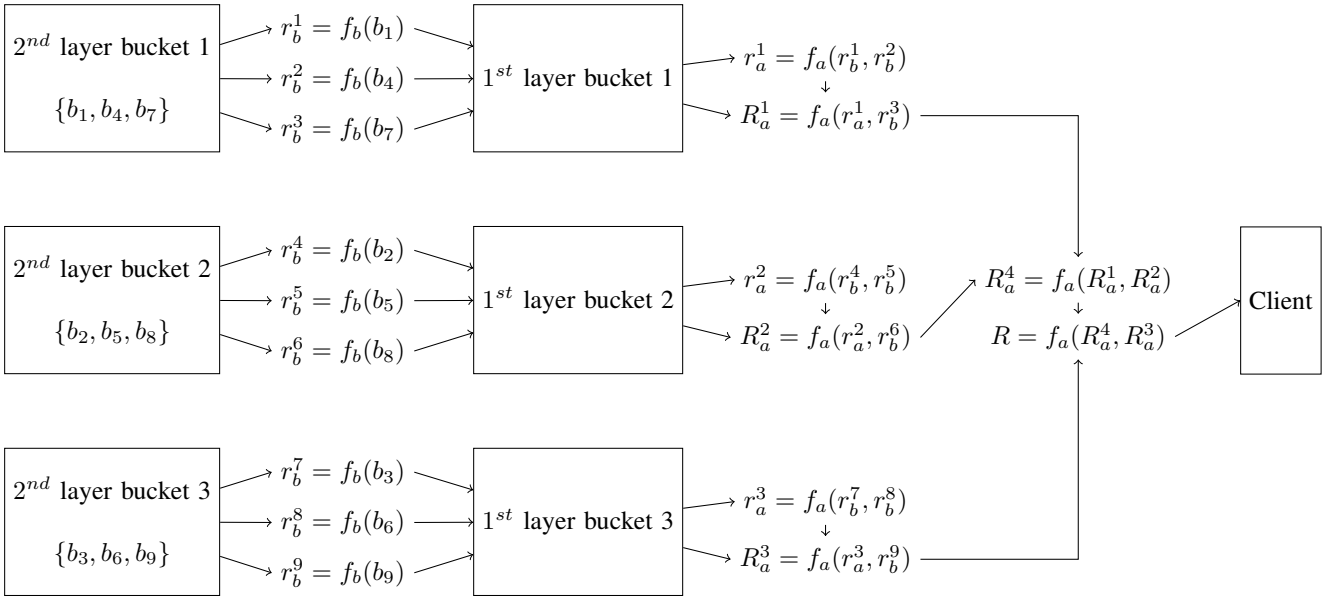

Fig. 4. Architecture of tool

Fig. 5. User interface

Fig. 6. Average value of United States Dollar per month in the year 2014

Fig. 7. Maximum value of Great British Pound per year

VII. Experimental Results

The exchange rate use case presented in section VI did not allow to evaluate the efficiency of the platform in relation to the size of the overall dataset. To determine the efficiency in that matter we used synthetic dataset that contained of the same value in all dimensions in all records. We use 16 nodes of cluster for the SD2DS buckets. Each node consists of one first layer bucket and one second layer bucket. Additional node was used for client application. In all cases each block consisted of 1,000 records. The time given in all experiments was the average value of 1,000 tries.

In our first evaluation we measured the processing time of 16 blocks with different number of dimensions. The results are presented in figure 9. We measured the processing time for blocks that contained 1 to 100 dimensions. The obtained results of processing time were independent of the number of data dimensions. It oscillated from 0.008 to approximately 0.0083 second for all cases regardless on the dimensions number.

The goal of the next experiment was to evaluate the processing time in relation to the number of data records. We evaluated our tool with the records number from 1,000 to 128,000. The results are presented in figure 10. The overall figure can be divided into three sections. The first section (from 1,000 to 16,000 records) presents the situation where the processing time drastically increased. It was caused by the fact that the processing was not distributed on the whole set of buckets. Because of that the the client waits for reduced number of buckets to response and the number of execution of aggregate function is also reduced. The second section (from

Fig. 8. Average value of Euro in days of the month in the year 2012

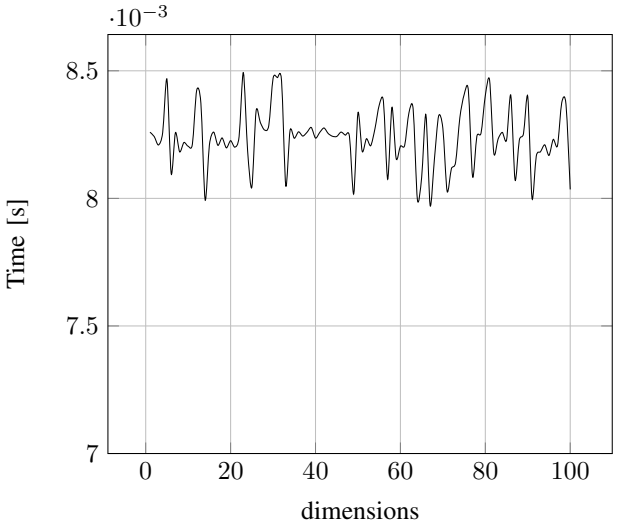

Fig. 9. Processing time in relation to the number of data dimensions

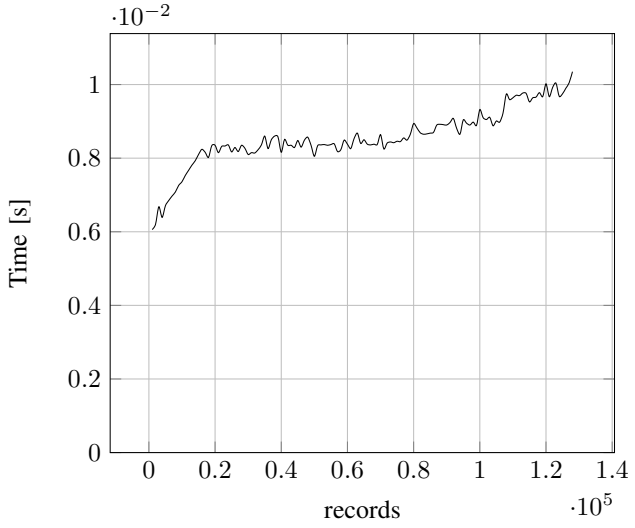

Fig. 10. Processing time in relation to the number of data records

16,000 to approx. 70,000) presents the situation where the processing time was very similar in each case. It represents the situation where all of the data were properly distributed on all buckets within the cluster. In the third section (from approx. 70,000 and above) the processing time started to increase slightly. It represents the situation when the buckets became heavy loaded and additional portions of data increased the overall processing time. This situation indicated the need to introduce additional buckets to the structure to ensure scalability.

VIII. Conclusions

In this paper we introduced our SD2DS based BI platform that is suitable for processing Big Data sets. We chose our Scalable Distributed Two-Layer Data Store for this purpose because of its efficiency that had been established in our previous work. We introduced a special data model that gave us the opportunity to take advantage of its high performance to achieve our goal. We also proposed a special processing model which allowed to distribute the computation on the nodes in the cluster on similar way than MapReduce model. The preliminary experiments, carried out in this paper, are very promising and we are planning to develop truly scalable BI platform as our future work.

References

[1] A. Krechowicz, S. Deniziak, M. Bedla, A. Chrobot, and G. Łukawski, "Scalable distributed two-layer block based datastore," in *International Conference on Parallel Processing and Applied Mathematics*. Springer International Publishing, 2015, pp. 302–311.

[2] M. García and B. Harmsen, *Qlikview 11 for developers*. Packt Publishing Ltd, 2012.

[3] Oracle, "Oracle business intelligence 12c," https://www.oracle.com/solutions/business-analytics/business-intelligence/index.html, 2017, accessed 3rd April 2017.

[4] SAP, "Business intelligence (bi) tools & software," https://www.sap.com/solution/platform-technology/analytics/business-intelligence-bi.html, 2017, accessed 3rd April 2017.

[5] IBM, "Business intelligence," https://www.ibm.com/business-intelligence, 2017, accessed 3rd April 2017.

[6] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash *et al.*, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 1071–1080.

[9] S. Chen, "Cheetah: a high performance, custom data warehouse on top of mapreduce," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1459–1468, 2010.

[10] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in hadoop mapreduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2014–2015, 2012.

[11] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[12] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191*, 2013.

[13] C. J. Tauro, S. Aravindh, and A. Shreeharsha, "Comparative study of the new generation, agile, scalable, high performance nosql databases," *International Journal of Computer Applications*, vol. 48, no. 20, pp. 1–4, 2012.

[14] W. Litwin, M.-A. Neimat, and D. A. Schneider, *LH*: Linear Hashing for distributed files*. ACM, 1993, vol. 22, no. 2.

[15] K. Sapiecha and G. Lukawski, "Scalable distributed two-layer data structures (SD2DS)," *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 4, no. 2, pp. 15–30, 2013.

[16] K. Sapiecha, G. Łukawski, and A. Krechowicz, "Enhancing throughput of scalable distributed two–layer data structures," in *Parallel and Distributed Computing (ISPDC), 2014 IEEE 13th International Symposium on*. IEEE, 2014, pp. 103–110.

[17] E. Plugge, T. Hawkins, and P. Membrey, *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*, 1st ed. Berkely, CA, USA: Apress, 2010. ISBN 1430230517, 9781430230519

[18] J. Petrovic, "Using memcached for data distribution in industrial environment," in *Systems, 2008. ICONS 08. Third International Conference on*. IEEE, 2008, pp. 368–372.

[19] A. Krechowicz, A. Chrobot, S. Deniziak, and G. Łukawski, "SD2DS-based datastore for large files," in *Federated Conference on Software Development and Object Technologies*. Springer, 2015, pp. 150–168.

[20] Memcached, "Memcached – A Distributed Memory Object Caching System," http://memcached.org, accessed 3rd April 2017.

[21] S. Deniziak, T. Michno, and A. Krechowicz, "The scalable distributed two-layer content based image retrieval data store," in *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE, 2015, pp. 827–832.

[22] ——, "Content based image retrieval using modified scalable distributed two-layer data structure." *International Journal of Computer Science & Applications*, vol. 13, no. 2, 2016.

[23] T. Michno and A. Krechowicz, "SD2DS database in the direction of image retrieval," *Applications of information technologies - theory and practice*, vol. 11, 2015.

[24] S. Deniziak, G. Łukawski, M. Bedla, and A. Krechowicz, "A scalable distributed 2-layered data store (SD2DS) for internet of things (IoT) systems," *Measurement Automation Monitoring*, vol. 61, no. 7, pp. 382–384, 2015.

[25] A. Krechowicz, "Scalable distributed two-layer datastore providing data anonymity," in *International Conference: Beyond Databases, Architectures and Structures*. Springer, 2015, pp. 262–271.

[26] A. Krechowicz and S. Deniziak, "SD2DS-based anonymous datastore for IoT solutions," *DEStech Transactions on Computer Science and Engineering*, no. wcne, 2016.

[27] Narodowy Bank Polski, "Statistics and reporting," http://www.nbp.pl/home.aspx?f=/statystyka/kursy.html, accessed 3rd April 2017.