

Review scientific paper/Pregledni naučni rad

A REVIEW ON BIG DATA OPTIMIZATION TECHNIQUES

PREGLED BIG DATA OPTIMIZACIJSKIH TEHNIKA

Vedrana Nerić¹, Nermin Sarajlić²

Abstract: Analysis of representative tools for SQL query processing on Hadoop (SQL-on-Hadoop systems), such as Hive, Impala, Presto, Shark, show that they are not still sufficiently efficient for complex analytical queries and interactive query processing. Existing SQL-on-Hadoop systems have many benefits from the application of modern query processing techniques that have been studied extensively for many years in the database community. It is expected that with the application of advanced techniques, the performance of SQL-on-Hadoop systems can be improved. The main idea of this paper is to give a review of big data concepts and technologies, and summarize big data optimization techniques that can be used for improving performance when processing big data.

Keywords: Big Data, Hadoop, SQL-on-Hadoop, Hive, optimization

Sažetak: Analiza reprezentativnih alata za obradu SQL upita na Hadoop platformi (SQL na Hadoop-u sistema), kao što su Hive, Impala, Presto, Shark, pokazuju da oni nisu još uvijek dovoljno efikasni za složene analitičke upite i interaktivnu obradu upita. Postojeći SQL na Hadoop-u sistemi imaju puno koristi od primjene savremenih tehnika za obradu upita koje su u velikoj mjeri proučavane dugi niz godina u zajednici baza podataka. Očekuje se da se primjenom naprednijih tehnika, performanse SQL na Hadoop-u sistema mogu poboljšati. Glavna ideja ovog rada je dati pregled Big Data koncepta i tehnologija, i sumirati Big Data optimizacijske tehnike pomoću kojih se mogu poboljšati performanse prilikom obrade velikih količina podataka.

Ključne riječi: Big Data, Hadoop, SQL na Hadoop-u, Hive, optimizacija

INTRODUCTION

Over the past decades, dominant solutions in working with data are traditional relational database management systems (RDBMS), which are based on a relational model where data and relationships between data are stored in tabular form. Structured query language (SQL) is used when working with relational databases. These systems are only applicable to structured data and cannot handle heterogeneity and a large amount of data that is today in the rank of even exabytes and zettabytes.

Big data is a term that refers to a large amount of different data that is collected at high speed, and also to technologies for storing, searching, processing, and analyzing data. Big data is often described with 3Vs dimensions [1]:

- Volume – large amount of data that is collected, processed and made available for analysis,
- Velocity – the high speed at which new data arrives and speed at which that data needs to be processed,
- Variety – diversity of data available in various forms and sources, most commonly unstructured.

Due to the RDBMS limitations in terms of support for big data, there has been an emergence of new technologies and highly scalable and flexible systems for storing data. The concept of non-relational (NoSQL – Not Only SQL) databases was created to cope with the rapid growth of a large amount of unstructured data. Originally closed source NoSQL databases were developed by large companies for their specific needs, such as BigTable (Google) and DynamoDB (Amazon). The success of these systems has triggered the development of some similar open-source databases such as MongoDB, Hypertable, HBase, Cassandra, Redis, and others. Non-relational databases can be divided into the following categories: key-value, column-based, document-based and graph-based databases. [2]-[7]

One of the important areas of research where big data provides opportunities and challenges is query optimization. Database management systems strive to process the query in the most efficient way to obtain the best results.

The main aim of this paper is a review on big data optimization techniques. The paper is organized as follows. After the introduction, Chapter 1 describes big data technologies. Big data optimization techniques are described in Chapter 2. The final chapter provides concluding remarks and possible guidelines for future research.

¹ Virgin Pulse, Tuzla, Bosnia and Herzegovina

² Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina

vedrana_neric@yahoo.com

Paper submitted: June 2020 Paper accepted: August 2020

1. BIG DATA TECHNOLOGIES

Hadoop is a set of technologies closely related to big data that form a powerful solution through data storage, data processing, system management, and integration of other modules. Hadoop is an open-source framework of Apache foundation that consists of four parts: Hadoop Common (set of libraries and configuration files), HDFS (Hadoop Distributed File System for data storage in a cluster), MapReduce (model for data processing), and YARN (Hadoop operating system for resource allocation and job management). In addition to the mentioned components, Hadoop relies on specialized tools for data collection (Flume, Kafka, Sqoop), data processing (Hive, Pig, Storm), management (Ambari, Falcon). [8]-[13]

SQL-on-Hadoop is a group of analytical tools that combine SQL querying with Hadoop elements. Hive data warehouse is one result of combining SQL and Hadoop with HiveQL for translating SQL queries into MapReduce jobs. SQL-on-Hadoop solutions enable processing big data stored on Hadoop with a familiar SQL interface instead of complex programming languages. Hive, Impala, Presto, Stinger, Shark, BigSQL, Drill, H-SQL, HAWQ, Hadapt, JethroData, PolyBase, Stinger, Splice Machine, Spark are some of SQL-on-Hadoop systems. [14]-[17]

One of the most developed and stable big data SQL frameworks is Hive. It was developed by Facebook to handle RDBMS restrictions when huge amounts of data emerged. Apache Hive enables easier big data processing on Hadoop. Hive applies the structure on Hadoop data and enables queries using SQL-like language called HiveQL. Custom mappers and reducers can be attached to HiveQL to express the complex logic of data processing. [18]-[20]

2. BIG DATA OPTIMIZATION TECHNIQUES

Although Hadoop/Hive scales nearly every amount of data, optimization needs to be done to process data quickly and to efficiently execute queries. There are Hive query optimization techniques available when executing Hive queries to improve Hive performance [21]-[30]:

- Tez Execution Engine:

Apache Tez is an application framework built on Hadoop Yarn for batch and interactive data processing that is used to execute queries faster. Tez is more flexible and more efficient than MapReduce for interactive queries. It fastens query execution time to 1-3 times. Using Tez, the response time is more efficient due to less job splitting and HDFS access, while using MapReduce, task is divided in more jobs and also HDFS accessing is more. Only the final result is stored to the HDFS after completion of all map and reduce tasks. Complex directed acyclic graphs (DAGs) can be processed as a single Tez job instead of multiple MapReduce jobs. The default execution engine is MapReduce and Tez engine can be enabled by setting the property:

```
set hive.execution.engine = tez;. [31]-[35]
```

- Vectorization:

Vectorization improves performance for operations such as scanning, filtering, aggregation, join, by retrieving 1024 rows in one operation. With vectorization, Hive can process a batch of rows instead of processing one row each time. Each batch consists of a column vector (an array of a primitive data type) and operations are performed on the entire vector, which improves cache usage. Vectorization significantly improves query execution time and can be enabled with two parameters settings:

```
set hive.vectorized.execution.enabled = true;
set hive.vectorized.execution.reduce.enabled = true;. [36], [37]
```

- Parallel execution:

MapReduce jobs on Hadoop can be executed in parallel and several executable queries on Hive automatically use this parallelism. In this way the overall execution time of the query can be reduced. During execution the queries can be converted into different stages (MapReduce stage, sampling stage, merge stage, limit stage, etc.), usually executed one after the other. Parallel execution in Hive enables these independent stages to be processed parallel to save overall job running time. This feature can be set with the following parameter:

```
set hive.exec.parallel = true;. [19]
```

- Compression:

Compression techniques significantly reduce the amount of data transferred, reducing the transfer of data between mappers and reducers. Compressed file size should not be more than a few hundred MB. Some of the Hive compression algorithms are gzip, bzip, lzo, snappy. It depends on the situation which compression algorithms will be suitable. Compression techniques, that best suit needs, should be chosen based on Hadoop infrastructure and data. Compression at the mapper and reducer side can be set using code:

```
set mapred.compress.map.output = true;
set mapred.output.compress = true;. [19]
```

- File format:

Some file formats, such as JSON, text type of format, are not a good choice for large systems with high data volume because they take a lot of space. Query performance can be drastically increased using an appropriate file format based on data. ORC (Optimized Row Columnar) format enables efficient ways to store Hive data by reducing the size of the original data by up to 75% and increases data processing speed. ORC was designed to overcome the limitations of other Hive formats (Text, Sequence, RCFILE) and gives better performance when it comes to reading, writing, and processing the data. ORC takes less time than RCFILE to access data and takes less space to store data. It supports compressed (ZLIB, Snappy) as well as uncompressed storage. Here is the syntax for using ORC file format for a new Hive table during creation:

```
CREATE TABLE table_name
(column1 type, column2 type,...)
STORED AS ORC;. [19], [38]
```

- Sampling:

Sampling allows users to take part of a data set and analyze it without having to analyze the entire data set. If a representative sample is used, then a query can return significant results, complete faster and use less computing resources. Sampling is most efficient for testing and auditing purposes. To use the sampling feature in Hive it is necessary to use TABLESAMPLE clause which helps in writing queries on a sample of data instead of the entire table that contains a large amount of data. There are multiple ways of sampling in Hive:

- sampling bucketed table – is used when it is needed to sample a particular out of a group of buckets or the total number of buckets,
- block sampling – deals in the percentage of data that needs to be sampled,
- length literal – is a way of sampling data or rows from a large data set where it is specified size or length of the input data,
- row count – is a type of sampling where it is provided the number of rows as the input. [20]

- Partitioning:

Hive reads all the data in the table directory and applies the query filters on it. That is slow and expensive because all data has to be read. With partitioning, data is stored in separate individual partitions (subdirectories) on HDFS. Instead of querying the entire dataset, only looking at the required partition reduces the query execution time to get results. This is sometimes known as horizontal partitioning. Splitting a table into partitions is useful when many queries refer to a specific column. There are two types of partition in Hive: static (default) and dynamic partition. Dynamic partition in Hive can be used by setting the following properties:

```
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = non-strict;.
```

Here is the syntax for creation partitions in Hive table:

```
CREATE TABLE table_name
(column1 type, column2 type,...)
PARTITIONED BY
(partition1 type, partition2 type,...);.
```

Some commonly used dimensions as partitions keys are:

- Partitions by date and time – date and time, such as year, month, day can be used as partition keys when data is associated with the time dimension.
- Partitions by locations – country, territory, state, city can be used as partition keys when data is location-related.
- Partitions by business logic – department, applications, customers, etc. can be used as partition keys when data can be separated by some business logic. [39], [40]

- Bucketing:

Bucketing data is similar to partitioning data. That is another way of the decomposing table into manageable parts and the user can set the size of those parts. Hive table is split into multiple Hive partitions, and Hive partitions are further divided into clusters or buckets that are easier

to manage and maintain, called clustering or bucketing. If the bucket key and join keys are common join performance can be improved with bucketing. To use bucketing it is necessary to set the following property before writing data to the bucketed table:

```
set hive.enforce.bucketing = true;
```

and for bucketing in the join operation:

```
set hive.optimize.bucketmapjoin = true;
```

that enables Hive to do bucket level join during the map stage join. Here is the syntax for dividing Hive table into buckets:

```
CREATE TABLE table_name
(column1 type, column2 type,...)
PARTITIONED BY
(partition1 type, partition2 type,...)
CLUSTERED BY (column1, column2,...) INTO
num BUCKETS;. [39], [40]
```

- Cost-based optimization:

Hive optimizes each query's execution plan before submitting for final execution, but optimization techniques are not based on the cost of the query. Cost-based optimization (CBO) is a new feature on Hive that offers better Hive query performance regarding cost, resulting in different decisions: which types of joins to perform, how to order joins, the degree of parallelism, etc. To use CBO, the following properties should be set at the beginning of the query:

```
set hive.cbo.enable=true;
set hive.compute.query.using.stats = true;
set hive.stats.fetch.column.stats = true;
set hive.stats.fetch.partition.stats = true;. [41], [42]
```

- Statistics:

Statistics allow Hive to optimize queries using a cost-based optimizer available in a newer version of Hive. This usually brings significant performance improvements. Hive can generate statistics for all types of columns. If there are a lot of columns, statistics can be generated only for selected columns. Statistics should be recomputed whenever the table content is changed. If the only partition is changed statistics can be recomputed for selected partitions. Statistics on tables and columns can be generated using the following commands:

```
ANALYZE TABLE table_name COMPUTE STATISTICS;
```

```
ANALYZE TABLE table_name COMPUTE STATISTICS FOR COLUMNS;.
```

Caching of statistics is allowed in newer Hive versions with HBase as a metadata store:

```
ANALYZE TABLE table_name CACHE METADATA;. [43]-[45]
```

- Bloom filter:

Bloom filter is a data structure that is designed to quickly check if an element is contained in a set, which greatly improves performance because irrelevant blocks do not have to be read. It is suitable for queries containing = operation in where clause and can be applied to numeric as

well as non-numeric data. When creating or altering table, a bloom filter can be specified by setting the property `orc.bloom.filter.columns` to the columns for which the bloom filter is wanted. It is only available for ORC format. Benefits of the bloom filter can be increased by inserting data only sorted on the columns for which bloom filter is defined to avoid that all blocks of a table contain all distinct values of the column. It is recommended to set the following property:

```
set hive.enforce.sorting = true;
```

and describe in the create table statement which column should be sorted. [45], [46]

- **Denormalization:**

Normalization is a standard process used to model tables with some rules to deal with data redundancy. Denormalization is one way to avoid highly normalized table structures, as they require queries with joins that are difficult and expensive operations to perform and one of the most common reasons for performance issues. Join requires a full map reduce job to execute in Hadoop and with denormalization the extra map reduce job could be avoided. [47]

- **Join optimization:**

If tables contain a large amount of data, other types of joins can be used to improve the performance of Hive queries, instead of normal SQL joins. Following joins can be used for Hive query optimization [48]-[50]:

- *Common join* is also called reduce side join and in Hive it is a basic join. For common joins, it is important to make sure that a big table is on the rightmost side.
- *Map join* is efficient when one of the join tables is small enough so that it can fit into the memory and join is performed in the map phase of MapReduce job. Since there is no reducer involved in map join, it is much faster when compared to regular join. Hive can convert common join into map join based on the input file size with the following setting:

```
set hive.auto.convert.join = true;.
```
- *Bucket map join* is a special type of map join applied on the bucket tables. The join tables must be bucket tables, join on buckets columns, and the bucket number in bigger tables must be a multiple of the bucket number in the small tables. With the following parameters bucket map join in Hive can be enabled:

```
set hive.optimize.bucketmapjoin = true;
```



```
set hive.optimize.bucketmapjoin.sortedmerge = true;.
```
- *Sort merge bucket (SMB) join* is the join on the bucket tables that have the same sorted, bucket, and join condition columns. SMB join reads data from both bucket tables and performs common join.
- *Sort merge bucket map (SMBM) join* is a special bucket join that triggers only map join and can avoid caching all rows in the memory as map join does.
- *Skew join* is helpful when a table is skewed (a table that is having values that are present in large numbers in the table compared to the other data). Skew join can be enabled with the following parameter:

```
set hive.optimize.skewjoin = true;.
```

- **Indexing:**

Using indexes (storage index, aggregate index, bitmap index, compact index) improves query execution performance by not scanning all rows in a table that has an index, but first checking the index and then going to a specific column and performing operation in less time. The use of indexing will create an index table as a reference for the original table. Better data access for operations such as where, group by, and join can be achieved with indexes. Following indexes can be used for Hive query optimization [51]-[54]:

- *Storage index* – Storage index is only available if the table is in ORC format in Hive. It enables skipping blocks where a value is not contained without needing to read the block. It is most suitable for numeric values and if a query has where clause with <, >, = operators or joins. For effective storage index it is needed to have inserted data into the table sorted on the columns for which the storage index is wanted. It is recommended to do this only for 1 or 2 columns, set in the create table statement which columns should be sorted and:

```
set hive.enforce.sorting = true;.
```
- *Aggregate index* – The aggregate index has no counterpart in the traditional database world. It is a table with predefined aggregations (average, count, sum, etc.) of a certain column.
- *Bitmap index* – Bitmap index in Hive can be compared to the bitmap index in traditional databases. It is suitable for queries using where clause with = operator and if the column has only a few distinct values. Bitmap index in Hive is a table that contains the unique values of the column with the block where they are stored.
- *Compact index* – Compact index in Hive can be compared to a normal index in traditional databases. It is suitable for columns containing a lot of distinct values and for queries using where clause with <, >, = operators or joins. It is recommended for numeric values. The compact index in Hive is a sorted table containing all the values of the column to be indexed and the blocks where they occur. The compact index can be defined on several columns.

3. CONCLUSION

Query performance and optimization when processing big data has become a very important task and challenge. Hive without optimization is useful only for the queries that require a scan of the whole table. Several types of Hive query optimization techniques, that are available when executing Hive queries to improve Hive performance, are described in this paper.

The contribution of this paper is the state of the art in the field of big data optimization techniques. Using these techniques it is possible to process the query efficiently in Hadoop using Hive. An interesting topic for future research is to explore new possibilities and ways of improving big data query performance.

REFERENCES

- [1] A. K. Bhadani, D. Jothimani: Big Data: Challenges, Opportunities, and Realities, chapter in an edited volume Effective Big Data Management and Opportunities for Implementation, 2016
- [2] M. Chen, S. Mao, Y. Zhang, V. C. M. Leung: Big Data: Related Technologies, Challenges, and Future Prospects, Springer Briefs in Computer Science, 2014
- [3] M. Madison, M. Barnhill, C. Napier, J. Godin: NoSQL Database Technologies, Journal of International Technology and Information Management, vol. 24, issue 1, article 1, 2015
- [4] G. Vaish: Getting Started with NoSQL, Packt Publishing, 2013
- [5] D. McCreary, A. Kelly: Making Sense of NoSQL, Manning Publications Co., 2014
- [6] S. Tiwari: Professional NoSQL, John Wiley & Sons, Inc., 2011
- [7] J. R. Lourenco, V. Abramova, M. Vieira, B. Cabral, J. Bernardino: NoSQL Databases: A Software Engineering Perspective, Springer International Publishing Switzerland, 2015
- [8] A. Pothuganti: Big Data Analytics: Hadoop-Map Reduce & NoSQL Databases, International Journal of Computer Science and Information Technologies (IJCSIT), vol. 6 (1), pp. 522-527, 2015
- [9] A. Y. Zomaya, S. Sakr: Handbook of Big Data Technologies, Springer, 2017
- [10] T. White: Hadoop: The Definitive Guide, O'Reilly, 2015
- [11] R. Jhaji: Apache Hadoop Cookbook, Exelixis Media P. C., 2016
- [12] C. Lam: Hadoop in Action, Manning Publications Co., 2010
- [13] A. Holmes: Hadoop in Practice, Manning Publications Co., 2012
- [14] Y. Chen, X. Qin, H. Bian, J. Chen, Z. Dong, X. Du, Y. Gao, D. Liu, J. Lu, H. Zhang: A Study of SQL-on-Hadoop Systems, Springer International Publishing Switzerland, 2014
- [15] X. Qin, Y. Chen, J. Chen, S. Li, J. Liu, H. Zhang: The Performance of SQL-on-Hadoop Systems: An Experimental Study, IEEE 6th International Congress on Big Data, 2017
- [16] D. Abadi, S. Babu, F. Ozcan, I. Pandis: Tutorial: SQL-on-Hadoop Systems, VLDB Endowment, vol. 8, no. 12, 2015
- [17] A. Tapdiya, D. Fabbri: A Comparative Analysis of state-of-the-art SQL-on-Hadoop Systems for Interactive Analytics, IEEE Big Data Conference, 2017
- [18] M. A. Kukreja: Apache Hive: Enterprise SQL on Big Data Frameworks, Technical Report, 2016
- [19] D. Du: Apache Hive Essentials, Packt Publishing, 2015
- [20] H. Bansal, S. Chauhan, S. Mehrotra: Apache Hive Cookbook, Packt Publishing, 2016
- [21] K. DeviPriya, V. C. Kumar: Efficient Measures for Improvement and Optimization of Big Data Hive Queries, Visleshana, vol. 2, no. 1, 2017
- [22] K. K. Mohan: Query Optimization in Big Data Hadoop using Hive, Innovation: International Journal of Applied Research, vol. 4, no. 1, 2016
- [23] B. Jain, M. Kakhani: Query Optimization in Hive for Large Datasets, Advances in Computer Science and Information Technology (ACSIT), vol. 2, no. 4, pp. 321-325, 2015
- [24] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, X. Zhang: Major Technical Advancements in Apache Hive, SIGMOD, 2014
- [25] <https://data-flair.training/blogs/hive-optimization-techniques/> Accessed February 2020
- [26] <https://www.qubole.com/blog/hive-best-practices/> Accessed February 2020
- [27] <https://www.qubole.com/blog/5-tips-for-efficient-hive-queries/> Accessed February 2020
- [28] <https://www.hdfstutorial.com/blog/hive-performance-tuning/> Accessed February 2020
- [29] <https://acadgild.com/blog/hive-optimization-techniques-with-examples> Accessed February 2020
- [30] <https://dzone.com/articles/how-to-improve-hive-query-performance-with-hadoop> Accessed February 2020
- [31] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, C. Curino: Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications, SIGMOD, 2015
- [32] K. Rattanaopas: A Performance Comparison of Apache Tez and MapReduce With Data Compression on Hadoop Cluster, 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017
- [33] P. Kannan: Beyond Hadoop MapReduce Apache Tez and Apache Spark, 2014
- [34] <https://www.cloudera.com/products/open-source/apache-hadoop/apache-tez.html> Accessed February 2020
- [35] <https://tez.apache.org/> Accessed February 2020
- [36] <https://cwiki.apache.org/confluence/display/Hive/Vectorized+Query+Execution> Accessed March 2020
- [37] S. Pal: SQL on Big Data – Technology, Architecture, and Innovation, Apress, 2016
- [38] <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC#LanguageManualORC-ORCFileFormat> Accessed March 2020
- [39] E. Costa, C. Costa, M. Y. Santos: Partitioning and Bucketing in Hive-Based Big Data Warehouses, Trends, and Advances in Information Systems and Technologies, pp. 764-774, 2018
- [40] E. Costa, C. Costa, M. Y. Santos: Evaluating Partitioning and Bucketing Strategies for Hive-based Big Data Warehousing Systems, Journal of Big Data, no. 34, 2019
- [41] S. Bagui, K. Devulapalli: Comparison of Hive's Query Optimisation Techniques, Int. J. Big Data Intelligence, vol. X, no. Y, 200x, 2018
- [42] T. Dokeroglu, M. S. Cinar, S. A. Sert, A. Cosar, A. Yazici: Improving Hadoop Hive Query Response Time Through Efficient Virtual Resource Allocation, Flexible Query Answering Systems, pp. 215-225, 2015

- [43] A. Gruenheid, E. Omiecinski, L. Mark: Query Optimization Using Column Statistics in Hive, IDEAS11, 2011
- [44] <https://cwiki.apache.org/confluence/display/Hive/StatsDev> Accessed March 2020
- [45] <https://snippetessay.wordpress.com/2015/07/25/hive-optimizations-with-indexes-bloom-filters-and-statistics/> Accessed March 2020
- [46] R. Jain, M. Rawat, S. Jain: Data Optimization Techniques using Bloom Filter in Big Data, International Journal of Computer Applications, vol. 142, no. 3, 2016
- [47] J. Rutherglen, D. Wampler, E. Capriolo: Programming Hive: Data Warehouse and Query Language for Hadoop, O'Reilly, 2012
- [48] <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+JoinOptimization> Accessed March 2020
- [49] <https://cwiki.apache.org/confluence/display/Hive/MapJoinOptimization> Accessed March 2020
- [50] <https://cwiki.apache.org/confluence/display/Hive/Skewed+Join+Optimization> Accessed March 2020
- [51] <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Indexing> Accessed March 2020
- [52] O. Mrudula, K. KvlIn, A. M. Sowjanya: Query Optimization Using Index Joins for Performance Gain in Hive, IJAICT, vol. 3, issue 08, 2016
- [53] A. Shrivas, D. Paranjpe: Optimizing the Performance of Hadoop Using Index-Based Join Operation, International Journal of Recent Development in Engineering and Technology (IJRDET), vol. 7, issue 4, 2018
- [54] G. Sravanthi, C. V. S. Satyamurty: Performance Evaluation and Optimization of Hybrid Indexes in Hive, International Journal of Research, vol. 7, issue IX, 2018

BIOGRAPHY

Vedrana Nerić graduated in 2006 from the Faculty of Electrical Engineering of the University of Tuzla, department of Technical Informatics. During the studies, as a recognition of the achieved success, she was awarded with three Silver and a Gold Medal by the University of Tuzla. Academic professional title Master of Electrical Engineering she acquired at the same faculty in 2013 and she is currently a Ph.D. student. She is employed in the company Virgin Pulse, Tuzla as a senior data engineer. In 2014, she was selected to the collaborative teaching assistant for the scientific field of Computer Science at the Faculty of Electrical Engineering of the University of Tuzla.

Nermin Sarajlić graduated from the Faculty of Electrical Engineering of the University of Tuzla in 1987. He acquired the Master of Sciences degree at the Faculty of Electrical Engineering and Mechanical Engineering of the University of Tuzla in 1997, and acquired Ph.D. degree at the Faculty of Electrical Engineering of the University of Tuzla in 2002. His field of interest is the calculation of the coupled electromagnetic-temperature fields; cryptography, cryptanalysis.