UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Tofig Bakhshiyev

Hard and Soft Tuning of Spark Ecosystem Toward Query Energy Efficiency

Master's Thesis (30 ECTS)

Supervisor(s): Simon Pierre Dembele, PhD

Tartu 2024

Hard and Soft Tuning of Spark Ecosystem Toward Query Energy Efficiency

Abstract:
This thesis explores the energy efficiency of executing TPCH queries within the Apache Spark framework, explicitly focusing on diverse file formats (Parquet, CSV, Avro, and TBL) and varying partition sizes in a standalone configuration. The assessment measures energy consumption during the data reading and query processing phases. Initial comparisons are made regarding the characteristics of Parquet, CSV, and Avro formats, analysing their impact on the query performance of Spark. Additionally, the study investigates Spark's standalone configuration, scrutinising cluster settings, resource allocation, and hardware optimizations that influence energy usage during query execution. An integral part of this exploration involves comprehending how different partition sizes influence energy consumption. The evaluation systematically assesses the impact of partition sizes on IO operations, data shuffling, and overall energy consumption during query processing. Utilising TPCH queries as benchmarks, experiments are conducted across various file formats, partition sizes, and configurations. The outcomes offer practical insights for enhancing energy efficiency in Spark-based big data processing. This research contributes to the broader discourse on sustainable data processing, guiding practitioners to make energy-conscious decisions in Apache Spark environments.

Keywords: Energy evaluation, Partitioning, distributed systems, data processing, file formats

CERCS:P170, Computer Science

Sparki ökosüsteemi kõva ja pehme häälestamine päringute energiatõhususe suunas

Lühikokkuvõte:
Käesolevas töös uuritakse TPCH päringute täitmise energiatõhusust Apache Sparki raamistikus, keskendudes selgesõnaliselt erinevatele failivormingutele (Parquet, CSV, Avro ja TBL) ja erinevatele partitsioonide suurustele iseseisvas konfiguratsioonis. Hindamisel mõõdetakse energiakulu andmete lugemise ja päringu töötlemise faasis. Esmalt võrreldakse Parquet, CSV ja Avro formaatide omadused, analüüsides nende mõju Sparki päringute sooritamisele. Lisaks uuritakse Sparki eraldiseisvat konfiguratsiooni, uurides klastri seadistusi, ressursside jaotust ja riistvara optimeerimist, mis mõjutavad energiakasutust päringu täitmise ajal. Selle uurimise lahutamatu osa on mõista, kuidas erinevad partitsioonide suurused mõjutavad energiatarbimist. Hindamisel süstemaatiliselt hinnatakse partitsioonide suuruse mõju IO-operatsioonidele, andmete segunemisele ja üldisele energiatarbimisele päringute töötlemisel. Kasutades TPCH päringuid kontrollmõõduna, tehakse katseid erinevate failiformaatide, partitsioonide suuruse ja konfiguratsioonide vahel. Tulemused pakuvad praktilisi teadmisi energiatõhususe suurendamiseks Sparkipõhises suurandmete töötlemises. See uurimus aitab kaasa laiemale arutelule säästliku andmetöötluse teemal, suunates praktikuid tegema energiateadlikke otsuseid Apache Sparki keskkondades.

Võtmesõnad: Energiatarbimine, suurandmed, hajussüsteemid, andmetöötlus

CERCS:P170, Arvutiteadus

Contents

1 Introduction

In this thesis, we comprehensively explore the energy efficiency considerations associated with executing TPCH queries in Apache Spark. Our focus centres on evaluating the impact of different file formats—specifically, Parquet, CSV, and Avro—along with the influence of varying partition sizes in a standalone setup. Throughout our investigation, we meticulously measure energy consumption during data reading and query processing.

The initial phase of our study involves a comparative analysis of Parquet, CSV, and Avro formats, shedding light on their respective characteristics and consequential effects on Spark's query performance. Beyond file formats, we extend our inquiry to encompass Spark's standalone configuration. This entails a detailed examination of cluster settings, resource allocation strategies, and hardware optimizations, all of which play a pivotal role in shaping energy usage during the execution of queries.

A critical dimension of our exploration is understanding how different partition sizes impact energy consumption. This entails a systematic evaluation of the influence of partition sizes on IO operations, data shuffling processes, and the overall energy footprint during query processing.

Using TPCH queries as benchmarks, our experiments span diverse file formats, partition sizes, and configurations. The outcomes of this research aim to provide practical insights, offering guidance to practitioners seeking to optimise energy efficiency in Apache Spark-based big data processing. By contributing to the broader discourse on sustainable data processing, our findings aspire to empower decision-makers with the knowledge to make energy-conscious choices in Apache Spark environments.

2 Background

This section presents an overview of the technologies and methodologies employed in this thesis. This covers Spark SQL, various file formats, namely Parquet, Avro, TBL, and CSV, and general computer energy consumption discussion. Additionally, the motivation behind undertaking this thesis is explored.

2.1 Apache Spark

Apache Spark is a rapid cluster computing solution created for swift data processing. Built upon Hadoop MapReduce, it expands the MapReduce paradigm to support a broader range of tasks, including interactive queries and real-time stream processing. Spark's standout attribute lies in its utilisation of in-memory computing across clusters, significantly enhancing application processing speeds. It has five main components [Spa24a], namely Spark Core, Spark SQL, Spark Streaming, MLib, and GraphX as shown in Figure 1.

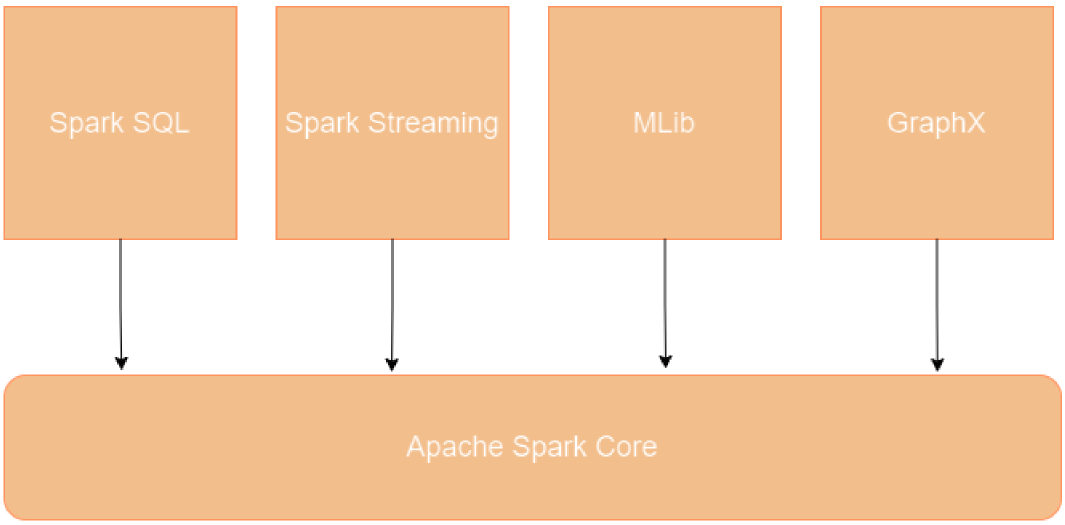

Figure 1. Apache Spark Components

2.1.1 Apache Spark Core

Apache Spark Core is the foundational framework of the Apache Spark platform. It provides distributed task scheduling, memory management, and fault recovery. It also includes Java, Scala, and Python APIs, enabling developers to interact with the Spark cluster and perform distributed data processing tasks [Spa24b].

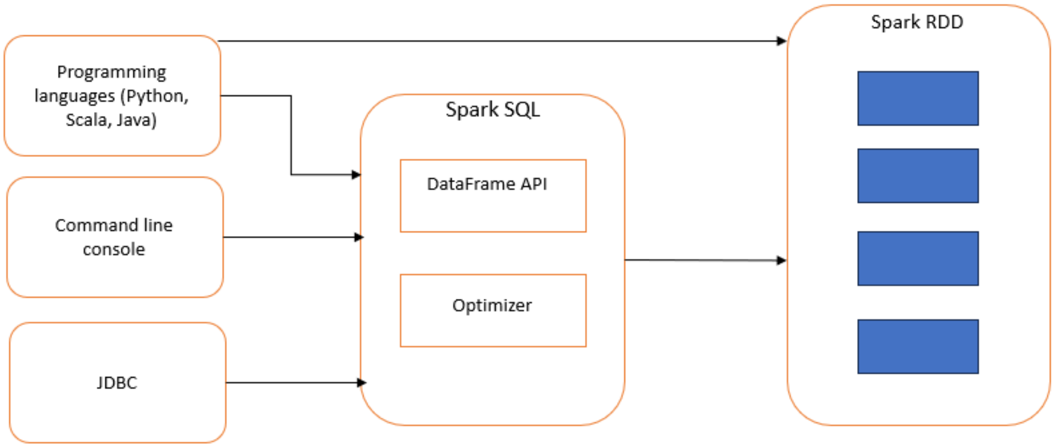

Figure 2. Spark SQL interaction with Spark RDD

2.1.2 Spark SQL

Spark SQL is a component of Apache Spark that enables the processing of structured and semi-structured data using SQL queries as shown in Figure 2. It allows users to seamlessly integrate SQL queries with Spark programs, providing a unified batch and real-time data processing platform. Spark SQL processes data by leveraging a query optimizer called Catalyst, which translates SQL queries into a physical execution plan. The process involves parsing, analysis, optimization, code generation, and execution [AXL+15], [Spa24g].

2.1.3 Apache Streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams [SDC+16], [Spa24h]. It allows developers to process data streams in real time using the same programming model as batch processing, simplifying the development of real-time analytics applications.

2.1.4 Apache MLib - Machine Learning Library

MLlib is Apache Spark's scalable machine learning library. It provides a wide range of distributed machine-learning algorithms and utilities, allowing users to build and deploy machine-learning models at scale [SDC+16], [Spa24e]. MLlib supports various tasks such as classification, regression, clustering, collaborative filtering, and dimensionality reduction.

2.1.5 Apache GraphX

GraphX is Apache Spark's graph processing library. It provides an API for manipulating and analysing graphs and graph-parallel computation. GraphX enables users to express graph computation within the Spark framework, allowing seamless integration with other Spark components for efficient large-scale graph processing [[SDC+16], [Spa24d]].

2.1.6 Spark RDD

Apache Spark RDD (Resilient Distributed Dataset) is a distributed, immutable collection of data items partitioned across nodes in a cluster [Spa24f]. RDDs support transformations and actions for parallel processing, ensuring fault tolerance and high performance.

2.1.7 Spark SQL: Relational Data Processing

Data Processing and Shuffling: During execution, Spark processes the data according to the optimised plan. If there are operations that require shuffling (data redistribution across partitions), Spark efficiently manages this process to minimise data movement and optimise performance. Spark SQL has two main components, as shown in Figure 2: Optimizer and Dataframe API.

Optimizer jobs: It has five stages, namely Parsing, Analysis, Optimization, Code Generation, and Execution [AXL+15].

- Parsing: Spark SQL parses the SQL queries to understand their syntactic structure.
- Analysis: The parsed queries undergo an analysis phase where Spark SQL checks for semantic errors, resolves references to tables and columns, and ensures the queries are logically sound.
- Optimization: Catalyst performs query optimization by transforming the logical execution plan into a physical execution plan. This includes optimizations like predicate pushdown, constant folding, and other rule-based transformations to enhance performance.
- Code Generation: Spark SQL uses code generation to generate Java bytecode for the physical execution plan. This compiled code is then executed on the Spark engine.
- Execution: The optimised and compiled code is executed on the Spark engine, processing the data in a distributed and parallelized manner across the Spark cluster.

DataFrame: This API provides a higher-level abstraction for working with structured data in Spark SQL [AXL$^+$15], [Spa24c].
It allows users to express data manipulation operations in a more concise and declarative manner than traditional RDD-based operations. DataFrames represent distributed collections of data organised into named columns, similar to tables in a relational database. The DataFrame API supports a wide range of operations, including filtering, grouping, joining, aggregating, and windowing, making it suitable for various data processing tasks. DataFrames seamlessly integrate with Spark SQL, allowing users to execute SQL queries directly on DataFrame objects and vice versa.

2.1.8 Spark SQL Key Benefits

Spark SQL supports structured and semi-structured data, and this allows users the flexibility to work with different data sources (such as Parquet, CSV, TBL, and Avro). In addition, Spark SQL extends its capabilities to support Structured Streaming, allowing users to process real-time data using the same high-level SQL constructs as batch processing. This simplifies the development of streaming applications, and Spark SQL inherits the scalability and performance benefits of the underlying Spark engine. It can efficiently process large-scale datasets in a distributed and parallelized fashion, making it suitable for big data analytics [Spa24g].

2.2 Storage Mechanisms

This section delves into various file formats commonly utilised in the industry alongside their underlying technologies, and these file formats are of two different types.

2.2.1 Storage models

In the literature, mainly we have two alternatives for storing formats, namely column and row-oriented [Wri24], as shown in Figure 3.

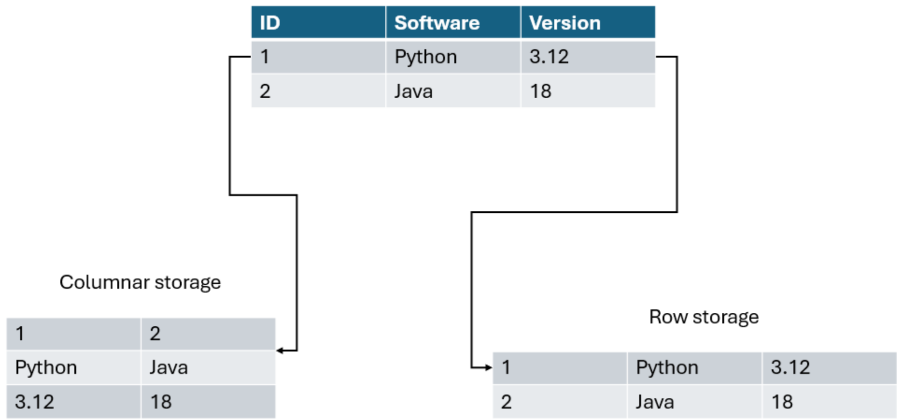

Figure 3. Columnar and row based storage

2.2.2 File Formats

In terms of file format technologies, Apache Parquet, Apache Avro, CSV, and TBL are used.

- **Apache Parquet:** Apache Parquet is a columnar storage file format designed for the Apache Hadoop ecosystem. It is optimised for efficient storage and processing of large-scale structured data. Parquet is particularly well-suited for analytical workloads in big data environments, and it supports various comparison methods; the most popular one is Gzip [Voh16b], [Com22b].

- **Apache Avro:** Apache Avro is a data serialisation system that efficiently exchanges data between systems and languages. It is a row-based format that focuses on providing a compact and efficient binary serialisation format. Avro is particularly well-suited for use cases requiring data schema evolution, efficient serialisation, and interoperability across different programming languages and systems [Voh16a], [Com22a].

- **CSV:** Comma-separated values (CSV) is a widely used file format for storing tabular data in plain text. It is a simple, lightweight format that is easy to understand and widely supported by various software applications and programming languages [Sha05].

- **TBL:** Tabular Data Package (TBL) is a file format and data packaging specification that aims to provide a standardised way to package and share tabular data and metadata. TBL files typically contain structured data organised in rows and columns, similar to CSV files, but with additional metadata to describe the data schema and other attributes [Com23].

2.3 Computer Energy Consumption Formulations

Energy, a fundamental concept in physics, manifests in various forms, such as electrical, magnetic, chemical, and nuclear. In this study, we focus on electrical energy, measured in Joules [DBOR20]. **Power**, denoted as P, represents the rate of energy transfer per unit time measured in Watts. It can be expressed as the work done, W, over a period, t, as per Equation 1.

$$P = \frac{W}{T} \tag{1}$$

Energy, denoted as E, is the total electrical energy consumed over time, calculated by multiplying power by time, as shown in Equation 2.

$$E = P \times T \tag{2}$$

In information technology, work entails activities associated with program execution. At the same time, power denotes the rate of electrical energy consumption per second, and energy represents the total electrical energy consumed over time.

Electrical power consumption can be categorised into *dynamic power* and *static power*.

- Static power, or leakage power ($P_{leakage}$), stems from leakage current ($I_{leakage}$) flowing in the system's idle state. This includes consumption from components like fans, processors, memory, and I/O devices when inactive, as expressed in Equation 3.

$$P_{leakage} = I_{leakage} \times Voltages \tag{3}$$

- Dynamic power consumption ($P_{dynamic}$) occurs during workload execution and is influenced by the type of workload and how it utilises the processor, memory, and I/O devices. Switched capacitance primarily drives dynamic power consumption, as detailed in Equation 4.

$$P_{dynamic} = \alpha \times c \times f \times V^2 \tag{4}$$

where Where α is the percentage of active gates, c is the capacitance, V is the voltage, and f is the frequency.

- Energy efficiency (EE) signifies optimal energy utilisation to provide the same service. It is defined as the ratio of performance to power, expressed as in Equation 5:

$$EE = \frac{useful\ energy}{total\ energy} \tag{5}$$

2.4 The motivation of the thesis

Data warehouses are critical infrastructure for storing and managing vast data, facilitating efficient data analysis and decision-making processes. However, the increasing energy needs of storage and data processing systems exacerbate environmental problems and significantly increase operational costs. In order to address these ecological issues while meeting the growing demand for data management services, initiatives need to be developed to facilitate the digital transition towards greater energy efficiency. This thesis builds upon previous research efforts to optimise the energy consumption of storage and data processing systems through efficient and appropriate utilisation of available resources. By targeting energy reduction, this thesis aims to mitigate the environmental impact of data warehouse operations while concurrently reducing operational expenses. Achieving this goal involves implementing energy-efficient hardware and software optimization techniques and adopting sustainable data management practices. In this thesis, we examine the energy consumption of the query processing system in the SparkSQL engine across various configurations. We begin by evaluating the impact of storage format choices on the system's total energy consumption. Then, we analyse the effect of the number of data partitions while query execution. Finally, we assess the impact of Scheduling policy by switching between FIFO to FAIR. These various assessments aim to guide storage or processing system administrators towards best practices that would optimise both the energy consumption of the system and the query execution performance. In summary, this thesis endeavours to optimise data warehouse operations by addressing the interconnected challenges of energy consumption, performance enhancement, and resource optimization. By implementing innovative strategies and technologies, it seeks to create a more sustainable, efficient, and resilient data infrastructure capable of meeting the demands of modern data-driven enterprises.

3 Literature review

In this section, after discussing the approaches used to evaluate the energy consumption of computer systems, we will cover the recent works related to energy efficiency techniques in the Apache Spark data processing engine.

3.1 Energy Evaluation

Efficient energy assessment relies on the collaboration between advanced computational frameworks and practical physical instruments. While energy assessment models offer thorough analyses and optimization plans, physical tools provide actual data collection and validation. The integration of these methods enables a comprehensive understanding of energy usage patterns and facilitates specific efficiency enhancements.

- **Energy Assessment Models:** These computational frameworks employ mathematical algorithms and simulation methods to forecast energy consumption and pinpoint optimization opportunities. These models deliver valuable insights into energy consumption trends by considering factors like equipment efficiency, operational parameters, and environmental factors.

- **Physical Tools:** Tools like smart meters, energy audit equipment, and data logging devices enable real-time data collection and analysis. They empower energy auditors and researchers to gather empirical data on energy usage, identify inefficiencies, and validate the conclusions drawn from computational models.

- **Combining Models and Tools:** Merging energy assessment models with physical tools heightens the accuracy and dependability of energy evaluations. This collaboration allows stakeholders to devise targeted energy-saving strategies based on empirical data and computational insights.

3.2 Taxonomy of Energy Efficiency (EE) in Apache Spark

An overload of work has been conducted to improve energy efficiency in data stores and processing engines. These works fall into two approaches: an approach based on hardware tuning and an approach based on software optimization. Many works in the Apache spark engine ecosystem have focused on resource management, job scheduling, processing tuning, and dynamic power management techniques to optimise the overall energy system consumption, as shown in Figure 4.

- Resource Management: Efficient cluster resource allocation and dynamic resource adjustment.

- Job Scheduling: Optimising task and stage scheduling to minimise idle time and resource contention.

- Data Processing Techniques: Utilising data partitioning, compression, and caching to reduce network and I/O energy consumption.

- Algorithmic Optimization: Choosing efficient algorithms and caching to minimise CPU and I/O operations.

- Hardware Considerations: Optimising CPU and memory utilisation for energy efficiency.
- Monitoring and Optimization: Continuous monitoring, profiling, and tuning to identify inefficiencies.
- Environment-aware Execution: Considering temperature, cooling, and power constraints.
- Power Management: Utilising DVFS and power-aware scheduling to optimise energy consumption.

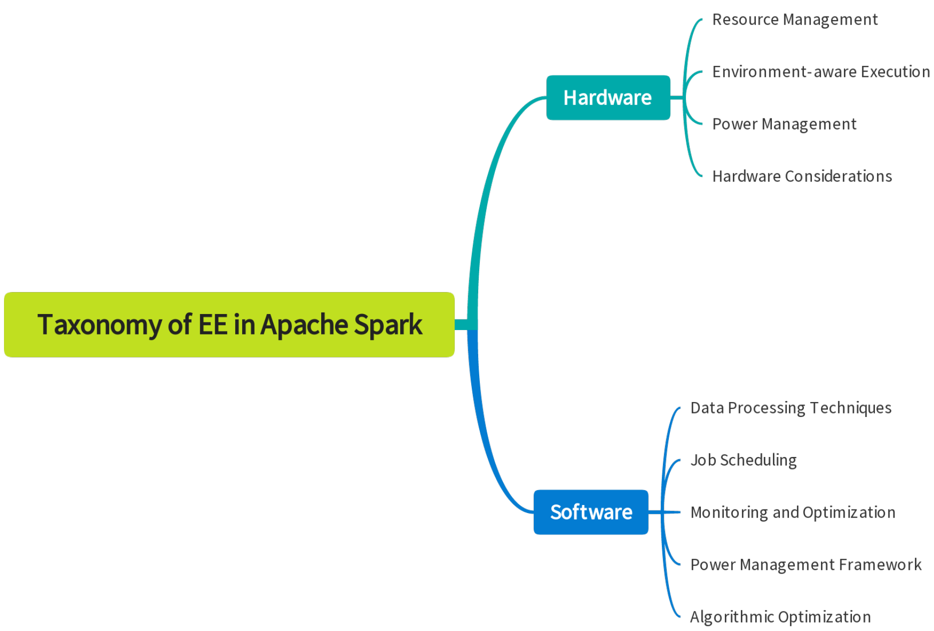

Figure 4. Taxonomy of energy efficiency in Apache Spark

3.2.1 Hardware approaches solution

Authors in [SSK16] explore how small, energy-efficient computer chips called System-on-Chip (SoC) platforms perform when used for Apache Spark data analysis. Even though these chips take longer to finish tasks than powerful processors, they use much less energy. These chips can be up to 3 times more energy-efficient for tasks like machine learning and graph computations. This means they can help save power and reduce costs in data centres. So, even though they are slower, they are better for saving energy, especially if energy efficiency is more important than speed.

In [HZK$^+$18], authors introduce a method for combining FPGA (Field-Programmable Gate Array) accelerators with Apache Spark clusters to improve performance and energy efficiency in big data processing. The researchers used a 2D-FFT (Two-Dimensional Fast Fourier Transform) algorithm as a case study to test the FPGA-based Spark framework. The results showed that the FPGA-based Spark implementation was 1.79 times faster than the CPU implementation. The authors aim to further enhance the performance of Spark clusters by equipping each slave node with an FPGA accelerator and optimising the implementation of other resource-intensive algorithms. This approach could significantly improve the efficiency of big data processing in various applications and industries.

Researchers in [KSKS18] demonstrate a new approach for easily using FPGA hardware in data centres with Spark as part of the VINEYARD project. This method lets FPGAs be used efficiently without changing application code, leading to faster speeds and less energy use in machine learning tasks. The researchers tested the KMeans machine learning algorithm. They found that the FPGA setup was twice as fast, used 23 times less energy than an Intel Xeon processor, and was 31 times faster and 29 times more energy-efficient than an ARM-only solution. The VINEYARD project aims to create a system for energy-efficient data centres using programmable hardware accelerators, promoting innovation in FPGA-based solutions for cloud computing.

In [Ngh18], authors focus on addressing the pressing issue of high energy consumption by data processing engines in data centres, exacerbated by the exponential growth in Big Data processing. The paper proposes the innovative Best Trade-off Point (BToP) method to tackle this challenge. This method offers a systematic approach, leveraging mathematical algorithms to identify the optimal trade-off point on an elbow curve, balancing performance against resources for efficient resource allocation in Hadoop MapReduce environments. Extending the applicability of the BToP method beyond Hadoop MapReduce, the paper applies it to the emerging cluster computing framework, Apache Spark. By utilising the BToP method, the study demonstrates improved performance and energy consumption compared to Spark's built-in dynamic resource allocation mechanism. Spark-bench tests confirm the effectiveness of employing the BToP method to determine the optimal number of executors for various workloads in production environments. The BToP method is distinguished by its ability to precisely identify the optimal number of executor resources for a workload, striking the ideal balance between performance and computing resources based on a runtime elbow curve derived from sampled executions within the target cluster. The versatility of the BToP techniques extends beyond cluster-computing frameworks like Hadoop and Spark, offering potential applications across diverse systems and applications relying on trade-off curves for informed decision-making. While this paper explicitly evaluates Apache Spark running on YARN for resource provisioning efficiency, the BToP method holds promise for optimising resource allocation in various computing environments.

Authors in [LWX+21] address the pressing issue of high energy consumption in Cloud data centres for big data processing, proposing a frequency-aware and energy-saving strategy termed FAESS-DVFS for Spark on YARN. This strategy aims to optimise energy consumption while maintaining Service Level Agreements (SLA) by implementing energy-saving measures at both the YARN and Spark layers. In the YARN layer, an optimal CPU frequency is determined based on the minimum energy efficiency ratio (EER) obtained from a status monitoring module. This frequency is selected to minimise energy consumption while ensuring SLA requirements. In the Spark layer, a task scheduling method is developed to dynamically adjust the CPU frequency of nodes throughout the lifecycle of different stages. This adjustment optimises energy consumption by leveraging dynamic voltage and frequency scaling (DVFS) in response to varying workload demands. Experimental tests conducted using Hibench demonstrate that the FAESS-DVFS method achieves significant energy savings of up to 29.5% compared to default algorithms in Spark on YARN, all while satisfying SLA constraints.

3.2.2 Software solution

The work in [LWF+20] presents an energy-aware scheduling algorithm, EASAS, designed to mitigate the escalating energy consumption associated with the rapid growth of big data applications in Apache Spark clusters. EASAS dynamically allocates tasks based on historical data, optimising energy usage while ensuring service level agreements (SLA) are met. Through comprehensive experimentation across various workloads from the HiBench suite, EASAS demonstrates remarkable energy savings, achieving reductions of up to 51.2% and 56.3% compared to traditional FIFO and FAIR scheduling strategies. These findings underscore EASAS's potential to significantly curb energy consumption in Spark clusters without compromising performance objectives. Future research will further optimise task scheduling to unlock even greater energy efficiencies.

[MZK17b] introduces a framework for efficient energy scheduling of Spark workloads, addressing the pressing need to minimise energy consumption in distributed processing systems while meeting performance requirements. The framework orchestrates the execution order of Spark applications, utilising dynamic voltage and frequency scaling (DVFS) to tune CPU frequencies and minimise energy usage. Experimental results demonstrate the framework's effectiveness in reducing energy consumption while satisfying application deadlines. The paper presents a novel framework designed to optimise energy usage in Spark workloads, which is crucial for reducing data warehouse operations' environmental impact and operational costs. The framework aims to balance energy efficiency with performance requirements by dynamically adjusting CPU frequencies based on workload characteristics. Experimental evaluations showcase the framework's effectiveness in minimising energy consumption while meeting application deadlines.

This paper aligns closely with the broader research objective of optimising energy consumption in data storage and processing systems. While the research introduction focuses on addressing ecological concerns and operational costs through energy-efficient practices, the presented framework contributes directly by providing a solution tailored to the Spark ecosystem. Leveraging dynamic scheduling techniques complements efforts to enhance resource utilisation and reduce environmental impact in data warehouse operations.

[IKB17] introduces dSpark, a lightweight resource allocation framework tailored for Apache Spark, overcoming limitations in existing methods. Unlike conventional approaches, dSpark autonomously determines a cost-efficient resource allocation plan, considering individual user deadlines, thus eliminating manual input. Moreover, dSpark incorporates a predictive model for application completion times, utilising application profiles to forecast task completion durations precisely. This enhances resource allocation efficiency, minimising both cost and resource consumption. The authors' findings demonstrate dSpark's efficacy in selecting optimal resource allocation strategies and enhancing performance across diverse user deadlines. Furthermore, dSpark simplifies deployment by removing the need for users to specify application types. Additionally, the accuracy of the framework depends on the depth of application profiling, with more thorough profiling yielding more precise predictions. Furthermore, while dSpark assumes homogeneous worker nodes, it can also adapt to heterogeneous environments.

The authors in [SLG$^+$22] introduce two scheduling algorithms, TPCBFD and EATPCBFD, to enhance energy efficiency and meet Service Level Agreement (SLA) requirements in Apache Spark. TPCBFD categorises tasks into three types and assigns them to nodes with superior performance, while EATPCBFD further optimises energy efficiency based on an energy consumption model. Experimental results demonstrate significant improvements in energy efficiency and SLA adherence compared to existing algorithms.

The authors in [MZK17a] introduce ExpREsS, a scheduling system designed for distributed processing frameworks like Apache Spark, with a focus on minimising energy consumption while meeting application performance requirements. ExpREsS utilises time-series prediction models to understand application energy usage and execution times, enabling it to apply dynamic voltage and frequency scaling (DVFS) techniques to reduce energy consumption effectively. Experimental results illustrate the benefits of ExpREsS in optimising energy usage while meeting application deadlines, outperforming existing scheduling approaches. Key contributions include formulating the problem of energy-efficient scheduling, proposing the ExpREsS scheduler, and providing methods for detecting and exploiting periodic power usage patterns. In summary, ExpREsS enhances the efficiency of mixed workloads in distributed processing systems, offering practical solutions for minimising energy utilisation and meeting performance goals.

3.3 Conclusion

The review of previous works regarding energy efficiency in Apache Spark show that various methods have been explored to target energy efficiency issues in Spark clusters. To facilitate the understanding, we have classified these works into two groups. The first, named hardware approaches, are solutions that leverage the redesigning and tuning of hardware, and the second, named software approaches, focus on managing resources, scheduling tasks, and optimising algorithms. Our proposition falls in the middle of these approaches; we aim to confront different configurations by tuning factors such as file formats, partition size, and memory size to determine and make recommendations for the one that fits with the green purpose. These recommendations will provide valuable insights for practitioners aiming to improve energy efficiency in their data processing workflows.

4 Methods

This chapter summarises the softwares and tools, data modelling, and queries used in the research thesis. It explains how the study was conducted, including the methods and materials used. Also, it emphasises the importance of these components in reaching the research goals.

4.1 Softwares and Tools

This study conducted energy measurements using the Yocto-Watt device, Python, Scala, and SQL were used as the programming and query languages, while Apache Spark 3.5 was utilised for data processing and TPC-H benchmark (Implementations Repository).

4.1.1 Yocto-Watt

This tool is a digital watt-meter designed to monitor the power usage of electrical devices, as shown in Figure 5. It functions with both AC and DC currents. It calculates the actual power consumption for AC currents, making it suitable for monitoring inductive loads.

Additionally, it can measure power consumption over a specified period with an accuracy of 1mWh, 1%. Moreover, the device is isolated, ensuring that the sensor component is electrically separated from the USB component, allowing voltage differences to be measured within the range of -250V to 250V. [YW23].

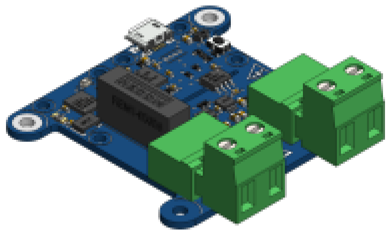

Figure 5. Yocto-Watt device

4.1.2 TPC-H Benchmark Datasets

The TPC-H benchmark is used for decision support and includes different business-related queries and data modifications. It is designed to be relevant across different industries and demonstrates systems that analyse large datasets and answer complex business questions. The benchmark measures performance using the TPC-H Composite Query-per-Hour Performance Metric (QphH-Size), which considers database size, query processing power, and throughput for single and multiple users [Ben23].

- **Data Modeling**
  The TPC-H benchmark employs a star schema data model with a central fact table encircled by several dimension tables, as shown in Figure 6. This model is prevalent in decision support systems and data warehousing applications, facilitating streamlined querying and analysis of extensive datasets [Ben23].

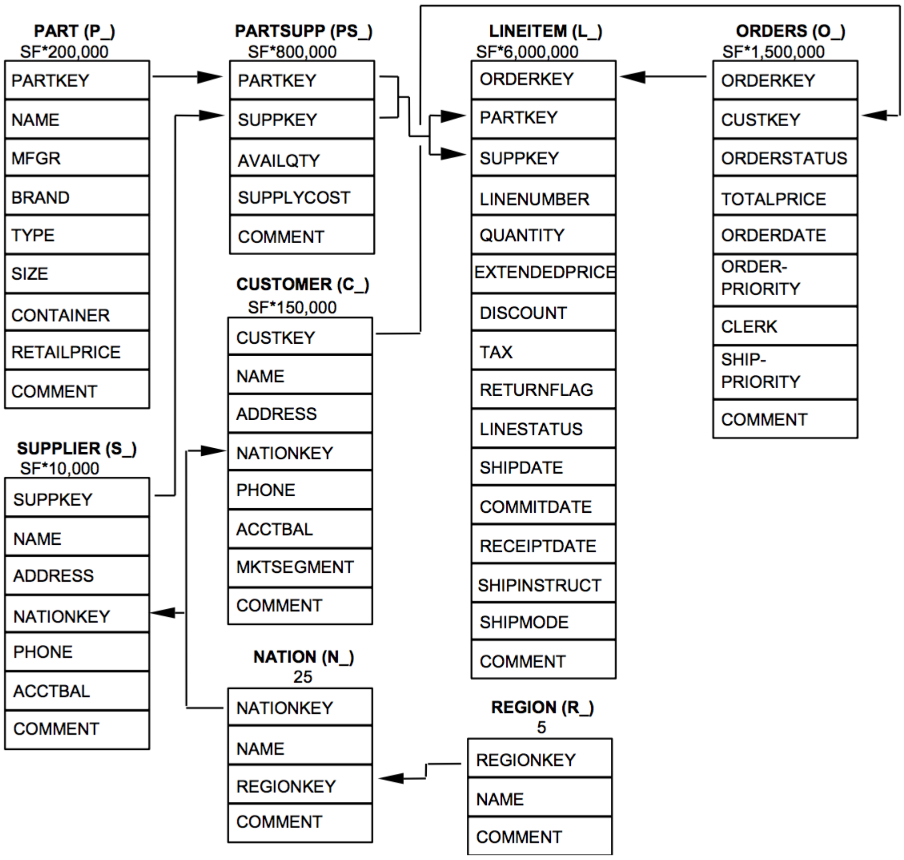

Figure 6. TPC-H Star Schema Data Model

- **TPC-H Queries**
  The TPC-H benchmark employs complex decision support queries to simulate real-world business situations, assessing database systems' capacity to handle large data volumes and perform advanced tasks like aggregations, joins, and filtering, which is prevalent in data warehousing and business intelligence contexts, and there are 22 queries (See Appendix I).

4.2 Environment

In Figure 7 of the thesis, the illustration depicts a personal computer connected to a lab computer via SSH, with a Yocto-Watt device connected to the lab computer for real-time energy measurement. SSH connection to the lab computer is chosen to avoid unnecessary energy consumption when running Ubuntu's graphical user interface (UI) and additional services. In addition, lab computer specification shown in Table 1.

Systems	Description
Operation System	Linux Ubuntu 64 bit 22.04.2 LTS
CPU	Intel Core i7-6700 CPU 3.40 GHz - 4 cores, 8 threads
RAM	16 gigabyte DDR4 2133MHz
Datasize	43 gigabyte
Thermal design power	65 W
Development tools	Python, Scala, Apache Spark, Pandas, Python Threading
Apache Spark Config	Standalone mode, one master, 10 gigabytes executor memory, one executor with 8 threads

Table 1. OS and Computer Specification

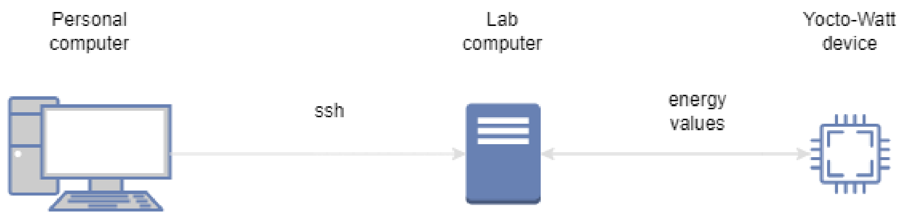

Figure 7. Energy Values by Query and File Format.

4.3 Power and performance measurement process

In the thesis, the process involves concurrently initiating both a Spark job and the Yocto-Watt device, achieved through utilising the multiprocessing library in Python. This approach enables the execution of both tasks simultaneously. Upon completion of the Spark job, the Yocto-Watt stops energy measurement. Furthermore, the Python script allows for specifying parameters such as partition size, file format, and query number. These parameters facilitate the testing of individual queries or varying partition sizes. Furthermore, the Yocto-Watt device records energy measurements at one-second intervals and logs them to an Excel sheet. However, due to this setup, the energy values captured may be near real-time. Additionally, a finish message is sent to the queue upon completion of the Spark job processing. Subsequently, the Yocto-Watt process receives this message and concludes the energy measurement. Consequently, the energy values corresponding to one or two rows at the end of the Excel sheet may not be directly linked to the energy measurement of the Spark job.

5 Results

This section extensively examines the performance and energy aspects of various file formats, presenting results obtained for different partition sizes. It delves into the impact of file formats on performance metrics and energy consumption. Additionally, it provides detailed insights into the outcomes observed when varying partition sizes, shedding light on how these alterations affect the system's overall performance and energy efficiency.

5.1 Experiment 1 - File Formats confrontation

In this experimentation, we kept the default partition number at 200 (default size 128 megabytes) in the default configuration of Apache Spark while utilizing a single executor with 8 threads and 10 gigabytes of memory. As depicted in Figures 8 and 9, Apache Parquet demonstrates lower energy consumption during read and data processing operations. In generating random data, there was not an Apache Parquet file. Instead, we converted the data into three widely used file formats: Apache Parquet, Apache Avro, and CSV. Following this conversion, the Apache Parquet file was observed to be smaller than the others due to its effective compression technique. Despite the fact that Parquet requires a data decompression step when processing queries, it nevertheless remains the most efficient in terms of time and energy consumption.

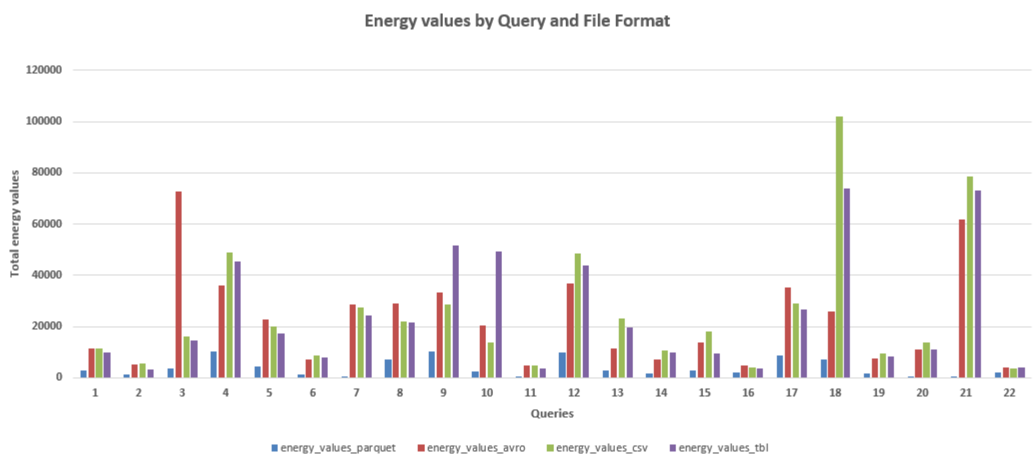

Figure 8. Energy Values by Query and File Format.

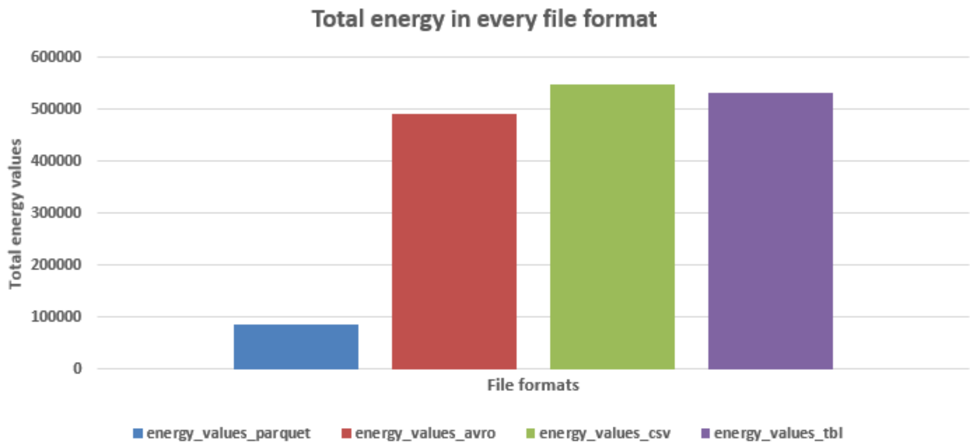

Figure 9. Total Energy Values in every File Format

5.2 Experiment 2 - Different Partition Numbers

This analysis explores the effects of different partition sizes on time-performance and energy consumption. In this experiment, we selected partition numbers of 16, 78, 300, 400, 500, and 600, utilizing best file format the Apache Parquet from the previous experimentations. And, we employed a single executor with 8 threads and 10 gigabytes of memory. We try to asses the best number of partitions using the formula mentioned below (Formula 6) which use executor memory efficiently. From our formula, we set executor memory to 10 gigabytes and using default the partition size (128 megabytes), as a result, we found 78 partition number. Therefore, we perform experimentation with this number and others partition numbers choosed randomly to evaluate Apache Spark's performance and energy consumption. Various scenarios involving partition numbers below and above 78 were included based on random selection.

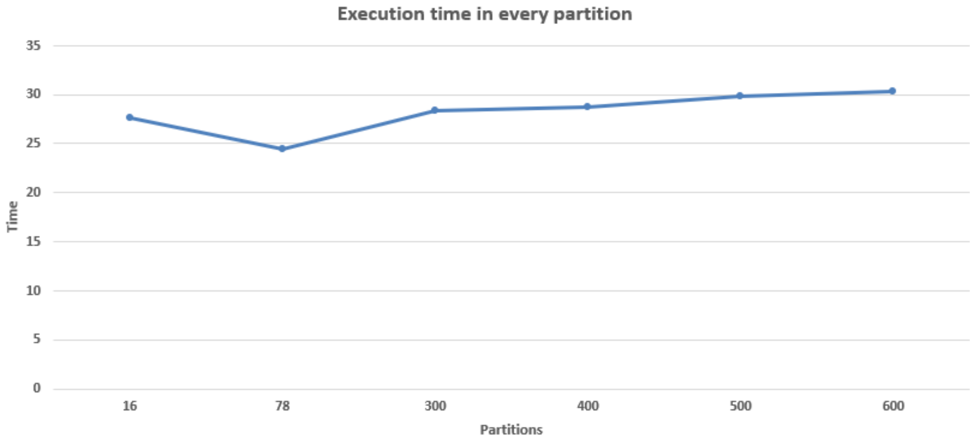

Figure 10. Partitions and Time Performance

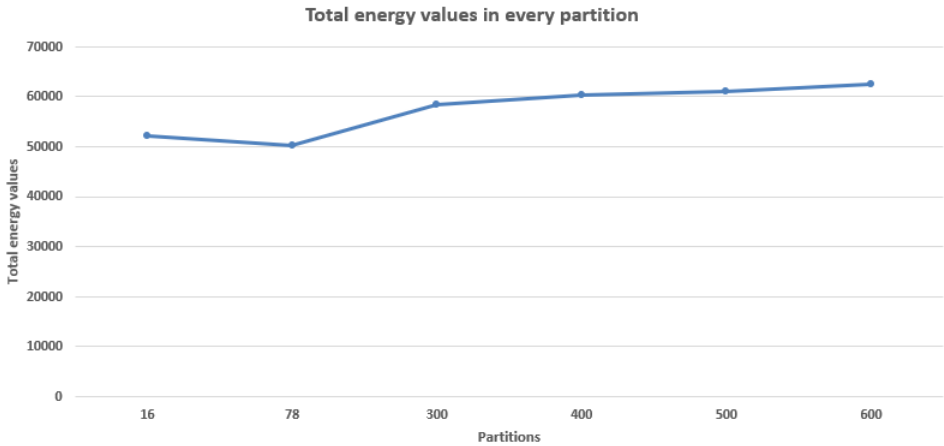

Figure 11. Partitions and Energy

And, we can see results for 78 partitions for energy and time in all queries, as shown in Figure 10 and 11.

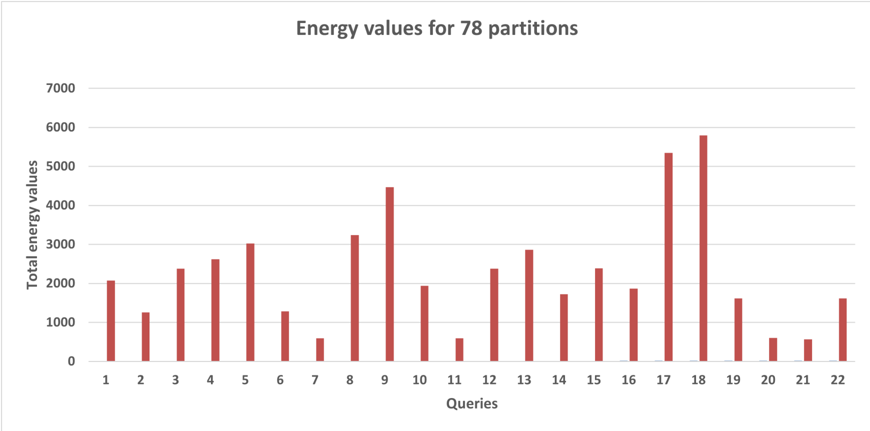

Figure 12. Energy values in all queries for partition number 78.

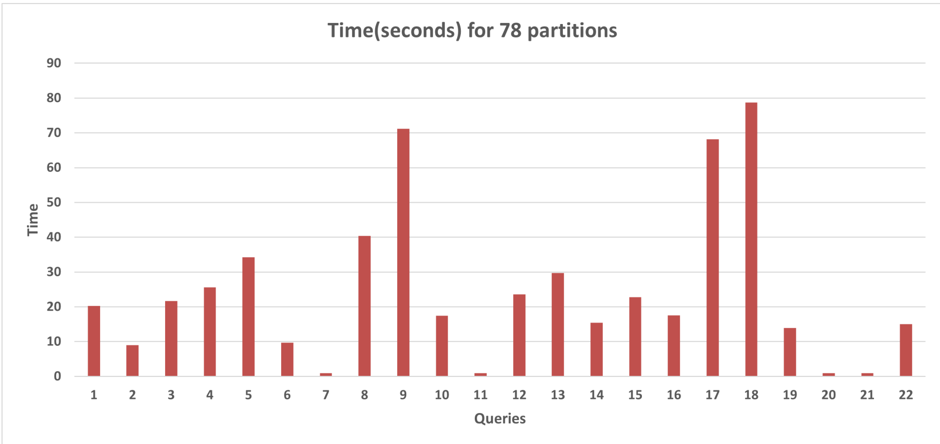

Figure 13. Time in all queries for partition number 78

Figure 10 and 11 illustrates that a partition size of 78 outperforms other partition sizes. The formula used for this particular partition size is provided below and Figure 12 and 13 show execution time and energy values in single queries.

$$\frac{ExecutorMemorySize(mb)}{PartitionSize(mb)} = NumberofPartitions \tag{6}$$

The default partition size in Apache Spark is 128 megabytes and 200 partitions, corresponding to an HDFS block. Given our utilisation of a 10-gigabyte executor memory, the formula induces a result of 78 partitions.

5.3 Experiment 3 - Different Partition Size

In this section, we extended the experiment by introducing new parameters. We opted for 300 partitions with a partition size of 33 megabytes to observe the impact on performance and we will compare with 300 partitions with 128 megabytes size in the 18th query which consumed much more time and energy than other queries (the plan's execution is on Execution plan. Figures 14 and 15 show that utilising 300 partitions with a size of 33 megabytes is not as efficient as using 300 partitions with a size of 128 megabytes in terms of both time and energy consumption. This can be attributed to the small partition size, which introduces additional overhead in scheduling multiple tasks and managing more extensive metadata. Similar to multithreaded applications, increasing parallelism does not necessarily result in improved performance. Additionally, opting for larger partition sizes may result in reduced concurrency and heightened memory pressure during transformations involving shuffling.

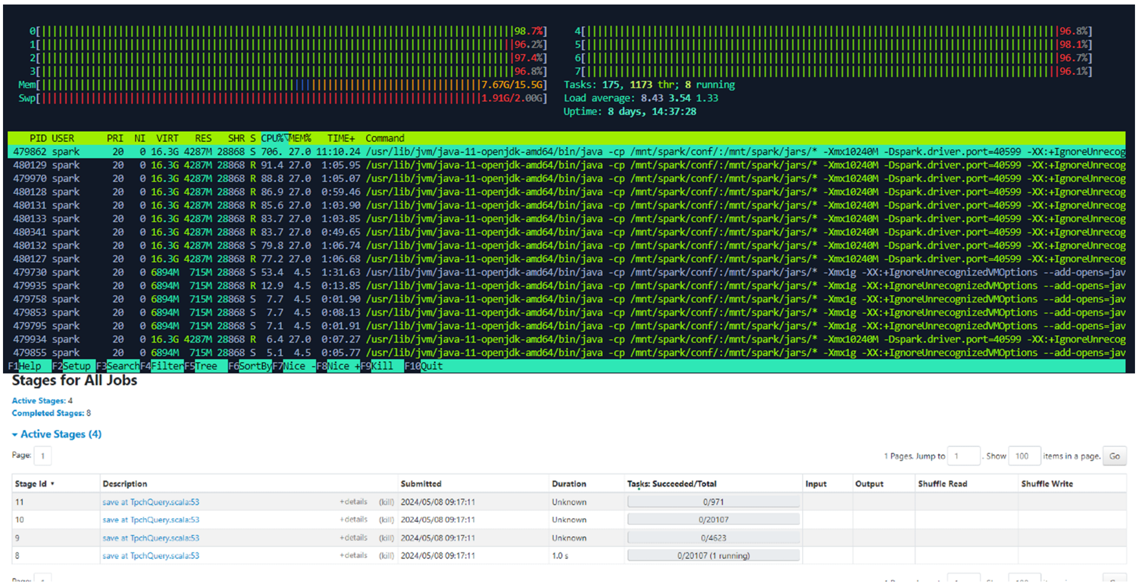

Figure 14. Partition 300 with 33 megabytes

Figure 14 illustrates that memory was not utilised efficiently, leading to Spark creating an excessive number of tasks, exceeding 21,000. This inefficiency can be attributed to the small partition size chosen.

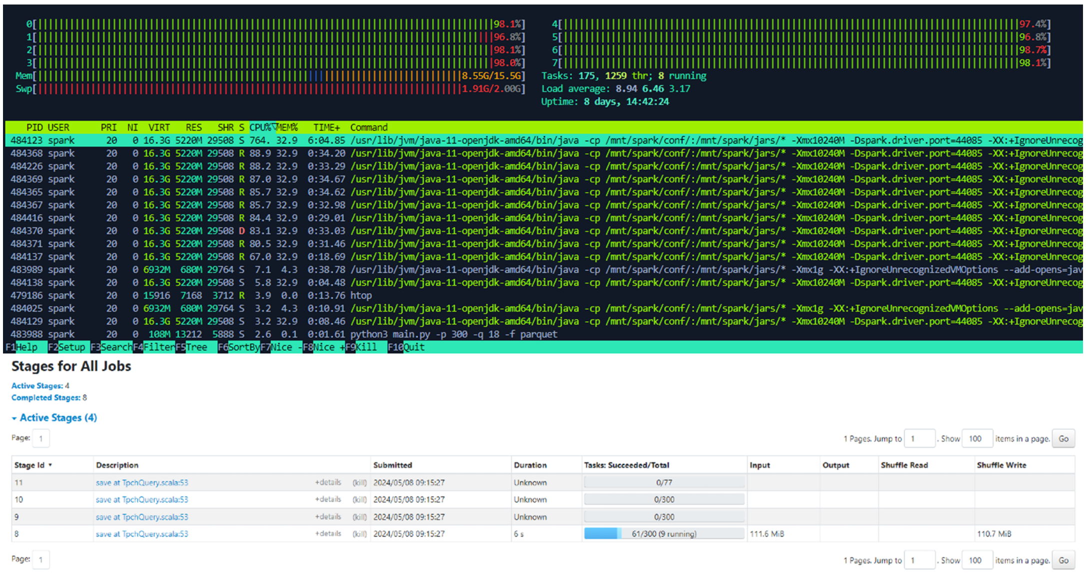

Figure 15. Partition 300 with 128 megabytes

In contrary, with a partition size of 128 megabytes, we observe that memory utilisation improves, with over 300 tasks being created for each job, as shown in Figure 15.

5.4 Experiment 4 - Scheduling policy FIFO to FAIR

- FIFO (First In, First Out): FIFO is a basic scheduling policy where tasks are executed in the order they were submitted to the cluster. In other words, the first task submitted is the first one to be executed, and so on. While FIFO scheduling is simple, it may only sometimes be the most efficient, especially in multi-tenant environments where different users or applications may have varying priorities.

- FAIR: FAIR scheduling is a more sophisticated approach that aims to provide better resource allocation and fairness among multiple applications or users sharing the same cluster. With FAIR scheduling, resources are divided into pools, and each pool is allocated a particular share of the cluster resources. Within each pool, tasks are scheduled using the FIFO policy. This ensures that each pool receives a fair share of the resources, regardless of the workload or number of tasks submitted [Spa23].

In the experiment, 78 partitions were used with 128 megabytes which was the best one among other partitions after changing scheduling policy FIFO to FAIR. After experiment, 57.24 seconds and total 4954.58 energy consumption in FAIR scheduler, in FIFO, it was 78.75 second and nearly total 5794.14, as shown in Figure 16.

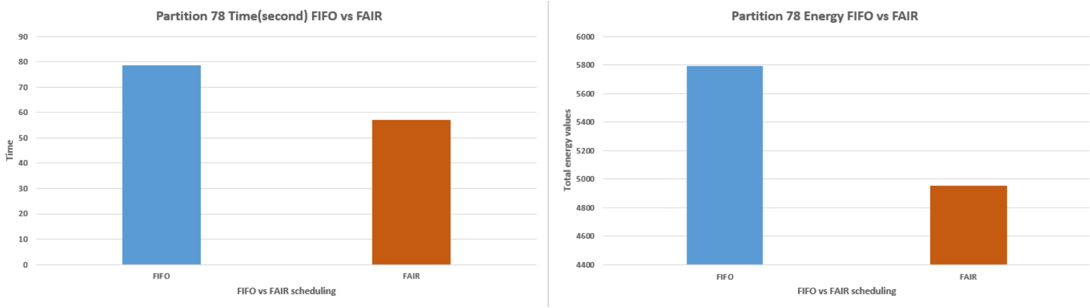

Figure 16. FIFO vs FAIR scheduling with 78 partitions

6 Discussion and recommendation

After conducting various experiments under different scenarios, it becomes evident that Apache Parquet exhibits lower energy consumption during reading and data processing tasks. This advantage can be attributed to the smaller size of Apache Parquet files, resulting from its efficient compression technique and columnar-oriented structure. These features contribute to enhanced energy efficiency and quicker data reading capabilities.

Furthermore, analysing different partition sizes reveals diverse impacts on time performance and energy consumption. Notably, employing a specific formula yields more optimal partition numbers, as observed in Experimentation 2. Additionally, when considering a reduction in partition size from 128 megabytes to 33 megabytes and we used Formula 6 above divinding exector memory by 300 partition numbers, it equals to 33 meagabytes, significant effects on performance and energy consumption emerge, primarily due to the proliferation of multiple tasks.

Lastly, exploring scheduling algorithms underscores that the FAIR algorithm is much better regarding performance and energy consumption due to FAIR scheduler mode is a good way to optimize the execution time of multiple jobs inside one Apache Spark program. Unlike FIFO mode, it shares the resources between tasks and therefore, do not penalize short jobs by the resources lock caused by the long-running jobs. This finding emphasises the importance of scheduling policies in optimising resource utilisation and system efficiency. These insights highlight the multifaceted nature of optimising performance and energy consumption in Apache Spark, underscoring the need to consider various factors in system configuration and resource allocation carefully.

7 Perspectives

In perspective, this study provides valuable insights into the performance and energy efficiency of Apache Spark in distributed data processing environments. Moving forward, several routes for future research and development emerge:

- Integration with Kubernetes and scaling the Spark cluster to evaluate performance and energy in containerized environments.
- Exploration of other alternatives of data storage solutions, such as columnar and row-based databases, for optimised data retrieval and processing.
- Development or utilisation of machine learning models for predictive analytics, leveraging historical query data to anticipate energy consumption and performance outcomes in the aim to select the plan that is more energy-optimised in the Catalyst optimizer.
- Investigation of other workload optimization techniques and resource provisioning strategies to enhance Spark's efficiency in diverse computational workflows.

8 Conclusions

In recent years, energy efficiency has become one of the major design requirements of computer system components, ranging from a simple laptop to cloud environment. The thing that had fostered this is the over energy consumption of components have practically not stopped to grow. In addition to exorbitant operating costs, high energy consumption leads to significant emissions of greenhouse gases into the environment responsible for climate change. In this thesis, we focused on evaluating the energy consumption of the Spark data processing system by comparing the different data storage files under different scenarios. Our thesis aimed to identify optimal configurations and data file formats that minimise energy consumption while enhancing performance in distributed data processing environments. Through comprehensive experimentation and analysis, insights were gained into the efficiency of Apache Parquet in reducing energy consumption, the significance of selecting optimal partition sizes for improved performance, and the impact of scheduling algorithms on resource utilisation. These findings contribute to advancing the understanding of energy-efficient and high-performance data processing in Apache Spark, paving the way for future research and development in this field.

References

[AXL+15] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1383–1394, 2015.

[Ben23] TPC-H Benchmark. Tpc-h benchmark, 2023. May, 2024.

[Com22a] Community. Apache avro, 2022. May, 2024.

[Com22b] Community. Apache parquet, 2022. May, 2024.

[Com23] Community. Tabular data package, 2023. May, 2024.

[DBOR20] Simon Pierre Dembele, Ladjel Bellatreche, Carlos Ordonez, and Amine Roukh. Think big, start small: a good initiative to design green query optimizers. *Cluster Computing*, 23:2323–2345, 2020.

[HZK+18] Junjie Hou, Yongxin Zhu, Linghe Kong, Zhe Wang, Sen Du, Shijin Song, and Tian Huang. A case study of accelerating apache spark with fpga. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 855–860. IEEE, 2018.

[IKB17] Muhammed Tawfiqul Islam, Shanika Karunasekera, and Rajkumar Buyya. dspark: Deadline-based resource allocation for big data applications in apache spark. In *2017 IEEE 13th International Conference on E-Science (e-Science)*, pages 89–98. IEEE, 2017.

[KSKS18] Christoforos Kachris, Ioannis Stamelos, Elias Koromilas, and Dimitrios Soudris. Seamless fpga deployment over spark in cloud computing: A use case on machine learning hardware acceleration. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings 14*, pages 673–684. Springer, 2018.

[LWF+20] Hongjian Li, Huochen Wang, Shuyong Fang, Yang Zou, and Wenhong Tian. An energy-aware scheduling algorithm for big data applications in spark. *Cluster Computing*, 23:593–609, 2020.

[LWX+21] Hongjian Li, Yaojun Wei, Yu Xiong, Enjie Ma, and Wenhong Tian. A frequency-aware and energy-saving strategy based on dvfs for spark. *The Journal of Supercomputing*, 77:11575–11596, 2021.

[MZK17a] Stathis Maroulis, Nikos Zacheilas, and Vana Kalogeraki. Express: Energy efficient scheduling of mixed stream and batch processing workloads. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 27–32. IEEE, 2017.

[MZK17b] Stathis Maroulis, Nikos Zacheilas, and Vana Kalogeraki. A framework for efficient energy scheduling of spark workloads. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 2614–2615. IEEE, 2017.

[Ngh18] Peter P Nghiem. Best trade-off point method for efficient resource provisioning in spark. *Algorithms*, 11(12):190, 2018.

[SDC+16] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016.

[Sha05] Yakov Shafranovich. Common format and mime type for comma-separated values (csv) files. Technical report, 2005.

[SLG+22] Wenhu Shi, Hongjian Li, Junzhe Guan, Hang Zeng, et al. Energy-efficient scheduling algorithms based on task clustering in heterogeneous spark clusters. *Parallel Computing*, 112:102947, 2022.

[Spa23] Apache Spark. Apache spark job scheduling, 2023. May, 2024.

[Spa24a] Apache Spark. Apache spark, 2024. May, 2024.

[Spa24b] Apache Spark. Apache spark core, 2024. May, 2024.

[Spa24c] Apache Spark. Apache spark dataframe api, 2024. May, 2024.

[Spa24d] Apache Spark. Apache spark graphx, 2024. May, 2024.

[Spa24e] Apache Spark. Apache spark mllib, 2024. May, 2024.

[Spa24f] Apache Spark. Apache spark rdd, 2024. May, 2024.

[Spa24g] Apache Spark. Apache spark sql, 2024. May, 2024.

[Spa24h] Apache Spark. Apache spark streaming, 2024. May, 2024.

[SSK16] Ioannis Stamelos, Dimitrios Soudris, and Christoforos Kachris. Performance and energy evaluation of spark applications on low-power socs. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 300–305. IEEE, 2016.

[Voh16a] Deepak Vohra. *Apache Avro*, pages 303–323. Apress, Berkeley, CA, 2016.

[Voh16b] Deepak Vohra. *Apache Parquet*, pages 325–335. Apress, Berkeley, CA, 2016.

[Wri24] Gavin Wright. Row and column-oriented storage, 2024. May, 2024.

[YW23] Yocto-Watt. Yocto-watt, 2023. May, 2024.

Appendix I

TPC-H Queries

```sql
-- Query 1.
SELECT l_returnflag,
       l_linestatus,
       SUM(l_quantity) AS sum_qty,
       SUM(l_extendedprice) AS
       sum_base_price,
       SUM(l_extendedprice * ( 1 - l_discount )) AS
       sum_disc_price,
       SUM(l_extendedprice * ( 1 - l_discount ) * ( 1 +
              l_tax )) AS sum_charge,
       Avg(l_quantity) AS avg_qty,
       Avg(l_extendedprice) AS avg_price,
       Avg(l_discount) AS avg_disc,
       Count(*) AS count_order
FROM   lineitem
WHERE  l_shipdate <= DATE '1998-12-01' - interval '[
      DELTA]' day (3)
GROUP  BY l_returnflag,
          l_linestatus
ORDER  BY l_returnflag,
          l_linestatus;

-- Query 2.
SELECT    s_acctbal,
          s_name,
          n_name,
          p_partkey,
          p_mfgr,
          s_address,
          s_phone,
          s_comment
FROM      part,
          supplier,
          partsupp,
          nation,
          region
WHERE     p_partkey = ps_partkey
AND       s_suppkey = ps_suppkey
AND       p_size = [SIZE]
AND       p_type LIKE '%[TYPE]'
AND       s_nationkey = n_nationkey
AND       n_regionkey = r_regionkey
AND       r_name = '[REGION]'
AND       ps_supplycost =
          (
             SELECT min(ps_supplycost)
             from   partsupp,
                    supplier,
                    nation,
                    region
             WHERE  p_partkey = ps_partkey
             AND    s_suppkey = ps_suppkey
             AND    s_nationkey = n_nationkey
             AND    n_regionkey = r_regionkey
             AND    r_name = '[REGION]' )
ORDER BY s_acctbal DESC,
          n_name,
          s_name,
          p_partkey;

-- Query 3.
SELECT l_orderkey,
        SUM(l_extendedprice * ( 1 - l_discount )) AS
              revenue,
        o_orderdate,
        o_shippriority
FROM    customer,
        orders,
        lineitem
WHERE   c_mktsegment = '[SEGMENT]'
        AND c_custkey = o_custkey
        AND l_orderkey = o_orderkey
        AND o_orderdate < DATE '[DATE]'
        AND l_shipdate > DATE '[DATE]'
GROUP  BY l_orderkey,
          o_orderdate,
          o_shippriority
ORDER  BY revenue DESC,
          o_orderdate;

-- Query 4.
SELECT o_orderpriority,
       Count(*) AS order_count
FROM   orders
WHERE  o_orderdate >= DATE '[DATE]'
       AND o_orderdate < DATE '[DATE]' + interval '3'
             month
       AND EXISTS (SELECT *
                   FROM   lineitem
                   WHERE  l_orderkey = o_orderkey
                          AND l_commitdate <
                              l_receiptdate)
GROUP  BY o_orderpriority
ORDER  BY o_orderpriority;

-- Query 5.
SELECT n_name,
       SUM(l_extendedprice * ( 1 - l_discount )) AS
             revenue
FROM   customer,
       orders,
       lineitem,
       supplier,
       nation,
       region
WHERE  c_custkey = o_custkey
       AND l_orderkey = o_orderkey
       AND l_suppkey = s_suppkey
       AND c_nationkey = s_nationkey
       AND s_nationkey = n_nationkey
       AND n_regionkey = r_regionkey
       AND r_name = '[REGION]'
       AND o_orderdate >= DATE '[DATE]'
       AND o_orderdate < DATE '[DATE]' + interval '1'
             year
GROUP  BY n_name
ORDER  BY revenue DESC;

-- Query 6.
SELECT Sum(l_extendedprice*l_discount) AS revenue
FROM   lineitem
WHERE  l_shipdate >= date '[DATE]'
AND    l_shipdate <  date '[DATE]'   + interval '1'
      year
AND    l_discount BETWEEN [DISCOUNT] - 0.01 AND    [
      DISCOUNT] + 0.01
AND    l_quantity < [QUANTITY];

-- Query 7.
SELECT supp_nation,
       cust_nation,
       l_year,
       SUM(volume) AS revenue
FROM   (SELECT n1.n_name                              AS
      supp_nation,
               n2.n_name                              AS
                      cust_nation,
               Extract(year FROM l_shipdate)          AS
                      l_year,
               l_extendedprice * ( 1 - l_discount ) AS
                      volume
        FROM   supplier,
               lineitem,
               orders,
               customer,
               nation n1,
               nation n2
```

```sql
         WHERE  s_suppkey = l_suppkey
                AND o_orderkey = l_orderkey
                AND c_custkey = o_custkey
                AND s_nationkey = n1.n_nationkey
                AND c_nationkey = n2.n_nationkey
                AND ( ( n1.n_name = '[NATION1]'
                        AND n2.n_name = '[NATION2]' )
                      OR ( n1.n_name = '[NATION2]'
                           AND n2.n_name = '[NATION1]' ) )
                AND l_shipdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31')
        AS
        shipping
GROUP  BY supp_nation,
          cust_nation,
          l_year
ORDER  BY supp_nation,
          cust_nation,
          l_year;

-- Query 8.
SELECT o_year,
       SUM(CASE
             WHEN nation = '[NATION]' THEN volume
             ELSE 0
           END) / SUM(volume) AS mkt_share
FROM   (SELECT Extract(year FROM o_orderdate)       AS o_year,
                l_extendedprice * ( 1 - l_discount ) AS volume,
                n2.n_name                            AS nation
         FROM   part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
         WHERE  p_partkey = l_partkey
                AND s_suppkey = l_suppkey
                AND l_orderkey = o_orderkey
                AND o_custkey = c_custkey
                AND c_nationkey = n1.n_nationkey
                AND n1.n_regionkey = r_regionkey
                AND r_name = '[REGION]'
                AND s_nationkey = n2.n_nationkey
                AND o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31'
                AND p_type = '[TYPE]') AS all_nations
GROUP  BY o_year
ORDER  BY o_year;

-- Query 9.
SELECT nation,
       o_year,
       Sum(amount) AS sum_profit
FROM   (SELECT n_name
                AS
                       nation,
                Extract(year FROM o_orderdate)
                AS
                       o_year,
                l_extendedprice * ( 1 - l_discount ) - ps_supplycost * l_quantity
                AS
                       amount
         FROM   part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
         WHERE  s_suppkey = l_suppkey
                AND ps_suppkey = l_suppkey
                AND ps_partkey = l_partkey
                AND p_partkey = l_partkey
                AND o_orderkey = l_orderkey
                AND s_nationkey = n_nationkey
                AND p_name LIKE '%[COLOR]%') AS profit
GROUP  BY nation,
          o_year
ORDER  BY nation,
          o_year DESC;

-- Query 10.
SELECT c_custkey,
       c_name,
       SUM(l_extendedprice * ( 1 - l_discount )) AS revenue,
       c_acctbal,
       n_name,
       c_address,
       c_phone,
       c_comment
FROM   customer,
       orders,
       lineitem,
       nation
WHERE  c_custkey = o_custkey
       AND l_orderkey = o_orderkey
       AND o_orderdate >= DATE '[DATE]'
       AND o_orderdate < DATE '[DATE]' + interval '3' month
       AND l_returnflag = 'R'
       AND c_nationkey = n_nationkey
GROUP  BY c_custkey,
          c_name,
          c_acctbal,
          c_phone,
          n_name,
          c_address,
          c_comment
ORDER  BY revenue DESC;

-- Query 11.
SELECT ps_partkey,
      Sum(ps_supplycost * ps_availqty) AS value
FROM   partsupp,
      supplier,
      nation
WHERE  ps_suppkey = s_suppkey
      AND s_nationkey = n_nationkey
      AND n_name = '[NATION]'
GROUP  BY ps_partkey
HAVING
      Sum(ps_supplycost * ps_availqty) >
      (SELECT Sum(ps_supplycost * ps_availqty) * [fraction]
      FROM   partsupp,
             supplier,
             nation
      WHERE  ps_suppkey = s_suppkey
             AND s_nationkey = n_nationkey
             AND n_name = '[NATION]')
ORDER  BY value DESC;

-- Query 12.
SELECT l_shipmode,
       SUM(CASE
             WHEN o_orderpriority = '1-URGENT'
                   OR o_orderpriority = '2-HIGH' THEN 1
             ELSE 0
           END) AS high_line_count,
       SUM(CASE
             WHEN o_orderpriority <> '1-URGENT'
                  AND o_orderpriority <> '2-HIGH' THEN 1
             ELSE 0
           END) AS low_line_count
FROM   orders,
       lineitem
WHERE  o_orderkey = l_orderkey
       AND l_shipmode IN ( '[SHIPMODE1]', '[SHIPMODE2]' )
       AND l_commitdate < l_receiptdate
       AND l_shipdate < l_commitdate
       AND l_receiptdate >= DATE '[DATE]'
       AND l_receiptdate < DATE '[DATE]' + interval '1' year
GROUP  BY l_shipmode
ORDER  BY l_shipmode;
```

```sql
-- Query 13.
SELECT c_count,
       Count(*) AS custdist
FROM   (SELECT c_custkey,
               Count(o_orderkey)
        FROM   customer
               LEFT OUTER JOIN orders
                            ON c_custkey = o_custkey
                               AND o_comment NOT LIKE
                                   '%[word1]%[word2
                                   ]%'
        GROUP  BY c_custkey)AS c_orders (c_custkey,
              c_count)
GROUP  BY c_count
ORDER  BY custdist DESC,
          c_count DESC;

-- Query 14.
SELECT 100.00 * SUM(CASE
                      WHEN p_type LIKE 'PROMO%' THEN
                           l_extendedprice *
                                                      (

                      ELSE 0
                    END) / SUM(l_extendedprice * ( 1 -
                          l_discount )) AS
       promo_revenue
FROM   lineitem,
       part
WHERE  l_partkey = p_partkey
       AND l_shipdate >= DATE '[DATE]'
       AND l_shipdate < DATE '[DATE]' + interval '1'
            month;

-- Query 15.
CREATE VIEW revenue[STREAM_ID]
            (
                        supplier_no,
                        total_revenue
            )
            AS
SELECT   l_suppkey,
         sum(l_extendedprice * (1 - l_discount))
FROM     lineitem
WHERE    l_shipdate >= date '[DATE]'
AND      l_shipdate <  date '[DATE]' + interval '3'
       month
GROUP BY l_suppkey;SELECT   s_suppkey,
         s_name,
         s_address,
         s_phone,
         total_revenue
FROM     supplier,
         revenue[STREAM_ID]
WHERE    s_suppkey = supplier_no
AND      total_revenue =
         (
                SELECT Max(total_revenue)
                FROM   revenue[STREAM_ID] )
ORDER BY s_suppkey;DROP VIEW revenue[STREAM_ID];

-- Query 16.
SELECT p_brand,
       p_type,
       p_size,
       Count(DISTINCT ps_suppkey) AS supplier_cnt
FROM   partsupp,
       part
WHERE  p_partkey = ps_partkey
       AND p_brand <> '[BRAND]'
       AND p_type NOT LIKE '[TYPE]%'
       AND p_size IN ( [size1], [size2], [size3], [
            size4],
                      [size5], [size6], [size7], [
                           size8] )
```

```sql
       AND ps_suppkey NOT IN (SELECT s_suppkey
                              FROM   supplier
                              WHERE  s_comment LIKE '%
                                     Customer%
                                     Complaints%')
GROUP  BY p_brand,
          p_type,
          p_size
ORDER  BY supplier_cnt DESC,
          p_brand,
          p_type,
          p_size;

-- Query 17.
SELECT Sum(l_extendedprice) / 7.0 AS avg_yearly
FROM   lineitem,
       part
WHERE  p_partkey = l_partkey
       AND p_brand = '[BRAND]'
       AND p_container = '[CONTAINER]'
       AND l_quantity < (SELECT 0.2 * Avg(l_quantity)
                         FROM   lineitem
                         WHERE  l_partkey = p_partkey)
                               ;

-- Query 18.
SELECT c_name,
       c_custkey,
       o_orderkey,
       o_orderdate,
       o_totalprice,
       Sum(l_quantity)
FROM   customer,
       orders,
       lineitem
WHERE  o_orderkey IN (SELECT l_orderkey
                      FROM   lineitem
                      GROUP  BY l_orderkey
                      HAVING Sum(l_quantity) > [
                             quantity])
       AND c_custkey = o_custkey
       AND o_orderkey = l_orderkey
GROUP  BY c_name,
          c_custkey,
          o_orderkey,
          o_orderdate,
          o_totalprice
ORDER  BY o_totalprice DESC,
          o_orderdate;

-- Query 19.
SELECT Sum(l_extendedprice * (1 - l_discount) ) AS
      revenue
FROM   lineitem,
       part
WHERE  (
              p_partkey = l_partkey
       AND    p_brand = '[BRAND1]'
       AND    p_container IN ( 'sm case',
                               'sm box',
                               'sm pack',
                               'sm pkg')
       AND    l_quantity >= [QUANTITY1]
       AND    l_quantity <= [QUANTITY1] + 10
       AND    p_size BETWEEN 1 AND     5
       AND    l_shipmode IN ('air',
                             'air reg')
       AND    l_shipinstruct = 'deliver IN person' )
OR     (
              p_partkey = l_partkey
       AND    p_brand = '[BRAND2]'
       AND    p_container IN ('med bag',
                               'med box',
                               'med pkg',
                               'med pack')
       AND    l_quantity >= [QUANTITY2]
       AND    l_quantity <= [QUANTITY2] + 10
       AND    p_size BETWEEN 1 AND    10
       AND    l_shipmode IN ('air',
                             'air reg')
       AND    l_shipinstruct = 'deliver IN person' )
OR     (
              p_partkey = l_partkey
```

```
        AND     p_brand = '[BRAND3]'
        AND     p_container IN ( 'lg case',
                                 'lg box',
                                 'lg pack',
                                 'lg pkg')
        AND     l_quantity >= [QUANTITY3]
        AND     l_quantity <= [QUANTITY3] + 10
        AND     p_size BETWEEN 1 AND     15
        AND     l_shipmode IN ('air',
                              'air reg')
        AND     l_shipinstruct = 'deliver IN person' );

-- Query 20.
SELECT    s_name,
          s_address
FROM      supplier,
          nation
WHERE     s_suppkey IN
          (
                 SELECT ps_suppkey
                 FROM   partsupp
                 WHERE  ps_partkey IN
                        (
                               SELECT p_partkey
                               FROM   part
                               WHERE  p_name LIKE '[
                                      COLOR]%' )
                 AND    ps_availqty >
                        (
                               SELECT 0.5 * Sum(
                                      l_quantity)
                               FROM   lineitem
                               WHERE  l_partkey =
                                      ps_partkey
                               AND    l_suppkey =
                                      ps_suppkey
                               AND    l_shipdate >=
                                      date('[DATE]') and
                                       l_shipdate < date
                                      ('[DATE]') +
                                      interval '1' year
                                      ) )
AND       s_nationkey = n_nationkey
AND       n_name = '[NATION]'
ORDER BY s_name;

-- Query 21.
SELECT s_name,
        Count(*) AS numwait
FROM    supplier,
        lineitem l1,
        orders,
        nation
WHERE   s_suppkey = l1.l_suppkey
        AND o_orderkey = l1.l_orderkey
        AND o_orderstatus = 'F'
        AND l1.l_receiptdate > l1.l_commitdate
        AND EXISTS (SELECT *
                    FROM   lineitem l2
                    WHERE  l2.l_orderkey = l1.
                           l_orderkey
                           AND l2.l_suppkey <> l1.
                                  l_suppkey)
        AND NOT EXISTS (SELECT *
                        FROM   lineitem l3
                        WHERE  l3.l_orderkey = l1.
                               l_orderkey
                               AND l3.l_suppkey <> l1.
                                      l_suppkey
                               AND l3.l_receiptdate >
                                      l3.l_commitdate)
        AND s_nationkey = n_nationkey
        AND n_name = '[NATION]'
GROUP   BY s_name
ORDER   BY numwait DESC,
           s_name;

-- Query 22.
SELECT cntrycode,
       Count(*)       AS numcust,
       Sum(c_acctbal) AS totacctbal
FROM   (
              SELECT substring(c_phone from 1 FOR 2)
                     AS cntrycode,
                       c_acctbal
              FROM   customer
              WHERE  substring(c_phone FROM 1 FOR 2)
                     IN ('[I1]', '[I2]','[I3]','[I4]','
                     [I5]','[I6]','[I7]') and c_acctbal
                      > ( select avg(c_acctbal) from
                     customer where c_acctbal > 0.00
                     and substring (c_phone from 1 for
                     2) in ('[I1]','[I2]','[I3]','[I4]'
                     ,'[I5]','[I6]','[I7]') ) and not
                     exists ( select * from orders
                     where o_custkey = c_custkey ) ) as
                      custsale group by cntrycode order
                      by cntrycode;
```

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Tofig Bakhshiyev**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Hard and Soft Tuning of Spark Ecosystem Toward Query Energy Efficiency,
   (title of thesis)

   supervised by Simon Pierre Dembele.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Tofig Bakhshiyev
15/05/2024