Araştırma Makalesi
Adıyaman Üniversitesi
Mühendislik Bilimleri Dergisi
24 (2024) 428-437

AN ANALYSIS OF APACHE SPARK AND GPU PERFORMANCES ON DATABASE SQL QUERIES FOR DISTRIBUTED NETWORKS

Mehmet TURAN[1], M. Emin TENEKECİ[2], Kemal GÜNER[3*],

[1]Adıyaman Üniversitesi, Bilgi İşlem Daire Başkanlığı, Adıyaman, 02040, Türkiye

[2,3]Harran Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Şanlıurfa, 63300, Türkiye

Geliş Tarihi/Received Date: 01.07.2024 Kabul Tarihi/Accepted Date: 03.10.2024 DOI:10.54365/adyumbd.1508182

ABSTRACT

The use of GPU in different fields and its successful results initiate efforts to use GPU in database systems. It is also effective in distributed systems and computer networks in that it accelerates computational tasks by leveraging parallel processing capabilities across multiple nodes and for tasks that require high computational power, such as network traffic analysis and real-time data processing. Digital transformation in all areas of life has led to the emergence of needs such as increased data diversity and faster data analysis. Upgrading the hardware capacity of the system or conducting software-based studies are possible solutions to analyze this data to meet the needs. In this study, Apache Spark and GPU performance differences are examined in commonly used SQL queries on big data. In this context, SQL queries such as grouping, sorting, and filtering, which are widely used in data analysis, are used. While the queries performed with the GPU showed similar results in simple queries compared to the queries performed with Apache Spark, the GPU was completed 3x faster in queries requiring calculation.

Keywords: *GPU, Apache Spark, Distributed Networking, Big Data, HPC*

DAĞITIK AĞLAR İÇİN VERİ TABANI SQL SORGULARINDA APACHE SPARK VE GPU PERFORMANSLARININ ANALİZİ

ÖZET

Her geçen gün farklı bir alanda kullanılmaya başlanan ve başarılı sonuçlar sergileyen GPU'nun veri tabanlarında kullanılmasına yönelik çalışmalar giderek yaygınlaşmaktadır. Ayrıca dağıtık sistemlerde ve bilgisayar ağlarında da birden fazla düğümde paralel işleme yeteneklerinden yararlanarak hesaplama görevlerini hızlandırmaya ve yüksek hesaplama gücü gerektiren ağ trafiği analizi, gerçek zamanlı veri işleme gibi görevlerde etkin olmaktadır. Hayatın her alanda gerçekleşen dijital dönüşüm veri çeşitliliğinde artış, verilerin daha hızlı analiz edilebilmesi vb. ihtiyaçların ortaya çıkmasına neden olmuştur. Bu verilerin analizi için sistem donanım kapasitesinin artırılması veya yazılım temelli çalışmalar ile ihtiyaçların karşılanabilmesine yönelik çözümler bulunmaktadır. Bu çalışmada ise büyük verilerde Apache Spark ve GPU'nun yaygın olarak kullanılan SQL sorgularındaki performans farklılıkları incelenmiştir. Bu kapsamda veri analizinde genel olarak kullanılan gruplandırma, sıralama ve filtreleme gibi SQL sorguları kullanılmıştır. GPU ile gerçekleştirilen sorguların Apache Spark ile gerçekleştirilen sorgulara göre basit sorgularda benzer sonuçlar sergilerken, hesaplama gerektiren sorgularda GPU'nun 3x kadar daha kısa sürede sonuçlandırmıştır.

Anahtar Kelimeler: *GPU, Apache Spark, Dağıtık Ağlar, Büyük Veri, HP*

e-posta[1] : mturan@adiyaman.edu.tr ORCID ID: https://orcid.org/0000-0002-8038-0749

e-posta[2] : etenekeci@harran.edu.tr ORCID ID: https://orcid.org/0000-0001-5944-4704

e-posta[3] : kguner@harran.edu.tr ORCID ID: https://orcid.org/0000-0003-3495-9044 (Sorumlu Yazar)

1. Introduction

Different concepts and approaches have emerged as a result of the developments aimed at increasing the quality of life. Depending on the object used, it is of great importance in terms of effective use of resources, determining new strategies for solving problems, increasing satisfaction and reducing errors, etc. The concept of data, whose value increases depending on its variety and size, becomes meaningful if it is possessed on a regular base and accurately. By utilizing data in many fields, it is possible to make inferences about how to move forward and what kind of measures should be taken. Data, which was kept limited until recently, has led to the emergence of new systems or solutions in larger volumes, more variables, and faster increases in parallel with the digital transformation that has taken place in recent years. While there was no significant problem in the studies that were widely used and carried out with computers in the past, the growing amount of data and the need to get results from these data in a shorter time have led to the emergence of different technologies.

Initially, CPUs, disks, and RAMs were used for faster processing for long-running operations. When the system used was not sufficient to meet the needs in terms of performance, the problem was solved with new hardware with a higher capacity. Although the developments in the hardware sector have reached a certain point, the amount of data used in processing not only has begun to increase rapidly but also it has been used in different fields. Real-time access to data and the ability to process it in a shorter time are of great importance for sectors operating in similar fields to compete and be advantageous [1].

Efforts to improve the performance of data processing such as analysis, storage, and visualization are of great importance. This has led to significant developments depending on the impact of big data analysis and evaluations of these analyses [2]. Moreover, the state-of-the-art open-source parallel processing frameworks/programming models such as Impala, Hadoop, Spark, and MapReduce have evolved to overcome the issues of distributed big data. In these frameworks/models, the need for more efficient processing of data is met by adding a new server when the server is insufficient with horizontal scalability. Apache Spark, one of the scalable structures used in big data processing, stands out amongst others. With Spark, the system can be scaled to thousands of nodes. With its ability to perform in-memory computation, Apache Spark is preferred because it gives faster results than other big data processing frameworks [3-4]. Today, with the increase in RAM memory capacities, the increase in performance with memory access instead of disk access offers new opportunities to those working in this field [5-6]. In applications utilizing Spark, the size and type of input data and the design of algorithms have a great impact on the performance of the computation process [7-10].

Significant progress has been achieved in GPU technology with the stagnation of developments in the CPU compared to the past, the ability to program GPUs, and their use in different fields. Modern GPUs are not only powerful graphics cards, but they are also preferred for computationally demanding operations due to their parallel computing capability and faster processing compared to traditional CPUs due to their high memory bandwidth [11-13]. The high performance in operations performed with GPU has led to an increase in the number of studies in this field and the concept of GPU Computing has become widespread. Compared to traditional microprocessors, GPUs are seen as a good alternative due to their high capacity in computationally demanding operations. GPUs allow not only graphics operations and calculations but also the execution of any algorithm. The complexity of the hardware used and the complexity of the user functions play an important role in achieving the success obtained from GPU applications [14].

Parallel data processing and the ability to perform parallel computation are the key features that make GPUs stand out. In the microprocessors being developed for GPUs, efforts are being made to increase not only the thread performance but also the number of cores. While the GPU's performance and potential hold great promise for new computer systems, its architecture and programming model are distinctly different from other uniform processors. Increasing the number of programmable processor cores in the GPU has a significant impact on increasing the total system output [15-16]. Furthermore,

combining processors allows for very effective load balancing. This is because any processing function can utilize the entire array of processors.

Achieving high performance in GPU-executed operations depends on using the full memory bandwidth of the GPU and designing the application in a way that is compatible with parallelization. CUDA's scalable programming model enables parallel computing applications to handle large numbers of workloads while allowing parallel processor cores to scale [17]. The computational performance obtained from the GPUs can be achieved by using more than one CPU, but it is not as successful as GPUs in terms of performance/power consumption. GPUs are very efficient in terms of the work they do and the energy they consume [18]. Applications that are not organized according to GPU operation cause a great loss in terms of the amount of energy consumed due to the inability to use GPU capacity effectively.

With the digital transformation, every object has become part of a network. The fact that these objects communicate with each other according to certain rules and the unique addressing of the data has led to the emergence of different approaches. With these developments, it has become possible for every object that is part of daily life to be part of a network, to record the activities of these objects depending on their functioning, and to transfer these records to certain centers depending on the user's permission. This situation makes it easier for service providers to work on many aspects such as development and security based on this data. Big data is generated in every area where technology is used, such as internet service providers, servers, and applications. In the relevant units of many large or small enterprises, there are systems for tracking activities. Billions of data can be recorded in these systems every day. It is of great importance that the processes related to the analysis of these data can be carried out in a shorter time and reports can be created for the needs, and the loss that will occur in the sectors where these data are used can be minimized and measures can be taken. Therefore, it becomes a necessity rather than a need to realize the data analysis process as earliest as possible.

There are many studies on the processing of databases using GPU. Due to the programmability of the GPU, it is seen as a potential hardware that can show higher performance than CPU-based applications in many areas. In this study, the effect of the GPU's parallelization and internal memory advantage on the performance of in-memory data analytics operations on the congestion that prevents full performance in data transfer between CPU and RAM memory is observed. Although the intended use of the data is different, the underlying variables and characteristics are similar. For this purpose, the performance of the GPU in processing log data by considering the processing power and operating principles of the GPU and the performance differences of Apache Spark in the same processes are analyzed.

With two different computing technologies, GPU and Apache Spark, the same SQL queries for commonly used operations such as querying, filtering, grouping, and computation in data were repeatedly performed at different times and for certain numbers of times in two different models and the completion times of the operations were compared.

2. Related Work

Parallel communication and synchronization, as well as sustainable scalability, is one of today's hottest topics. Organizations that direct their activities based on data analysis have a great advantage in their competition with each other [19]. This situation requires the development of new tools to solve the challenges caused by the volume, diversity, and speed characteristics of big data.

Since NVIDIA introduced parallel computing on GPUs with CUDA in 2006, there have been many efforts for different purposes. Although GPU-based database applications are still at the beginning of the road, serious work continues in this field [20]. GPU-based systems can be a good option in terms of both energy and cost to accelerate applications in big data processing and intensive computing. While GPUs are a good option for reducing computational load and efficient resource scheduling, they also have some challenges. The most important challenge is that achieving high performance depends on advanced application development knowledge. Wrede and Ernsting [14] argue that GPUs can provide

significant acceleration in certain situations, but the complexity of the hardware and its functions is a determining factor in achieving this result.

With the introduction of CUDA and OpenCL, the programming of GPUs has become much more efficient. GPU cards, which were mathematically used to greatly increase video processing and transformation tasks, were then used in different areas. He et al. [21], using CUDA, designed a join algorithm that also includes sorting and achieved 2-7 times the performance compared to CPU. Again, Bakkum and Skadron [22] concluded simple queries performed by executing SQLite on GPU within the virtual machine infrastructure 20x faster than SQLite.

Wu [23], considering the limited memory capacity of the GPU, the algorithms applied to the data cannot be used directly for big data. With CUDA unified virtual addressing, data is not necessarily copied to the GPU, as the GPU can directly access the data in CPU memory. However, the performance of the algorithm used will be decisive, depending on the PCIe bandwidth. According to He et al. (2009), the bandwidth for data exchange between GPU and CPU is limited. In addition, the CPU does not allow dynamic memory allocation for GPU cores, so algorithms need to be designed carefully.

Breß et al. [24] argued that a hybrid approach that decides whether to execute the query on the GPU or CPU with a self-adjusting query editor would be a better solution in terms of performance with the HyPE application they developed. Again, Ilić et al. [25], with the application they developed, stated that the use of two processing centers together, depending on the scheduling library to execute the work to be done on the CPU or GPU, contributed significantly to the performance increase.

Li et al. [26] have achieved a performance increase of up to 18x in different machine learning applications with the HeteroSpark platform but emphasized the insufficient memory size when using large data in experiments with a single GPU, and the complexity of data partitioning in multi-GPU applications. In a detailed analysis of queries performed on CPU and GPU, the GPU can only be used for specific queries that can be fixed in memory, which causes problems in compatibility with data storage systems [27].

3. Material & Method

The materials used in this study, which analyzes big data queries on Apache Spark and GPU, consist of hardware, software, and data. The hardware includes a Tesla T4 GPU, 12 GB DDR RAM, and an Intel Xeon 2200 MHz CPU. The software is a Python application using Apache Spark and BlazingSQL libraries. Apache Spark is an open-source, scalable, fault-tolerant, easy-to-use & fast distributed computing framework that runs on datasets. It is also widely used in Java, Scala, Python, and R due to its development APIs and extensive libraries. It supports code reuse across multiple workloads such as real-time data analysis and engineering, graphics processing, data science, and machine learning. For example, it enables parallelization of machine learning algorithms and data transformation logic, enabling fast processing on distributed storage systems of varying sizes and types. Another important reason why Apache Spark preferred in this study is that Spark applications can perform approximately 100 times faster in-memory operations on Hadoop clusters and 10 times faster on disk. In addition, the MapReduce feature allows Spark to be implemented independently. On the other hand, since Apache Spark is an open-source, distributed processing system used for big data workloads, it has some limitations inherent to distributed systems, such as delays, congestion, packet loss, and varying speed due to internet infrastructure services.

Table 1. Processing parquet file with BlazingSQL

```
....
# Create table from parquet file
bc.create_table(deletions, '/content/deletions.parquet')
# SQL Query processing
result = bc.sql('SELECT count(*) FROM main.deletions GROUP BY
creator(key)').get() result_gdf = result.columns
#Showing results print(result_gdf)
....
```

BlazingSQL, on the other hand, offers the convenience of performing the same operations on the GPU that cuDF does on the GPU with the SQL interface instead of the DataFrame functions of ETL raw data performed on the GPU with cuDF (RAPIDS). As in cuDF, the Python language in BlazingSQL offers ease of use and supports many SQL functions, although it cannot perform all the SQL functions. It is easier to create tables with BlazingSQL because Parquet holds metadata. Table 1 shows an example of using BlazingSQL with Python language. The last component of our study is the data, which is approximately 8GB in size and consists of 62,000,000 rows of deletion activity logs for March 2013 provided by the Freebase API under the CC-BY license.

Table 2. The specification of Tesla T4 GPGPU

Specs	
GPU Architecture	NVIDIA Turing
Number of NVIDA Tensor Core	320
Number of NVIDIA Cuda Core	2560
Single Precision	8.1 TFLOPS
Mixed Precision	(FP16/FP32) 65 TFLOPS
INT8	130 TOPS
INT4	260 TOPS
GPU Memory (GPU Belleği)	16 GB GDDR6 300 GB/sec
ECC	Yes
Interconnect Bandwidth	32 GB/sec
System Iterface	x16 PCIe Gen3
Form Factor	Low-Profile PCIe
Thermal Solution	Pasif
Compute APIs	CUDA, NVIDIA TensorRT™, ONNX

Since cuDF and BlazingSQL libraries are compatible with this GPU card, Tesla T4 Graphics processing unit (GPU) is preferred. Additionally, T4 puts forward the benefits of high throughput and lower power consumption as well.

For query performance to be realized in a shorter time in big data, it is necessary to use column-oriented data, which is also suitable for the operation of the GPU. For this purpose, our activity records in CSV format have been converted to column-oriented Parquet format that can be used in both Apache Spark and our GPU application. Converting our 8GB raw CSV data into Parquet format resulted in a compressed data file of approximately 2GB. The reduction in data size is due to the fact that Parquet stores and compresses the data in binary format.

The only difference between row-oriented databases and column-oriented databases is that when a row with row-oriented data is called, the data that is not needed in the row is included in the process, and then the target data can be reached by filtering the unwanted part. In column-oriented data, only the desired row of the relevant column is called (Figure 1)

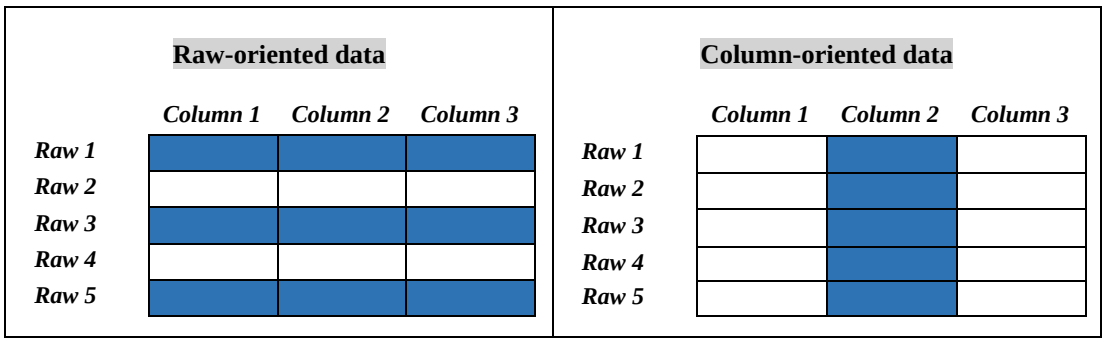

Figure 1. Row and Column oriented databases

The types of queries to be performed for the data are given in Table 3. The reason why these queries were chosen is that they are widely used querries in many applications. The queries were executed at least 5 times at different times with both GPU and Apache Spark and the average execution time was taken into account for experimental results.

Table 3. Query types

1.	Selection of top 10 records with all fields
2.	Sort by any field
3.	Top 10 records with selected specific fields.
4	Number of total records in any domain
5.	Number of different records in any domain
6.	Grouping by a specific field and the number of records in that group
7.	Choosing according to a certain condition

The details of the query types mentioned in Table 3 for SQL, Spark DataFrame and cuDF DataFrame queries in our Python implementation can be seen in Table 4.

Table 4. Queries for SQL, cuDF, and Apache Spark

1.	SQL	SELECT * FROM Deletions LIMIT 10
	cuDF DataFrame	cuDF.head(10)
	Spark DataFrame	sparkDF.limit(10)
2.	SQL	SELECT * FROM Deletions ORDER BY creation_timestamp
	cuDF DataFrame	cuDF["creator"].sort_values(['creator'])
	Spark DataFrame	sparkDF.orderBy("creation_timestamp")
3.	SQL	SELECT creator, deletor FROM Deletions LIMIT 10
	cuDF DataFrame	cuDF[["creator","deletor"]].head(10)
	Spark DataFrame	sparkDF.select("creator","deletor").limit(10)
4	SQL	SELECT COUNT(*) FROM Deletions
	cuDF DataFrame	cuDF["creator"].count()
	Spark DataFrame	sparkDF.count()
5.	SQL	SELECT COUNT(DISTINCT creator) FROM Deletions
	cuDF DataFrame	cuDF["creator"].unique().count()
	Spark DataFrame	sparkDF.select("creator").distinct().count()
6.	SQL	SELECT COUNT(*) as NumOfEvents, creator FROM Deletions GROUP BY creator
	cuDF DataFrame	cuDF.groupby("creator").agg({"creator":"count"})
	Spark DataFrame	sparkDF.groupBy("creator").count().select("creator","count")
7.	SQL	SELECT * FROM Deletions WHERE deletor = '/user/funderhill'
	cuDF DataFrame	cuDF. query('deletor == '/user/funderhill')
	Spark DataFrame	sparkDF.where("deletor == '/user/funderhill'")

While processing with Apache Spark, the database is selected through the Driver, then the queries are interpreted with the help of SparkContext, and the related calculations are performed with the help of the Cluster Manager, and the task is completed by performing the tasks on the JVM Nodes where the executors are located (Figure 2).

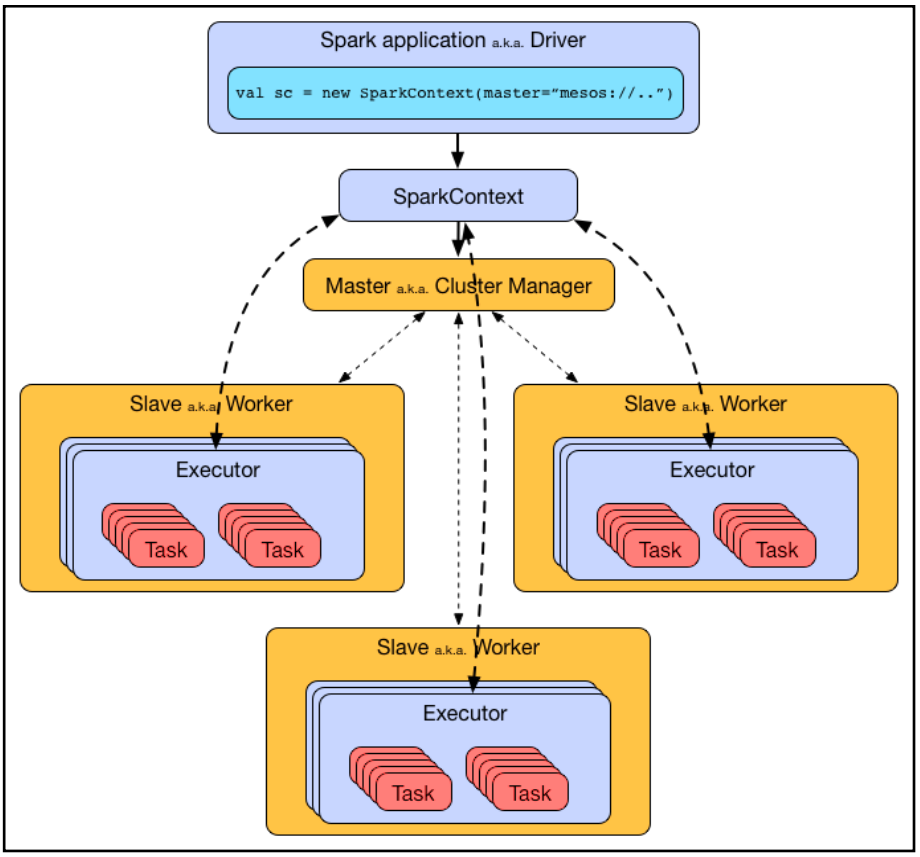

Figure 2. Apache Spark query workflow [28].

After BlazingSQL makes the SQL query compatible on cuDF, Python functions are converted to PTX (the pseudo-assembly language used in CUDA programming) code. The request is then converted to PTX code with CUDA JIT, and finally, the query is executed on the CUDA device using Python Numba and CUDA driver API, and the results are transferred to the CPU (Figure 3).

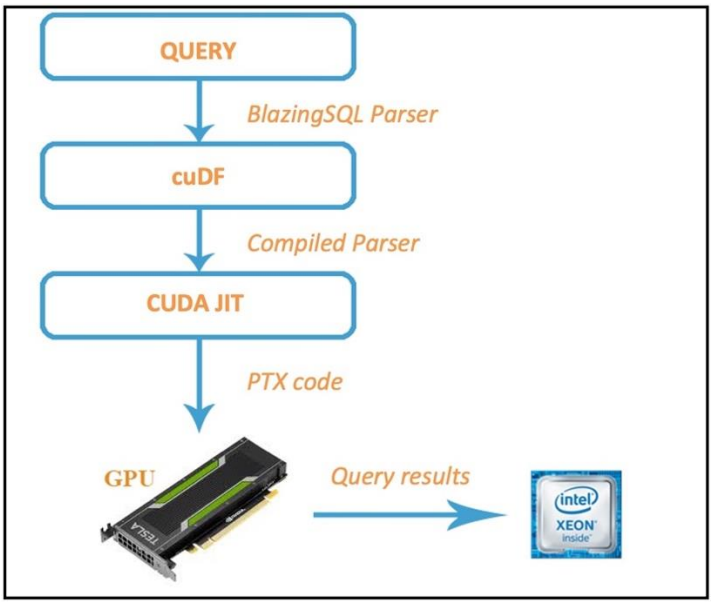

Figure 3. GPU query workflow

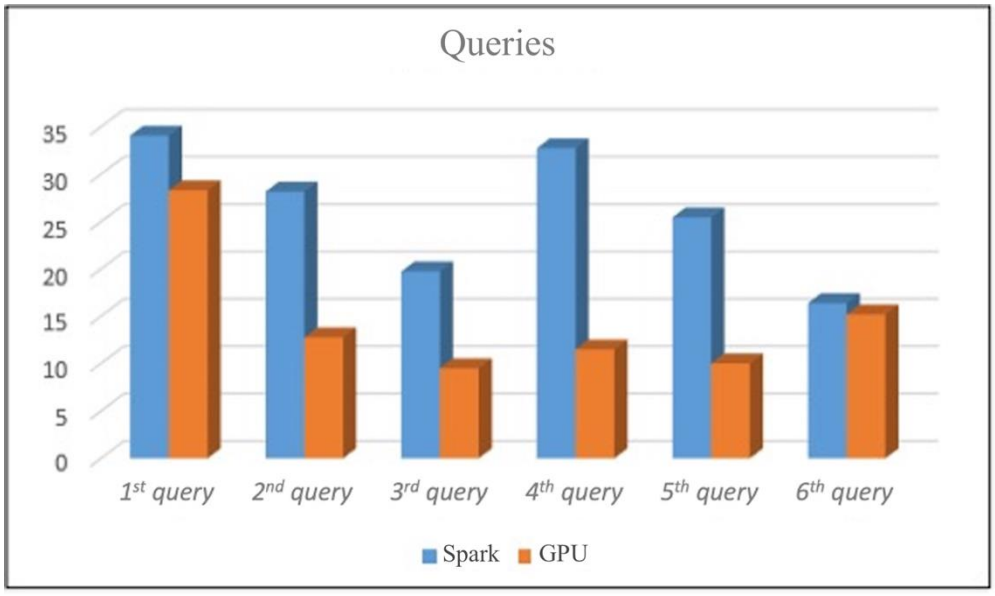

Figure 4. The execution times of the queries

In Figure 4, where the performances of two different computing technologies are examined, the execution times of the query operations are considered. No significant difference was observed in the 1st query, where there was no calculation process, and the 6th query, in which conditional filtering was performed. On the other hand, in sorting, grouping, and conditional sorting, the GPU completed the process as little as 3 times faster than Apache Spark (Figure 4).

Moreover, the potential of the GPU in parallel computing is used in the analysis of big data, and its effects on performance are examined. For this purpose, Apache Spark, which stands out due to its success in in-memory computation, and GPU's SQL query execution times are evaluated. For the effective use of GPU, it is of great importance that the application is developed in accordance with the structure and functioning of GPU. While no significant difference was observed in the execution times of requests for simple queries, the GPU had positive effects on the performance increase for computationally demanding queries.

The large volume characteristic of big data also requires high-capacity memory allocation. The limited GPU memory causes the used data size to be within these limits. At this point, in GPU workstations with multiple GPUs, GPUs can be parallelized among themselves to work with larger data.

4. Results and Future Work

In this study, the characteristics of big data, GPU's potential in computational operations, and Apache Spark's power in big data processing are discussed. We used a column-oriented data storage format that both reduces unnecessary work assigned to the CPU and fits the GPU's architecture. In terms of the performance of queries performed with the GPU, it was examined whether the computational power of the GPU would have the same effect on data querying as Apache Spark queries performed with the CPU. For this purpose, activity (log) records kept for different purposes in many sectors were used. The same queries were executed at different times with certain periods and the average execution times for each query type were compared.

Queries were executed on both CPU and GPU, and the same queries were executed in different time periods and a certain number of times, and each query was compared separately. Overall, no significant performance difference was observed in basic queries, but in queries that required sorting, grouping,

and conditional computation, the GPU completed the query in approximately 3x less time. As a result, it was seen that it is possible to perform operations in a shorter time by using GPU in data querying as in many computationally demanding operations.

Nowadays, there are more and more studies on the use of GPUs in terms of computational success and energy-efficient use of big data. Deep learning, machine learning graphics applications, engineering applications, and applications used extensively in scientific research also have GPU support. Recently, there has been much scientific research on the use of GPU in big data solutions such as distributed computing and networking. It will not be a surprise that large-scale data solutions such as Hadoop, Spark, and Hive will have GPU support in the near future.

Conflict of Interests

The authors of the article declare that they have no personal or financial conflict of interest with any institution, organization or person.

References

[1] Einav L, Levin J. The Data Revolution and Economic Analysis. Innovation Policy and the Economy. University of Chicago Press 2014; 14(1): 1-24

[2] Mishra R, Sharma R. Big Data: Opportunities and Challenges. International Journal of Computer Science and Mobile Computing. 2015; 4(6): 27-35.

[3] Spark 101: What Is It, What It Does, and Why It Matters. https://mapr.com/blog/spark-101- what-it-what-it-does-and-why-it-matters (Erişim Tarihi: 03.06.2024)

[4] Ahmed N, Barczak ALC, Susnjak T, Rashid MA. Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. Journal of Big Data 7. 2020;110.

[5] The New Era of Big Data. https://www.forbes.com/councils/forbestechcouncil/2023/05/24/the-new-era-of-big-data/ (Erişim Tarihi: 24.5.2023)

[6] Guner K, Kosar T. Energy-Efficient Mobile Network I/O. In: IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 2018; 1-6.

[7] Wang K, Khan MMH. Performance Prediction For Apache Spark Platform. In: High Performance Computing and Communications (HPCC), IEEE 7th International Symposium On Cyberspace Safety and Security (CSS), IEEE 12th International Conference On Embedded Software and Systems (ICESS), IEEE 17th International Conference, New York, 2015; 166-173.

[8] Tang S, He B, Yu C, Li Y, Li K. A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications. In: IEEE Transactions on Knowledge and Data Engineering. 2022; 24(1):71-91.

[9] Lunga D, Gerrand J, Yang L, Layton C, Stewart R. Apache Spark Accelerated Deep Learning Inference for Large Scale Satellite Image Analytic In IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 2020 13(1):271-283

[10] Tang Z, Zeng A, Zhang X, Yang L, Li K. 2020, Dynamic memory-aware scheduling in spark computing environment. Journal of Parallel and Distributed Computing 2020 141: 10-22

[11] Şahin H. Real Time Orthorectification Of Images By General Purpose Computation On Graphical Processing Units Method. Ph.D. thesis Istanbul: Istanbul Technical University: 2016.

[12] Tirmazi M, Basat RB, Gao J, Yu M. Cheetah: Accelerating Database Queries with Switch Pruning. In: ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2020; 2407–2422

[13] Guner K, Nine MSZ, Bulut MF, Kosar T. Fasthla: Energy-efficient mobile data transfer optimization based on historical log analysis. In: 16th ACM International Symposium on Mobility Management and Wireless Access, 2018; 59-66.

[14] Wrede F, Ernsting S. Simultaneous CPU-GPU Execution of Data Parallel Algorithmic Skeletons. International Journal of Parallel Programming. 2017; 46(1): 42-61

[15] Wolf M. High-Performance Embedded Computing: Applications in Cyber-Physical Systems and Mobile Computing. 2nd ed. Morgan Kaufmann; 2014; 59-138.

[16] Lee R, Zhou M, Li C, Hu S, Teng J, Li D, Zhang X. The art of balance: a RateupDB™ experience of building a CPU/GPU hybrid database product. In: proceedings of the VLDB Endowment. 2021; 14(12); 2999–3013.

[17] Nvidia 2011a. Cuda C Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (Erişim Tarihi: 21.02.2018)

[18] Nvidia 2011b. Energy Efficiency. http://www.nvidia.com/object/gcr-energy-efficiency.html (Erişim Tarihi: 22.02.2018)

[19] The Rise of the Data Economy: Driving Value Through Internet of Things Data Monetization. https://www.ibm.com/downloads/cas/4jroldq7 (Erişim Tarihi: 07.11.2017)

[20] Farooqui N, Roy I, Chen Y, Talwar V, Barik R, Lewis BT, Shpeisman T, Schwan K. Accelerating Data Analytics on Integrated Gpu Platforms via Runtime Specialization. International Journal of Parallel Programming. 2016; 46: 336-375.

[21] He B, Yang K, Fang R, Lu M, Govindaraju NK, Luo Q, Sander P. (2008). Relational Joins on Graphics Processors. In: proceedings of the Acm Sigmod International Conference on Management of Data, Sigmod2008, Vancouver, Canada; 2008; 511-524.

[22] Bakkum P, Skadron K. 2010. Accelerating Sql Database Operations on A Gpu with Cuda. In Gpgpu10: Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processing Units, Newyork; 2010; 94-103.

[23] Wu H. Acceleration and Execution of Relational Queries Using General Purpose Graphics Processing Unit (GPGPU). Ph.D. thesis. Atlanta: Georgia Institute of Technology; 2015.

[24] Breß S, Schallehn E, Geist I. Towards Optimization of Hybrid CPU/GPU Query Plans in Database Systems. In: New Trends in Databases and Information Systems 2013; 27-35.

[25] Ilić A, Pratas F, Trancoso P, Sousa L. High-Performance Computing on Heterogeneous Systems: Database Queries on CPU and GPU. In: High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives, 2011; 202–222.

[26] Li P, Luo Y, Zhang N, Cao Y. Heterospark: A Heterogeneous CPU/GPU Spark Platform for Machine Learning Algorithms. In: International Conference of Networking Architecture and Storage, Boston, USA; 2015; 347-348.

[27] Yuan Y, Lee R, Zhang X. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. In: Proceedings of The Vldb Endowment, 2013; 6(10): 817-828.

[28] Khourdifi Y, Elalami A, Bahaj M, Zaydi M, Er-Remyly O. Chapter 9 - Framework for integrating healthcare big data using IoMT technology. In: Advances in ubiquitous sensing applications for healthcare, Computational Intelligence for Medical Internet of Things (MIoT) Applications, Academic Press 2023; 14; 191-210