ELEKTROTEHNIŠKI VESTNIK 91(5): 272-283, 2024
ORIGINAL SCIENTIFIC PAPER

# Optimization of Management and Processing of Big Data on a Platform for Distributed Data Storage

**Vedrana Nerić[1], Nermin Sarajlić[2], Đulaga Hadžić[3]**

*[1] Electrical Engineering and Computer Science, Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina*
*[2] Electrical Engineering and Computer Science, Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina*
*[3] Department of Software Engineering, Polytechnic Faculty, University of Zenica, Bosnia and Herzegovina*
*E-mail: vedrana.neric@untz.ba*

**Abstract.** Traditional systems for managing relational databases (Relational Database Management Systems, or RDBMS) use structured data that is organized into tables, rows, and columns with defined relations between the tables. These systems are suitable for working with moderate to large amounts of structured data but may have problems dealing with extremely large amounts of data as well as unstructured data. In order to overcome the problems in the operation of these systems, the big data concept appeared, which implies the application of various technologies that enable the management of huge amounts of diverse data that are collected at a high speed. The paper investigates optimization techniques for managing and processing large amounts of data on a platform for a distributed data storage. An experimental optimization setup is performed to improve the performance when executing queries on large amounts of data. Experimental results demonstrated on a large amount of data on a specific platform show that it is possible to efficiently improve the optimization.

**Keywords:** big data, Hadoop, Hive, Cloudera, optimization

**Optimizacija upravljanja in obdelave velikih podatkov na platformi za porazdeljeno shranjevanje podatkov**

Tradicionalni sistemi za upravljanje relacijskih baz podatkov (Relational Database Management Systems ali RDBMS) uporabljajo strukturirane podatke, ki so organizirani v tabele, vrstice in stolpce z definiranimi odnosi med tabelami. Ti sistemi so primerni za delo z zmernimi do velikimi količinami strukturiranih podatkov, vendar imajo lahko težave pri ravnanju z izjemno velikimi količinami podatkov kot tudi z nestrukturiranimi podatki. Za premostitev težav pri delovanju teh sistemov se je pojavil koncept velikih podatkov, ki pomeni uporabo različnih tehnologij, ki omogočajo upravljanje ogromnih količin raznovrstnih podatkov, ki se zbirajo z veliko hitrostjo. Prispevek raziskuje optimizacijske tehnike za upravljanje in obdelavo velikih količin podatkov na platformi za porazdeljeno shranjevanje podatkov. Izvedena je bila eksperimentalna postavitev z namenom optimizacije, ki zagotavlja boljše delovanje pri izvajanju poizvedb pri velikih količinah podatkov. Eksperimentalni rezultati kažejo, da je možno učinkovito izboljšati optimizacijo, kar je prikazano na veliki količini podatkov na določeni platformi.

## 1 INTRODUCTION

With each increase in the amount of data and with each improvement of information systems in data collection from distributed and diverse sources, the amount of problems related to the quality of data also increases. The information and data quality are the topics of extensive and very active research both from the perspective of information systems and databases. [1]

Big data refers to large and complex data sets that traditional systems are unable to process, manage, and analyze in reasonable time frames. The big data concept is characterized by three primary attributes known as 3V [2], [3]:

- Volume: a large amount of data that can come from different sources,
- Velocity: the high speed at which data is collected from different devices, social networks, websites, online transactions,
- Variety: different types of data that can be structured (databases, spreadsheets), semi-structured (XML, JSON files), and unstructured (text documents, images, audio, video).

For the big data description, two more attributes are added to the three above mentioned [3], [4]:

- Veracity: truthfulness, which refers to the quality and reliability of data, which may include incomplete and inconsistent data, which may present challenges in analysis and decision-making,
- Value: the value of data is in its potential to provide organizations with a competitive advantage, improve decision-making, and drive innovation.

With big data technologies, the data also must be managed and processed intelligently to efficiently

*Received 1 April 2024*
*Accepted 9 October 2024*

execute queries, and optimization needs to be done in order to process the data quickly. There are many optimization techniques that can be used, but the specific configurations and optimizations that are chosen need to be aligned with the use cases and data processing requirements to get the best performances.

The paper is structured into six chapters: Platform (Chapter 2), Hive optimization techniques (Chapter 3), Experimental setup (Chapter 4), Results (Chapter 5), and Conclusion (Chapter 6).

# 2 PLATFORM

## 2.1 Hadoop

The technology most often associated with big data is Hadoop [5]. Hadoop is an open-source framework of the Apache Foundation that is used to store and process large amounts of data [6]. Hadoop consists of four basic components: Hadoop Common (a set of libraries and configuration files that are required for the operation of Hadoop itself), HDFS (Hadoop distributed file system that is responsible for storing data in the cluster), MapReduce (a data processing model), and YARN (Hadoop operating system that is in charge of resource allocation and job management) [7].

## 2.2 Cloudera

Cloudera Distribution for Hadoop (CDH) [8] is an open-source distribution of the Cloudera platform that includes Apache Hadoop. The platform is designed to help organizations manage and analyze large amounts of data, including structured and unstructured data, in a distributed and scalable manner. CDH is a comprehensive solution for storing, processing, and analyzing large amounts of data, which extends the capabilities of the core Hadoop ecosystem by integrating additional tools and components, providing management and monitoring tools, improving security, and providing various services to help organizations harness the power of big data in their operations.

CDH includes an Apache Hive [9] data warehouse designed to simplify querying and analysis of large data sets stored in distributed storage systems, such as the Hadoop Distributed File System (HDFS). Hive enables users to work with large amounts of data using a familiar SQL-like interface, making it accessible to a large number of data professionals.

# 3 HIVE OPTIMIZATION TECHNIQUES

In order to improve the performance of Hive queries and data processing, Hive optimization techniques are used. Optimizing Hive queries is essential when dealing with large data sets to reduce the query execution time and resource usage. The paper discusses the most commonly used Hive optimization techniques [10] - [12]:

- Predicate pushdown [13] is a Hive optimization technique used to improve the query performance by having data filtering happen at the data source level (e.g. HDFS) before the data is sent to Hive for further processing. In this way, the amount of data that needs to be read and processed is reduced, unnecessary data is avoided, and data I/O is reduced, which results in faster query execution. This technique is enabled with the parameter `hive.optimize.ppd`, which is set to `true` by default.
- Vectorization significantly reduces the CPU load and improves the query performance by processing data in batches (vectors) rather than row by row. Vectorization processing allows operators and functions in Hive queries to operate on entire vectors of data [14]. For example, instead of processing one row at a time, a filter or aggregation can be applied to a set of rows together. Vectorization can be enabled by setting the `hive.vectorized.execution.enabled` parameters to `true`.
- Hive allows a table to be organized into multiple partitions where the same types of data can be grouped together. Partitioning [15] can improve the performance and help organize data. When querying, a specific partition of the table containing the query value is accessed, thus reducing the I/O time required to execute the query and increasing execution speed. With static partitioning, the values of the partitioned columns need to be manually passed when loading data into the table. Dynamic partitioning automatically creates partitions based on the values in a specified column, allowing for more flexible and scalable data storage. When creating a Hive table, it is specified `PARTITIONED BY` part of the command.
- Bucketing [15] is a Hive data organization technique similar to partitioning with the added functionality of dividing large data sets into smaller chunks. A Hive table is divided into multiple Hive partitions, and the Hive partition is further divided into clusters or buckets that are easier to manage and maintain, which is called clustering or bucketing. When creating a Hive table, it is specified `CLUSTERED BY` part of the command.
- Parallelism focuses on the parallel execution of tasks within Hive query and data processing jobs [16]. Hive automatically parallelizes tasks for certain types of joins (e.g. map join) and aggregation operations, so it is necessary to ensure that queries are structured to take advantage of the optimization. Parallelism can be enabled by setting the `hive.exec.parallel` parameters to `true`.
- Cost-based optimization (CBO) [11] is a technique used for Hive optimization by evaluating the cost of different query execution plans and choosing the most efficient one. This optimization approach takes

statistics and data information into account and helps Hive make more informed decisions about how to execute queries. CBO can be enabled by setting the parameter `hive.cbo.enable` to `true`.

- Statistics allow the query optimizer to make decisions about query execution plans and reduce the query execution time [17]. The `ANALYZE TABLE` command is used to collect Hive table statistics, gathering information about data distribution, number of rows, and column statistics. For query optimization, it is necessary to periodically update the statistics of tables and columns to take into account changes in the distribution and amount of data.
- Skew join [10] occurs when one or more keys in a join operation have a disproportionately large amount of associated data, causing a few tasks to process most of the data while other tasks are idle, resulting in an uneven resource usage and slower the query performance. Skew join can be enabled by setting the `hive.optimize.skewjoin` parameters to `true`. During execution, skew keys are detected, and instead of being processed, they are temporarily stored in the HDFS directory. In the next map-reduce job, those skew keys are processed, so the next map-reduce job will be much faster since it will be a map join.
- In a traditional join operation, data from two or more tables is mixed and sorted by a common key before performing the join, which can be resource-intensive and time-consuming, especially when dealing with large data sets. With a map join [10], one of the tables (smaller) is completely loaded into memory as a hash table, and the other table (larger) is transferred row by row or block by block through mapper tasks. As each row from the larger table is processed, Hive uses the shared join key to find a match in the hash table that was created from the smaller table. In this way, the need to mix and sort data, as with a traditional join, is eliminated. Map join can be enabled by setting the `hive.auto.convert.join` parameters to `true`.
- Bucket map join [10] combines two optimization techniques map join and bucketing, which can be enabled by setting the `hive.optimize.bucketmapjoin` parameters to `true`. Sort merge bucket (SMB) join [10] is an extension of sort merge join, which combines the benefits of bucketing and sorting for even more efficient merge operations. SMB join can be enabled by setting the following parameters `hive.auto.convert.sortmerge.join` and `hive.optimize.bucketmapjoin.sortedmerge` to `true`. Sort merge bucket map (SMBM) join [10] is like an SMB join that only runs a map-side join and can avoid caching all rows in memory like a map join. SMBM can be enabled by setting the parameter `hive.auto.convert.sortmerge.join.to.mapjoin` to `true`.
- Compression [10] reduces the space required for the data storage. When data is compressed, the storage space is reduced, but I/O during query execution is also reduced, resulting in a faster query performance. Hive supports various compression codecs, such as Snappy, Gzip, LZO, and others. The Snappy codec is known for its balance between the compression ratio and speed, making it a popular choice for Hive tables. The compression codec can be specified using the `STORED AS` part of the command when creating or altering Hive tables.

## 4 Experimental Setup

### 4.1 Data Sets

The worldwide recognized Transaction Processing Performance Council Benchmark H (TPC-H) [18] is used for the analysis. It enables performance testing for data warehouse solutions. TPC-H consists of business-oriented queries and databases selected to have a broad industry relevance. TPC-H is a decision support system that enables the analysis of large amounts of data, the execution of queries with a high degree of complexity, and the provision of answers to critical business questions.

In TPC-H, a scale factor (SF) is used to describe the amount of data. For the analysis purposes, three databases are created: TPCH_0_5, TPCH_1 and TPCH_2. They contain different amounts of data with scaling factors of 0.5, 1, and 2, respectively. Each database contains the same tables with different numbers of records according to the scaling factor shown in Table 1. Different scaling factors are used for the analysis in order to perform performance testing and Hive optimization techniques on different amounts of data.

Table 1. Amount of data for different scaling factors.

| Table | SF = 0.5 | SF = 1 | SF = 2 |
|---|---:|---:|---:|
| region | 5 | 5 | 5 |
| nation | 25 | 25 | 25 |
| supplier | 5000 | 10000 | 20000 |
| customer | 75000 | 150000 | 300000 |
| part | 100000 | 200000 | 400000 |
| partsupp | 400000 | 800000 | 1600000 |
| orders | 750000 | 1500000 | 3000000 |
| lineitem | 2999671 | 6001215 | 11997996 |

### 4.2 Hive Configuration Parameters

For the analysis, in addition to the default combination of Hive configuration parameters (P1) that have set values after Cloudera platform installation, various Hive optimization techniques are used by adjusting additional parameters through combinations (P2–P10) in order to perform a comparison and determine their impact on execution time prompts versus the default settings. The Hive configuration parameters begin with "hive." and define properties of the Hive system. There is a large number of Hive configuration parameters. After a long analysis and measurement of the impact of various configuration parameters, individually and in combination with others, in addition to the default P1 (Table 2), those cases that stand out according to their results are presented in Table 3 (P2-P9) and Table 4 (P10).

Table 2. P1 default values of the analyzed Hive configuration parameters.

| Optimization technique | Parameter | Value |
|---|---|---|
| P1 | | |
| Execution Engine | hive.execution.engine | mr |
| Predicate Pushdown | hive.optimize.ppd | true |
| | hive.optimize.ppd.storage | true |
| | hive.ppd.remove.duplicatefilters | true |
| | hive.ppd.recognizetransivity | true |
| Vectorization | hive.vectorized.execution.enabled | true |
| | hive.vectorized.execution.reduce. enabled | true |
| | hive.vectorized.execution.reduce. groupby.enabled | true |
| Dynamic Partitioning | hive.exec.dynamic.partition | true |
| | hive.exec.dynamic.partition.mode | strict |
| | hive.exec.max.dynamic.partitions | 1000 |
| | hive.exec.max.dynamic.partitions .pernode | 100 |
| Indexing | hive.optimize.index.filter | true |
| | hive.index.compact.binary.search | true |
| Paralelism | hive.exec.parallel | false |
| | hive.exec.parallel.thread.number | 8 |
| Cost-based Optimization | hive.cbo.enable | false |
| Statistics | hive.stats.autogather | true |
| | hive.compute.query.using.stats | false |
| | hive.stats.fetch.column.stats | true |
| | hive.stats.fetch.partition.stats | true |
| Skew Join | hive.optimize.skewjoin | false |
| | hive.skewjoin.key | 100000 |
| | hive.skewjoin.mapjoin.map.tasks | 10000 |
| | hive.skewjoin.mapjoin.min.split | 33554432 |
| Map Join | hive.auto.convert.join | true |
| | hive.auto.convert.join. noconditionaltask | true |
| | hive.auto.convert.join. noconditionaltask.size | 20971520 |
| | hive.mapjoin.smalltable.filesize | 25000000 |
| Bucket Map Join | hive.optimize.bucketmapjoin | false |
| Sort merge bucket join | hive.auto.convert.sortmerge.join | false |
| | hive.optimize.bucketmapjoin. sortedmerge | false |
| Sort merge bucket map join | hive.auto.convert.sortmerge.join. to.mapjoin | false |
| Compression | mapred.compress.map.output | true |
| | mapred.output.compress | false |
| | hive.exec.compress.output | false |
| | hive.exec.compress.intermediate | false |

Table 3. P2 – P9 combinations of Hive configuration parameters.

| Optimization technique | Parameter | Value |
|---|---|---|
| P2 | | |
| Paralelism | hive.exec.parallel | true |
| | hive.exec.parallel.thread.number | 16 |
| P3 | | |
| Cost-based Optimization | hive.cbo.enable | true |
| P4 | | |
| Cost-based Optimization | hive.cbo.enable | true |
| Statistics | hive.stats.autogather | true |
| | hive.compute.query.using.stats | true |
| | hive.stats.fetch.column.stats | true |
| | hive.stats.fetch.partition.stats | true |
| P5 | | |
| Paralelism | hive.exec.parallel | true |
| | hive.exec.parallel.thread.number | 16 |
| Cost-based Optimization | hive.cbo.enable | true |
| Statistics | hive.stats.autogather | true |
| | hive.compute.query.using.stats | true |
| | hive.stats.fetch.column.stats | true |
| | hive.stats.fetch.partition.stats | true |
| P6 | | |
| Cost-based Optimization | hive.cbo.enable | true |
| Statistics | hive.stats.autogather | true |
| | hive.compute.query.using.stats | true |
| | hive.stats.fetch.column.stats | true |
| | hive.stats.fetch.partition.stats | true |
| | ANALYZE TABLE tablename COMPUTE STATISTICS | |
| P7 | | |
| Skew Join | hive.optimize.skewjoin | true |
| | hive.skewjoin.key | 100000 |
| | hive.skewjoin.mapjoin.map.tasks | 10000 |
| | hive.skewjoin.mapjoin.min.split | 33554432 |
| P8 | | |
| Map Join | hive.auto.convert.join | true |
| | hive.auto.convert.join. noconditionaltask | true |
| | hive.auto.convert.join. noconditionaltask.size | 20971520 |
| | hive.mapjoin.smalltable.filesize | 25000000 |
| Bucket Map Join | hive.optimize.bucketmapjoin | true |
| Sort merge bucket join | hive.auto.convert.sortmerge.join | true |
| | hive.optimize.bucketmapjoin. sortedmerge | true |
| Sort merge bucket map join | hive.auto.convert.sortmerge.join. to.mapjoin | true |
| P9 | | |
| Compression | mapred.compress.map.output | true |
| | mapred.output.compress | true |
| | hive.exec.compress.output | true |
| | hive.exec.compress.intermediate | true |

Table 4. P10 combination of Hive configuration parameters.

| Optimization technique | Parameter | Value |
|---|---|---|
| P10 | | |
| Multiple | | |
| Paralelism | hive.exec.parallel | true |
| | hive.exec.parallel.thread.number | 16 |
| Cost-based Optimization | hive.cbo.enable | true |
| Statistics | hive.stats.autogather | true |
| | hive.compute.query.using.stats | true |
| | hive.stats.fetch.column.stats | true |
| | hive.stats.fetch.partition.stats | true |
| Skew Join | hive.optimize.skewjoin | true |
| | hive.skewjoin.key | 100000 |
| | hive.skewjoin.mapjoin.map.tasks | 10000 |
| | hive.skewjoin.mapjoin.min.split | 33554432 |
| Map Join | hive.auto.convert.join | true |
| | hive.auto.convert.join. noconditionaltask | true |
| | hive.auto.convert.join. noconditionaltask.size | 20971520 |
| | hive.mapjoin.smalltable.filesize | 25000000 |
| Bucket Map Join | hive.optimize.bucketmapjoin | true |
| Sort merge bucket join | hive.auto.convert.sortmerge.join | true |
| | hive.optimize.bucketmapjoin. sortedmerge | true |
| Sort merge bucket map join | hive.auto.convert.sortmerge.join. to.mapjoin | true |
| Compression | mapred.compress.map.output | true |
| | mapred.output.compress | true |
| | hive.exec.compress.output | true |
| | hive.exec.compress.intermediate | true |

## 5 Results

### 5.1 Experimental results

For the analysis, ten different combinations of the parameters (P) are used, for which the time, expressed in seconds, during the execution of the queries (Q) is measured. The results are presented in tables for different databases that contain data according to scaling factors of 0.5, 1, and 2. Table 5 contains the query execution time for database TPCH_0_5, Table 6 for TPCH_1, and Table 7 for TPCH_2. When creating the databases, the `TEXTFILE` format is used for tables.

The same hardware and configuration are used for all tests. On a laptop with an Intel Core i5 processor and 32GB of RAM, a VMware Workstation virtual machine is used, on which the CDH 6.3.2 version of the Cloudera platform is installed with all the necessary components for the distributed storage of large amounts of data, their processing, and their analysis. For the analysis, the TPC-H queries are used. They are designed to analyze the functionality of the system and have a realistic context. A set of ten representative TPC-H [18] queries (Q1, Q4, Q6, Q11, Q12, Q13, Q15, Q16, Q20, and Q22) is selected so that combinations of simple and complex queries are included.

Before any changes to the platform, all the selected queries are run with the default Hive configuration parameters, and the execution time is recorded for all three databases, i.e., three different amounts of data. The execution time in seconds for the default configuration is shown in column P1 (Tables 5, 6, 7). Experimentation is then performed with various Hive configuration parameters and the application of some of the Hive optimization techniques, such as parallelism, cost-based optimization, statistics, join optimization, and compression. Hive allows setting hundreds of different parameters, and some of the most commonly used ones are used for the analysis purposes. Configuration changes are made systematically by changing one set of parameters related to a specific optimization technique or more, while other parameters are kept constant. The same set of queries is run with the parameters changed and the query execution time recorded in columns P2–P10 (Tables 5, 6, 7).

### 5.2 Analysis of the result

Based on the measured query execution times in seconds (Tables 5, 6, 7), which are run on databases TPCH_0_5, TPCH_1 and TPCH_2 for different combinations of Hive configuration parameters (Tables 3, 4), Tables 8, 9, 10 are created, in which the percentage difference in query execution time for combinations P2–P10 compared to the default setting of parameters P1 (Table 2) is calculated. The tables represent an important part of the analysis regarding the optimization of the Hive performance, and based on these results, the effect and impact of certain optimization techniques and the configuration parameters on the results can be seen.

Table 5. TPCH_0_5 (sec).

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Q1** | 184 | 152 | 150 | 143 | 137 | 148 | 134 | 158 | 158 | 160 |
| **Q4** | 194 | 191 | 206 | 212 | 198 | 214 | 192 | 191 | 205 | 243 |
| **Q6** | 65 | 64 | 64 | 62 | 58 | 65 | 59 | 57 | 65 | 63 |
| **Q11** | 268 | 259 | 271 | 285 | 253 | 261 | 259 | 259 | 236 | 252 |
| **Q12** | 213 | 238 | 208 | 206 | 202 | 213 | 208 | 209 | 191 | 230 |
| **Q13** | 186 | 192 | 197 | 189 | 196 | 202 | 188 | 188 | 184 | 205 |
| **Q15** | 373 | 406 | 345 | 369 | 364 | 353 | 355 | 372 | 370 | 423 |
| **Q16** | 289 | 260 | 280 | 282 | 250 | 281 | 257 | 267 | 252 | 399 |
| **Q20** | 397 | 368 | 379 | 361 | 330 | 384 | 376 | 372 | 373 | 348 |
| **Q22** | 248 | 241 | 250 | 249 | 239 | 251 | 231 | 232 | 249 | 263 |

Table 6. TPCH_1 (sec).

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Q1** | 231 | 225 | 207 | 217 | 206 | 211 | 229 | 219 | 225 | 243 |
| **Q4** | 311 | 300 | 327 | 300 | 274 | 319 | 264 | 267 | 270 | 292 |
| **Q6** | 78 | 74 | 75 | 74 | 74 | 75 | 76 | 74 | 81 | 76 |
| **Q11** | 301 | 272 | 280 | 278 | 270 | 280 | 260 | 273 | 262 | 275 |
| **Q12** | 273 | 300 | 298 | 297 | 265 | 295 | 276 | 274 | 278 | 306 |
| **Q13** | 231 | 236 | 221 | 233 | 219 | 224 | 208 | 198 | 221 | 231 |
| **Q15** | 504 | 587 | 480 | 516 | 514 | 479 | 494 | 505 | 520 | 570 |
| **Q16** | 289 | 271 | 283 | 289 | 265 | 301 | 301 | 300 | 267 | 293 |
| **Q20** | 498 | 421 | 470 | 482 | 444 | 460 | 449 | 476 | 456 | 495 |
| **Q22** | 271 | 250 | 263 | 269 | 271 | 263 | 242 | 262 | 274 | 273 |

Table 7. TPCH_2 (sec).

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Q1** | 368 | 346 | 338 | 354 | 326 | 336 | 340 | 347 | 354 | 364 |
| **Q4** | 447 | 417 | 470 | 483 | 439 | 462 | 478 | 420 | 435 | 474 |
| **Q6** | 119 | 105 | 105 | 111 | 106 | 102 | 119 | 111 | 112 | 111 |
| **Q11** | 312 | 271 | 317 | 312 | 284 | 306 | 300 | 309 | 306 | 315 |
| **Q12** | 448 | 485 | 450 | 456 | 446 | 449 | 468 | 437 | 450 | 483 |
| **Q13** | 249 | 242 | 258 | 241 | 266 | 247 | 251 | 262 | 247 | 274 |
| **Q15** | 807 | 884 | 749 | 771 | 792 | 745 | 743 | 753 | 745 | 917 |
| **Q16** | 354 | 326 | 335 | 333 | 337 | 339 | 338 | 316 | 326 | 613 |
| **Q20** | 672 | 689 | 698 | 732 | 742 | 679 | 642 | 652 | 664 | 835 |
| **Q22** | 315 | 283 | 313 | 305 | 306 | 304 | 294 | 297 | 300 | 335 |

Table 8. TPCH_0_5 (%).

| | P2% | P3% | P4% | P5% | P6% | P7% | P8% | P9% | P10% |
|---|---|---|---|---|---|---|---|---|---|
| **Q1** | -17,39 | -18,48 | -22,28 | -25,54 | -19,57 | -27,17 | -14,13 | -14,13 | -13,04 |
| **Q4** | -1,55 | 6,19 | 9,28 | 2,06 | 10,31 | -1,03 | -1,55 | 5,67 | 25,26 |
| **Q6** | -1,54 | -1,54 | -4,62 | -10,77 | 0,00 | -9,23 | -12,31 | 0,00 | -3,08 |
| **Q11** | -3,36 | 1,12 | 6,34 | -5,60 | -2,61 | -3,36 | -3,36 | -11,94 | -5,97 |
| **Q12** | 11,74 | -2,35 | -3,29 | -5,16 | 0,00 | -2,35 | -1,88 | -10,33 | 7,98 |
| **Q13** | 3,23 | 5,91 | 1,61 | 5,38 | 8,60 | 1,08 | 1,08 | -1,08 | 10,22 |
| **Q15** | 8,85 | -7,51 | -1,07 | -2,41 | -5,36 | -4,83 | -0,27 | -0,80 | 13,40 |
| **Q16** | -10,03 | -3,11 | -2,42 | -13,49 | -2,77 | -11,07 | -7,61 | -12,80 | 38,06 |
| **Q20** | -7,30 | -4,53 | -9,07 | -16,88 | -3,27 | -5,29 | -6,30 | -6,05 | -12,34 |
| **Q22** | -2,82 | 0,81 | 0,40 | -3,63 | 1,21 | -6,85 | -6,45 | 0,40 | 6,05 |

Table 9. TPCH_1 (%).

| | P2% | P3% | P4% | P5% | P6% | P7% | P8% | P9% | P10% |
|---|---|---|---|---|---|---|---|---|---|
| **Q1** | -2,60 | -10,39 | -6,06 | -10,82 | -8,66 | -0,87 | -5,19 | -2,60 | 5,19 |
| **Q4** | -3,54 | 5,14 | -3,54 | -11,90 | 2,57 | -15,11 | -14,15 | -13,18 | -6,11 |
| **Q6** | -5,13 | -3,85 | -5,13 | -5,13 | -3,85 | -2,56 | -5,13 | 3,85 | -2,56 |
| **Q11** | -9,63 | -6,98 | -7,64 | -10,30 | -6,98 | -13,62 | -9,30 | -12,96 | -8,64 |
| **Q12** | 9,89 | 9,16 | 8,79 | -2,93 | 8,06 | 1,10 | 0,37 | 1,83 | 12,09 |
| **Q13** | 2,16 | -4,33 | 0,87 | -5,19 | -3,03 | -9,96 | -14,29 | -4,33 | 0,00 |
| **Q15** | 16,47 | -4,76 | 2,38 | 1,98 | -4,96 | -1,98 | 0,20 | 3,17 | 13,10 |
| **Q16** | -6,23 | -2,08 | 0,00 | -8,30 | 4,15 | 4,15 | 3,81 | -7,61 | 1,38 |
| **Q20** | -15,46 | -5,62 | -3,21 | -10,84 | -7,63 | -9,84 | -4,42 | -8,43 | -0,60 |
| **Q22** | -7,75 | -2,95 | -0,74 | 0,00 | -2,95 | -10,70 | -3,32 | 1,11 | 0,74 |

Table 10. TPCH_2 (%).

| | P2% | P3% | P4% | P5% | P6% | P7% | P8% | P9% | P10% |
|---|---|---|---|---|---|---|---|---|---|
| **Q1** | -5,98 | -8,15 | -3,80 | -11,41 | -8,70 | -7,61 | -5,71 | -3,80 | -1,09 |
| **Q4** | -6,71 | 5,15 | 8,05 | -1,79 | 3,36 | 6,94 | -6,04 | -2,68 | 6,04 |
| **Q6** | -11,76 | -11,76 | -6,72 | -10,92 | -14,29 | 0,00 | -6,72 | -5,88 | -6,72 |
| **Q11** | -13,14 | 1,60 | 0,00 | -8,97 | -1,92 | -3,85 | -0,96 | -1,92 | 0,96 |
| **Q12** | 8,26 | 0,45 | 1,79 | -0,45 | 0,22 | 4,46 | -2,46 | 0,45 | 7,81 |
| **Q13** | -2,81 | 3,61 | -3,21 | 6,83 | -0,80 | 0,80 | 5,22 | -0,80 | 10,04 |
| **Q15** | 9,54 | -7,19 | -4,46 | -1,86 | -7,68 | -7,93 | -6,69 | -7,68 | 13,63 |
| **Q16** | -7,91 | -5,37 | -5,93 | -4,80 | -4,24 | -4,52 | -10,73 | -7,91 | 73,16 |
| **Q20** | 2,53 | 3,87 | 8,93 | 10,42 | 1,04 | -4,46 | -2,98 | -1,19 | 24,26 |
| **Q22** | -10,16 | -0,63 | -3,17 | -2,86 | -3,49 | -6,67 | -5,71 | -4,76 | 6,35 |

A negative percentage difference indicates a decrease in the execution time, which means that the optimized configuration is faster. A positive percentage difference suggests an increase in the query execution time with the optimized configuration, which is not desirable. A percentage difference close to 0% means there is a little or no change in the run time. A reduction in the Hive query execution time of 10% or more can be considered a significant improvement in the query performance.

The testing is performed with multiple queries on different amounts of data for different combinations of configuration parameters, so after displaying the percentage differences in the tables for three different databases for all combinations, the percentage differences are extracted for each combination of the parameters (P2-P10). For a better presentation and clarity of the results, their visualization is performed with a graphic representation in Figures 1 - 9. Based on the separated results in graphical displays, a more detailed analysis of the results and a better overview of the impact of each individual optimized configuration can be performed for each query and different databases.

Figures 1 - 9 provide a graphical representation of the percentage difference in the query execution time for the combinations (P2-P10) of configuration parameters compared to the default settings of P1 parameters for selected queries and different databases.

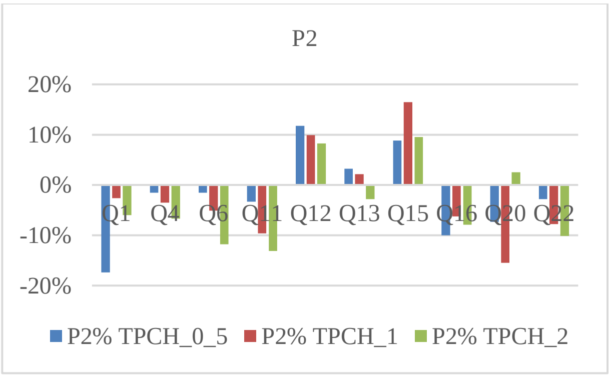

Figure 1. Parameters P2 (%).

The P2 combination (Figure 1) indicates the application of the optimization technique called parallelism. In Figure 1 with the P2 results, it can be noticed that parallelism is more efficient for larger data sets. For queries Q4, Q6, Q11, and Q22, the bigger negative percentage difference is presented for larger amounts of data in the TPCH_2 database in comparison to smaller amounts of data in the TPCH_0_5 and TPCH_1 databases. When dealing with a large data set that cannot fit in the memory, parallel processing can help by distributing the workload across multiple nodes. For smaller data sets that can fit in the memory, the overhead of matching parallel tasks can slow the execution, so in such cases, sequential processing is more efficient. The efficiency of parallelism also depends on the queries themselves. Some queries are sequential in nature and cannot be easily parallelized, and for such queries, this optimization technique will not provide significant improvements. For queries Q12 and Q15, the percentage difference is positive, so for these queries that contain aggregation, filtering, grouping, sorting, and joining with the largest table, parallelism gives worse results in comparison to default settings. If Hive queries are not optimized or have inefficient SQL logic, adding parallelism will not help either. But, for Q1, Q6, Q11, and Q20, significant improvements are obvious where the percentage difference is negative and higher than 10%. Queries involving multiple joins, subqueries, or complex transformations like Q11, Q16, Q20, and Q22 benefit from parallelism. Parallel execution allows these complex operations to be divided into smaller tasks that can be executed simultaneously.

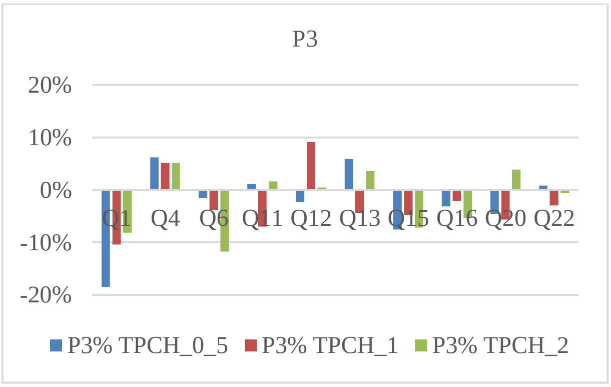

Figure 2. Parameters P3 (%).

The P3 combination (Figure 2) indicates the application of the optimization technique CBO (cost-based optimization). CBO can analyze alternative query execution plans and select a plan that minimizes resource usage and execution time. Based on the P3 results in Figure 2, CBO gives the best improvements for simple one-table queries without joins like Q1 and Q6, and it is also useful for complex queries with multiple joins, subqueries, and aggregations like Q15, Q16, and Q20. Queries with complex data filtering conditions can benefit from CBO ability to evaluate the filter selectivity and choose the most efficient order of operations.

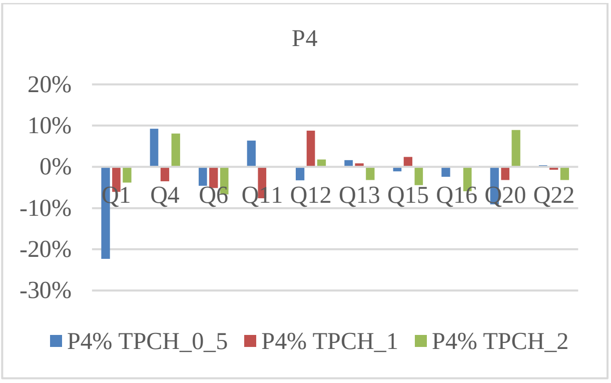

Figure 3. Parameters P4 (%).

The P4 combination (Figure 3) means the application of optimization techniques CBO and statistics. CBO relies on accurate statistics about tables and columns. When comprehensive statistics are available and up-to-date, CBO can make better decisions about query execution plans. Based on the P4 results in Figure 3, for some analyzed queries, the percentage difference is positive, and for some queries, it is negative, and the improvement also varies with different amounts of data. A significant improvement that is higher than 20% is obtained only for simple query Q1 with small amounts of data in the TPCH_0_5 database. So, the P4 combination can give small improvements that are less than 10% for some cases, like queries Q1, Q6, where select from only one table without joins is used.

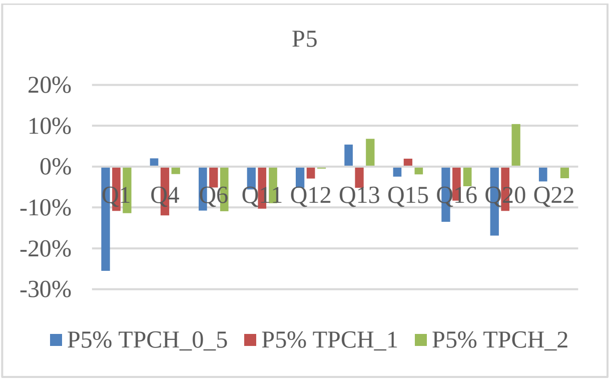

Figure 4. Parameters P5 (%).

The P5 combination (Figure 4) means the application of multiple optimization techniques, including parallelism, CBO, and statistics. In some cases, custom optimizations can be more efficient than relying only on one or two techniques, as shown by the P5 combination of several techniques, which gives better results compared to the combinations when parallelism (P2), CBO (P3), CBO and statistics (P4, P6) are used. This P5 combination enables improvements for almost all the analyzed queries and all three data sets, with the obvious negative percentage difference in Figure 4, which means a faster execution in comparison to default settings. So, the P5 combination can be recommended as efficient for different queries and amounts of data, with a decrease in the execution time and optimization improvement for most cases.

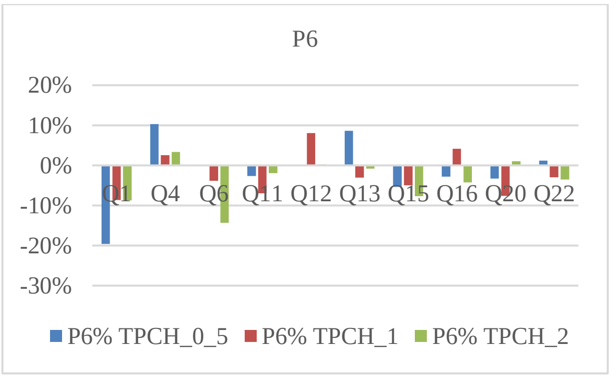

Figure 5. Parameters P6 (%).

The P6 combination (Figure 5) indicates the application of optimization techniques CBO and statistics with the `ANALYZE TABLE` commands to calculate statistics for each table. In order to achieve a better query performance using the CBO optimization technique, it is necessary to enable the collection of statistics for the Hive tables, which can be set at the database level or at the table level. In addition to the automatic collection of statistics, which at the database level is used for combinations of P4 and P5 with the `hive.stats.autogather` parameter, statistics can be manually collected for tables using the `ANALYZE TABLE` command. For the combination of configuration parameters P6, statistics are manually calculated for all tables from the databases TPCH_0_5, TPCH_1, and TPCH_2. The P6 combination gives similar results as the P4 combination. In Figure 5, significant improvements are visible only for simple one-table queries without joins Q1 and Q6, so this combination can be proposed for similar queries and smaller amounts of data.

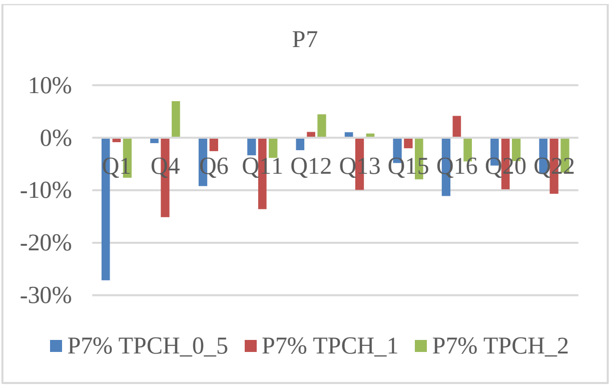

Figure 6. Parameters P7 (%).

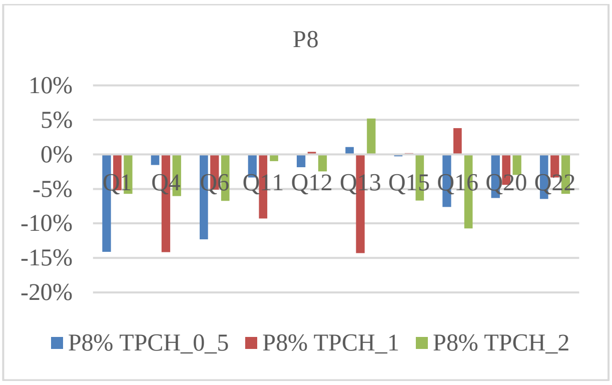

Figure 7. Parameters P8 (%).

The P7 combination (Figure 6) indicates the application of the skew join, and the P8 combination (Figure 7) indicates the application of the map join optimization technique. Skew join is designed to handle situations where there is skew data in the join keys. A map join is most efficient when one or more tables involved in the join operation are small enough to fit entirely in the memory. In such cases, loading a small table into the memory eliminates the need for expensive disk I/O operations, resulting in a faster query execution. The map join can also be useful when combined with other optimization techniques such as bucketing and sorting. Within the P8 combination, in addition to the configuration parameters for the map join, parameters for the bucket map join, sort merge bucket join, and sort merge bucket map join are also used. Unlike the previously mentioned combinations of optimization techniques that give improvements only in some cases, the P7 and P8 combinations give excellent performance improvements in almost all cases. Based on the P7 and P8 results in Figures 6 and 7, it is obvious that notable improvements with a negative percentage difference are obtained for most analyzed queries and amounts of data.

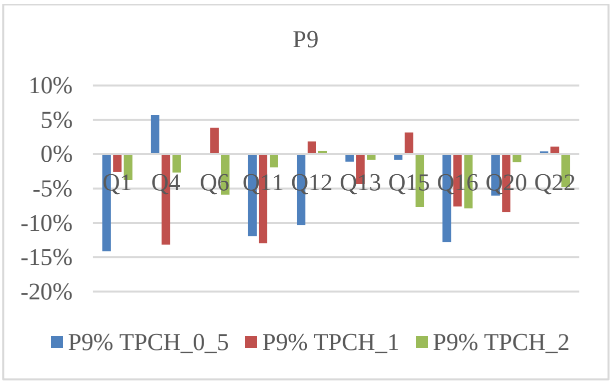

Figure 8. Parameters P9 (%).

The P9 combination (Figure 8) indicates the application of compression optimization technique. Compression can significantly improve the query performance by reducing the amount of data that needs to be read and transferred from the storage to the memory, especially when I/O is a bottleneck. Based on the P9 results in Figure 8, it can be seen that compression is effective for most analyzed queries and amounts of data. For some queries, like Q1, Q4, Q11, and Q16, the improvement is above 10% with a negative percentage difference. If the storage space is an issue, compression can help reduce the data storage footprint, potentially lowering the storage costs. For small data sets that easily fit in the memory, the benefits of compression may be minimal, and in such cases, the overhead of compression and decompression may outweigh the performance gains. For queries Q6, Q15, and Q22, the results are worse for smaller amounts of data in the databases TPCH_0_5 and TPCH_1, while the improvements with a negative percentage difference are obtained with a larger amount of data in the database TPCH_2. So, the compression efficiency depends on the amount of data and also on query complexity.

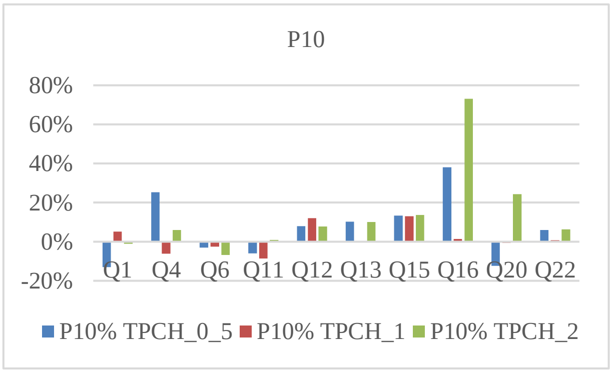

Figure 9. Parameters P10 (%).

The P10 combination (Figure 9) means the application of several optimization techniques: parallelism, CBO, statistics, skew join, map join, and compression. Based on this combination, which includes multiple optimization techniques at the same time, poor performance can be noticed in Figure 9. It is obvious that, in most cases, with the analyzed queries and amounts of data, a positive percentage difference is obtained, which means a slower execution in comparison to default settings, which is not desirable. It can be concluded that using a larger number of techniques is not a guarantee that the desired improvement will be obtained because there is a large number of impacting factors, such as the nature of the data, the specificity and complexity of the query, the hardware configuration, and the available resources.

## 5.3 *Correlation*

Based on the results in Tables 8, 9, 10, Figures 10, 11, 12 are created. They contain a visual representation of the calculated vertical correlation coefficients (all queries per one database) for the considered combinations of configuration parameters P2-P10 with the percentage differences in the query execution time compared to the default settings of P1. The results of the vertical correlation are obtained by dividing the results from Tables 8, 9, 10 with the best result for the observed combination of parameters P vertically for all queries. These results are shown for all queries by database: Figure 10 for TPCH_0_5, Figure 11 for TPCH_1, and Figure 12 for TPCH_2. Based on the correlation coefficients that are in the interval [-1, -0.5], it can be noticed that the best improvements are for the TPCH_0_5 database with combinations P5, P8, and P9; for the TPCH_1 database with combinations P5, P6, P7, and P9; and for the TPCH_2 database with combinations P2, P7, P8, and P9. Based on the correlation, it is shown that a strong negative correlation (close to -1.0) and optimization improvements can be achieved better with larger data sets. In most cases, combinations P5 (paralelism, CBO, statistics), P7 and P8 (join optimizations), and P9 (compression) give the best results in query performance improvements.

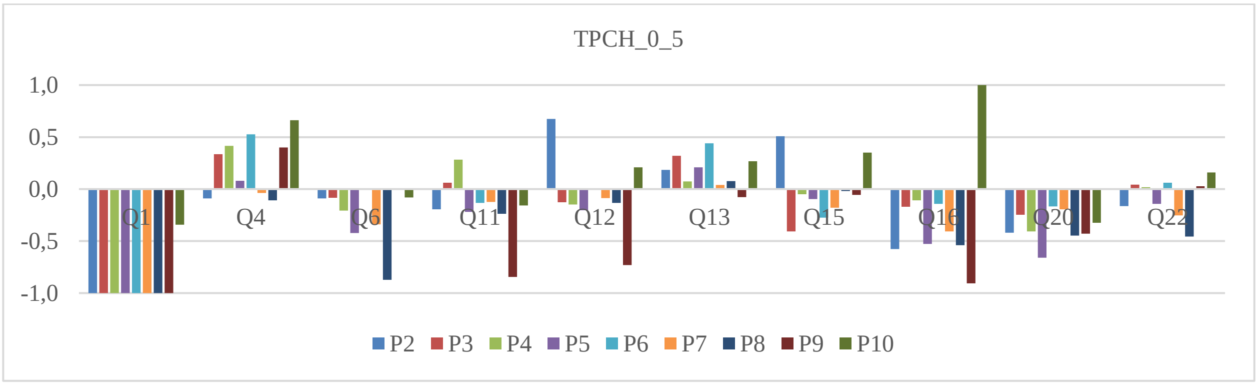

Figure 10. Vertical Correlation r(0.5).

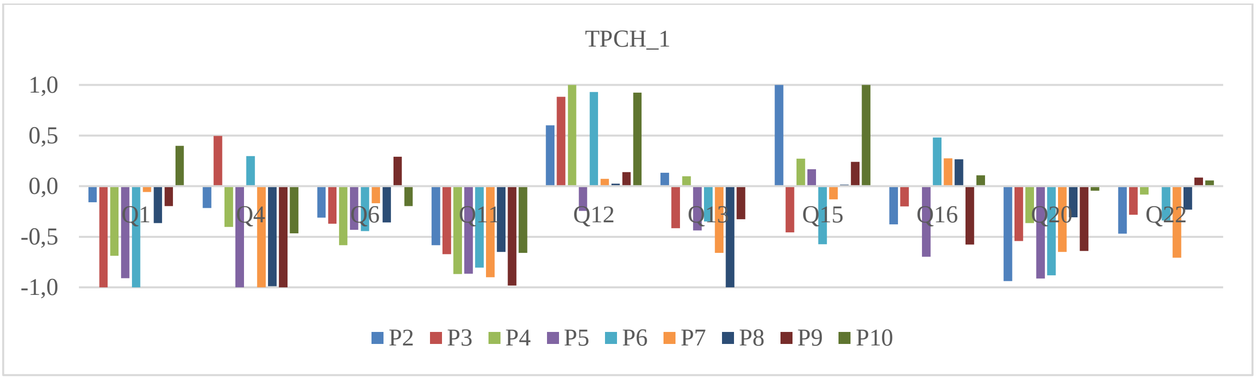

Figure 11. Vertical Correlation r(1).

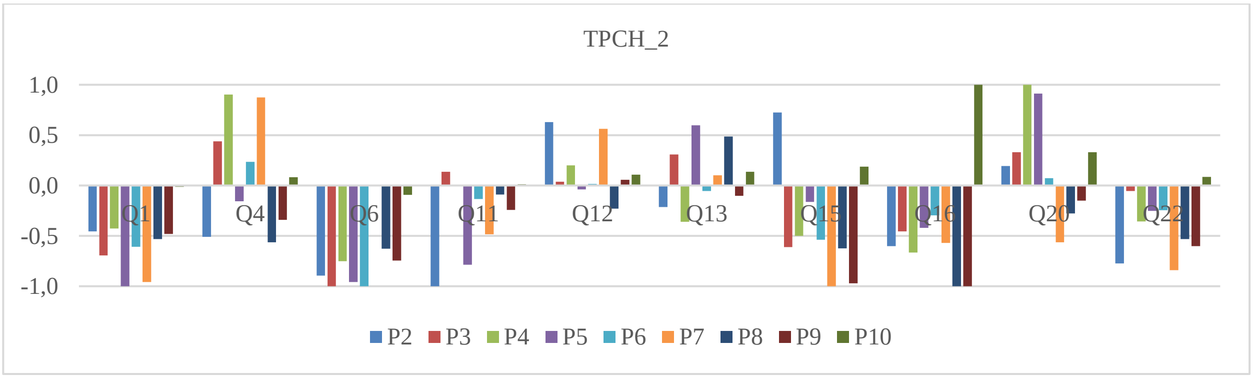

Figure 12. Vertical Correlation r(2).

## 6 Conclusion

Based on the experimental setup and testing of various Hive optimization techniques, it can be observed that their adequate application and optimization of configuration parameters can significantly improve the performance and reduce query execution time. Generally, a combination of parameters gives some degree of improvement and reduces the average query execution time compared to the default Hive configuration. The actual percent difference in the improvement may vary depending on various factors, such as the complexity and specificity of the query, characteristics and amount of data, adequate application, and choice of optimization techniques.

On the basis of the results analysis, it can be concluded that some optimization techniques used individually can give improvements only for some cases, like simple queries or small amounts of data, while a well-chosen combination of optimization techniques and configuration parameters can give improvements for most cases, different queries and amounts of data, like combinations P5 (parallelism, CBO, and statistics), P7 and P8 (join optimizations). But implementing multiple optimization techniques can introduce complexity into the Hive environment, leading to resource competition and increased load, as can be noticed with the P10 combination (parallelism, CBO, statistics, skew join, map join, and compression) that gives bad results in most cases.

Improving the optimization of the management and processing of large amounts of data on a distributed storage platform can be achieved through a combination of best practices, configuration adjustments, and leveraging the capabilities provided by the platform. Considering the fact that there is no ideal combination of the configuration parameters that would give the best results in terms of the performance for all possible queries and data, in order to achieve the best effect, it is necessary to adapt the configuration to specific use cases. This can be achieved by experimenting with different settings to arrive at an optimal configuration for the case at hand. Since the effectiveness of optimization strategies varies by use case and data set, it is essential to perform benchmarking, testing, and profiling to identify areas for improvement and adjust optimization accordingly.

On a concrete platform for a distributed data storage, CDH (Cloudera Distribution for Hadoop), it is shown how, for different amounts of data and the nature of queries, analysis, testing, and profiling of configuration parameters can be performed in order to obtain an adequate improvement in terms of the management and processing of large amounts of data. By carefully configuring the Hive parameters and using appropriate Hive optimization techniques, the query efficiency and performance improvement with a reduced query execution time can be achieved.

## References

[1] Đ. Hadžić, N. Sarajlić, “Methodology for Fuzzy Duplicate Record Identification Based on the Semantic-Syntactic Information of Similarity”, Journal of King Saud University – Computer and Information Sciences, 2018., doi: 10.1016/j.jksuci.2018.05.001

[2] 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.

[3] R. Kune, P. K. Konugurthi, A. Agarwal, R. R. Chillarige, R. Buyya, “The Anatomy of Big Data Computing”, Wiley Online Library, 2015.

[4] V. Nerić, T. Konjić, N. Sarajlić, N. Hodžić, “A Survey on Big Data in Medical and Healthcare with a Review of the State in Bosnia and Herzegovina”, Advanced Technologies, Systems, and Applications III, Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies (IAT), vol. 2, Springer, pp. 494-508, 2019.

[5] A. Y. Zomaya, S. Sakr, “Handbook of Big Data Technologies”, Springer, 2017.

[6] R. Jhajj, “Apache Hadoop Cookbook”, Exelixis Media P. C., 2016.

[7] T. White, “Hadoop: The Definitive Guide”, O’Reilly, 2015.

[8] Cloudera Inc., “Apache Hive Guide”, 2021.

[9] H. Bansal, S. Chauhan, S. Mehrotra, “Apache Hive Cookbook”, Packt Publishing, 2016.

[10] V. Nerić, N. Sarajlić, “A Review on Big Data Optimization Techniques”, B&H Electrical Engineering, vol. 14, pp. 13-18, 2020.

[11] V. Nerić, N. Sarajlić, “Big Data Optimizatization Using Hive”, Elektrotehniški Vestnik, 88(5), pp. 290-298, 2021.

[12] S. Bagui, K. Devulapalli, “Comparison of Hive’s Query Optimisation Techniques”, Int. J. Big Data Intelligence, 2018.

[13] Q. Liu, H. Hong, H. Zhu, H. Fan, “Research and Comparison of SQL Optimization Techniques Based on MapReduce”, International Conference on Computer Science and Application Engineering (CSAE), 2017.

[14] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson et al., “Major Technical Advancements in Apache Hive”, SIGMOD, 2014.

[15] 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.

[16] A. Barman, D. Paranjpe, “Improving the Performance of Hive by Parallel Processing of Massive Data”, International Journal of Recent Development in Engineering and Technology (IJRDT), vol. 7, issue 4, 2018.

[17] A. Gruenheid, E. Omiecinski, L. Mark, “Query Optimization Using Column Statistics in Hive”, IDEAS11, 2011.

[18] “TPC Benchmark H Standard Specification Revision 3.0.1.”, Transaction Processing Performance Council, 1993-2022.

**Vedrana Nerić** graduated in 2006 and received her M.Sc. degree in 2013 from the Faculty of Electrical Engineering of the University of Tuzla, Bosnia and Herzegovina. During her studies, she was presented three Silver and a Gold Medal by the same university. Currently, she is a Ph.D. student. She is employed with Virgin Pulse, Tuzla, as a senior data engineer. In 2014, she was nominated to a teaching assistant in the field of Computer and Information Science at the same faculty.

**Nermin Sarajlić** graduated in 1987 and received his M.Sc. degree in 1997 from the Faculty of Electrical Engineering and Faculty of Electrical Engineering and Mechanical Engineering, respectively, and his Ph.D. degree in 2002 from the Faculty of Electrical Engineering of the University of Tuzla, Bosnia and Herzegovina. His field of interest is the calculation of the coupled electromagnetic-temperature fields, cryptography, and crypto analysis.

**Đulaga Hadžić** graduated from the Faculty of Electrical Engineering of the University of Sarajevo in 1997. He obtained his M.Sc. degree in technical sciences in 2010 from the Faculty of Electrical Engineering of the University of Tuzla, where he received his Ph.D. degree in 2018. He is currently employed as a professor at the Department of Software Engineering at the Polytechnic Faculty of the University of Zenica, Bosnia and Herzegovina. His fields of interest are: web programming, user interfaces, data quality, data mining and cloud computing.