

Titre: Benchmarking Framework and Performance Modeling for Evaluating
the Performance of Spark-Based Data Science Projects

Auteur: Soude Ghari
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Ghari, S. (2023). Benchmarking Framework and Performance Modeling for
Evaluating the Performance of Spark-Based Data Science Projects [Thèse de
doctorat, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/55740/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/55740/>
PolyPublie URL:

**Directeurs de
recherche:** Heng Li, & Marios-Eleftherios Fokaefs
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Benchmarking Framework and Performance Modeling for Evaluating the
Performance of Spark-based Data Science Projects**

SOUDE GHARI

Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Août 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Benchmarking Framework and Performance Modeling for Evaluating the
Performance of Spark-based Data Science Projects**

présentée par **Soude GHARI**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Ettore MERLO, président

Heng LI, membre et directeur de recherche

Marios-Eleftherios FOKAEFS, membre et codirecteur de recherche

Zohreh SHARAFI, membre

Marin LITOIU, membre externe

RÉSUMÉ

La récente augmentation de l'adoption des sciences des données et du big data a eu un impact significatif sur Comment nous prenons des décisions basées sur les données. À mesure que ces technologies se sont répandues, la demande de modélisation et d'évaluation des performances des applications de sciences des données a augmenté. Pour répondre à ce besoin, une plateforme de référence a été développée pour évaluer les performances de ces applications. Cette plateforme intègre une phase de collecte de données où des métriques pertinentes sont rassemblées. Ces métriques sont essentielles pour comprendre et quantifier les performances du système. Les données collectées sont ensuite utilisées dans différents modèles de régression classiques et d'apprentissage en profondeur. Ces modèles sont entraînés pour prédire deux indicateurs clés de performance. Le débit indique la quantité de travail qu'un système peut gérer dans une période désignée, tandis que le temps de réponse fait référence à la vitesse à laquelle le système peut réagir et traiter les demandes entrantes. Les prédictions du débit et du temps de réponse sont basées sur différentes caractéristiques de charge de travail, configurations de la topologie Spark et configurations de l'infrastructure. En prenant en compte ces facteurs, la plateforme de référence permet une évaluation globale des performances des applications de sciences des données.

ABSTRACT

Machine learning and big data analytics are of great importance to enterprise business operations, aiming to uncover data-driven insights. Considerable effort is put into developing and deploying machine learning models, which are utilized for estimating and predicting economic factors, and even for interacting with clients. However, this planning remains largely dependent on human acumen and is expensive to determine in a systematic fashion with automated tools. Benchmarking is the process of efficiently running experiments to determine a system's performance requirements, among others, in order to aid planning and resource allocation. Benchmarking intelligent and data-intensive systems remains in its infancy and does not cover fully realistic or particular case studies. Benchmarking holds significant practical importance in various fields and industries. It provides a valuable tool for organizations to measure their performance, and make informed decisions. However, there are two significant challenges faced by these systems: volatility and variability. The latter concerns numerous conditions to be taken into consideration when evaluating the performance of analytics systems. The former denotes that any of these circumstances could vary at any time. The challenge is to be able to accurately estimate the performance and effectiveness of a data science platform in the presence of uncertainty and variability. Such an estimate can enable informed planning and potentially dynamic resource allocation for such projects. Moreover, we present a study of multiple machine learning models from simple one linear regression to more complicated ones like LSTM and MLP to estimate the performance of data science projects deployed in Apache Spark, a popular and flexible distributed analytics platform. We demonstrate the process of training such a model, from data collection to training and testing, and we systematically compare the various alternatives to help decision-makers choose the best configuration. The significant scientific contribution of this thesis can be found in the fact that machine learning testing tools and benchmarks primarily focus on assessing model performance rather than the software performance of the implementation. This emphasizes the significance of longitudinal workloads, as they play a pivotal role in impacting resource and time performance over the long term by saturation, unlike model accuracy, which may remain relatively stable. In summary, while batch workloads, predominantly employed by most benchmark tools, are appropriate for evaluating model accuracy, they may not be as suitable for evaluating software performance. Concerning the model itself, a substantial scientific contribution emerges from the incorporation of statistical testing within your pipeline for model comparison, a feature notably absent in many tools like AutoML.

TABLE OF CONTENTS

RÉSUMÉ	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	x
CHAPTER 1 INTRODUCTION	1
1.1 Element of the problem	1
1.2 Motivational example	3
1.3 Research objectives	5
1.3.1 Objective 1 : Machine learning benchmarking of Spark-based data science projects	5
1.3.2 Objective 2 : Performance modeling of Spark-based data science projects	7
1.4 Scientific contributions of the thesis	8
1.5 Structure of the thesis	10
CHAPTER 2 LITERATURE REVIEW	11
2.1 Background	11
2.1.1 Big data analytics	11
2.1.2 Apache Spark	13
2.1.3 System performnace	13
2.1.4 Performance model	14
2.2 Related work on benchmarking big data analytics	15
2.2.1 Taxonomy of big data benchmarks :	15
2.2.2 Benchmarking Techniques	16
2.2.3 Benchmarking Machine learning models	19
2.3 Related work on performance modeling of big data analytics	20
CHAPTER 3 SPARKPERF : A MACHINE LEARNING BENCHMARKING FRA- MEWORK FOR SPARK-BASED DATA SCIENCE PROJECTS	23
3.1 Preamble	23

3.2	Introduction	23
3.3	The SparkPerf benchmarking framework	25
3.3.1	Functional and Non-Functional Requirements	25
3.3.2	Components	28
3.4	Case studies	41
3.4.1	Experimental Setup	41
3.4.2	Case study 1) Portability	43
3.4.3	Customizability and extensibility	44
3.5	Reliability evaluation	46
3.6	Strength and limitation	50
3.7	Conclusion	52
CHAPTER 4 SPARKSIM : A PERFORMANCE MODEL FOR RESOURCE ALLO-		
CATION OF SPARK-BASED DATA SCIENCE PROJECTS		53
4.1	Preamble	53
4.2	Introduction	53
4.3	SparkSim model creation pipeline	55
4.3.1	Workload Design	55
4.3.2	Experiment execution and feature monitoring	58
4.3.3	Experimental setup	60
4.4	Performance modeling	61
4.4.1	Model evaluation	64
4.5	Model Validation	65
4.6	Results and Analysis	66
4.7	Discussion	72
4.8	Conclusion	74
CHAPTER 5 DISCUSSION		76
5.1	Integration the benchmarking with development environment : enhancing de-	
	cision support	77
5.2	Impact and contribution of the work	80
5.2.1	Representativeness of data	81
5.2.2	Generalization of findings :	83
CHAPTER 6 CONCLUSION		84
6.1	Summary of Works	84
6.2	Limitation	85

6.3 Conclusion	86
REFERENCES	87

LIST OF TABLES

Table 2.1	Overview of the related work on performance modeling with respect to the models developed, type of workload, set of input variables, targeted output variables, and whether a work has considered feature importance (The key difference of our work is utilizing response time and throughput metrics to capture the combined impact of transactional workload generators, Spark settings, and the underlying infrastructure simultaneously, therefore, the key difference lies in our case study. . .	18
Table 2.2	Comparison of benchmarking techniques with SparkPerf.	22
Table 3.1	Monitoring performance metrics and resource utilization	40
Table 3.2	Characteristics of experiment 1 and 2 regarding reliability case study	41
Table 3.3	Characteristic of the experiment with workload mixes regarding reliability case study	42
Table 3.4	Measuring Average and Standard Deviation for Testing Reliability of Experiment 1	42
Table 3.5	Measuring Average and Standard Deviation for Testing Reliability of Experiment 2	43
Table 3.6	The experimental characteristics of on-perm and remote server setups along with average response time	45
Table 3.7	Measuring Standard deviation for testing reliability of workload mixes for Average response time metric	45
Table 3.8	Measuring Standard deviation for testing reliability of workload mixes for throughput metric	47
Table 3.9	Characteristic of XGBoost classifier and Random forest	47
Table 4.1	Description of the Features and Output variables.	57
Table 4.2	Experimental Data for Machine Learning Model Training and Model Validation	59
Table 4.3	Hyper-parameter Tuning for the Different Models	62
Table 4.4	Experimental Configuration of the VM and Spark Cluster	64
Table 4.5	Comparison of Different Performance Evaluation Metrics and their corresponding Spearman rank coefficient : The top table shows the results of the Response Time Models, and the bottom table shows the results of the Throughput Models, both on the validation data.	68

Table 4.6	Whilcoxon signed-ranked test for measuring the significance of top 5 features	72
-----------	---	----

LIST OF FIGURES

Figure 3.1	The SparkPerf component structure	29
Figure 3.2	Django admin panel as an interface	31
Figure 3.3	A view of Spark Grafana Dashboard for the response time metric . .	31
Figure 3.4	Comparative Analysis (a) XGBoost classifier (b) Random forest in terms of throughput and (c) XGBoost classifier (d) Random forest classifier in terms of response time	48
Figure 4.1	The SparkSim workflow including workload generation and performance monitoring, model training, and the performance model	56
Figure 4.2	ScottKnot Test for Comparing ML Models Regarding Top Model as Response Time and Bottom as Throughput	67
Figure 4.3	ScottKnot Test for Comparing ML Models Regarding Top Model as Response Time and Bottom as Throughput	69
Figure 5.1	Integration approach components	78

CHAPTER 1 INTRODUCTION

1.1 Element of the problem

Data science has become an indispensable component of enterprise business operations, with a primary focus on uncovering valuable insights through data analysis. The complexity of data science projects presents a significant challenge, as they involve intricate and multifaceted components. Data science projects often require significant computing resources due to several reasons, and one of the primary factors is the complexity and size of the data being analyzed. As datasets grow in size, the computational demands increase proportionally, making it necessary to utilize more powerful and expensive computing resources. In addition to that, other factors come into play, such as the intricacy of the algorithms employed by data scientists and the implementation of parallel processing, among others.

Navigating and successfully completing such projects can be difficult, requiring precise estimation of resource requirements and associated costs to ensure the project is executed successfully. Accurately estimating computing resources for data science projects is of paramount importance to ensure successful execution, cost management, and time efficiency. Proper resource allocation avoids bottlenecks, enhances performance, and maximizes the quality of results. It allows for scalability, risk mitigation, and optimal return on investment, while also fostering team productivity. By avoiding wastage and under-allocation of resources, data science projects can navigate challenges effectively and deliver high-quality outcomes within set timelines. The primary challenge is to enhance the accuracy of planning for resource costs and human resources in data science projects.

Benchmarking plays a crucial role in addressing the challenge of enhancing the accuracy of planning for resource costs and human resources in data science projects. By comparing resource allocation and cost estimation with industry best practices, benchmarking enables performance comparison, identifies inefficiencies, and optimizes resource utilization [1]. It facilitates informed decision-making, sets realistic goals, and mitigates risks by learning from successful projects. Additionally, adopting best practices through benchmarking fosters a culture of continuous improvement, empowering data science teams to refine their resource planning strategies and achieve successful project outcomes.

Moreover, benchmarking enables the gathering of data and facilitates cost-effective experimentation, while also enabling comparisons of performance across various applications or hardware setups.

By conducting benchmarking exercises, data scientists can gather valuable insights and data on resource utilization, computational requirements, and execution times of data science projects. Such empirical evidence aids in evaluating the performance of different applications, identifying areas for improvement, and making informed decisions regarding resource allocation [2]. Moreover, benchmarking allows for iterative refinement of project plans, risk mitigation, and optimization of resource utilization, leading to more accurate resource cost and human resource planning in data science projects [3].

The current benchmarks available for big data analytics [4–8] have either been designed for frameworks like Hadoop or Hive or lack the necessary specificity to provide comprehensive insights specifically tailored to Spark, which is a prominent platform for big data analytics. In addition, these benchmarks primarily focus on “Batch workloads”, where the entire workload is executed until completion or failure, without considering long-term scalability and sustainability when dealing with continuous and fluctuating workloads [1]. In contrast, our framework leverages “Transactional workloads”, which involve a dynamic number of virtual users continuously sending requests throughout the experiment’s execution [1]. Transactional workloads are more important in many scenarios due to their relevance in real-time and critical operations. Unlike batch workloads that focus on processing large volumes of data, transactional workloads deal with individual transactions or smaller sets of data but require quick and precise handling. The following exemplar scenario indicates the significance of transactional workloads :

In the context of an insurance company, transactional workloads are of utmost importance. In this scenario, the significance of transactional workloads becomes evident due to the following reasons : In an insurance company with varying numbers of data scientists, each generating job submissions, the importance of transactional workloads becomes evident. These workloads play a crucial role in processing real-time tasks and accommodating concurrent job requests from a diverse number of data scientists. The efficient handling of sending requests ensures prompt response times, optimal resource utilization, and seamless coordination among the data scientists, leading to enhanced productivity and streamlined data-driven decision-making processes.

In fact, a batch workload typically refers to a workload that includes a single task that runs once and continues until it completes or stops [8]. In contrast, a transactional workload involves executing multiple small work units in an iterative manner throughout a benchmark’s duration [1]. Our framework, leverages “open transactional workloads”, which involve a dynamic number of virtual users continuously sending requests throughout the experiment [1].

The second challenge is the volatility and variability faced by data science systems, as well

as the need for accurately estimating performance in the presence of uncertainty and variability. Volatility refers to the unpredictable nature of circumstances surrounding data science systems. It means that any given condition or factor can change unexpectedly at any time. On the other hand, variability refers to the numerous conditions and factors that need to be considered when evaluating the performance of analytics systems. These factors can vary and impact the system’s performance. Accurately estimating the performance of data science systems in the presence of uncertainty and variability is a major challenge. A precise performance estimate is crucial for effective planning and dynamic resource allocation in data science projects. This study focuses on examining different machine learning and deep learning models to address this challenge specifically in the context of data science projects deployed in Apache Spark, as an analytics platform.

1.2 Motivational example

The current digital era is defined by the exponential growth of data across various facets of life, prompting businesses to integrate data acquisition and analysis into their core operations. In fact, in the financial services domain, data-driven predictions have become crucial for achieving objectives effectively. However, the complexity of big data analytics, combined with the intricacies of machine learning and artificial intelligence, presents challenges in optimizing analytics development, deployment, and operations [9].

In an imagined scenario, Bob is a project manager leading a large data science team in a prominent enterprise and faces a challenge. His team specializes in analyzing big data analytics and machine learning models, with a desire to enhance performance and optimize resource allocation costs. They are eager to identify the optimal configuration that strikes the right balance between performance and cost efficiency. To tackle this, they seek clarity on when to leverage the benefits of the public cloud and when to utilize on-premises resources. By understanding the scenarios that necessitate public cloud usage and those that can be managed on-premises, they aim to make informed trade-offs and strike the perfect equilibrium between on-prem and public cloud resources.

In Bob’s case, a benchmarking framework and performance modeling can help his data science team evaluate the performance of their analytics and machine learning models across various scenarios, considering both on-premises and public cloud resources.

In fact, the benchmarking framework starts by defining a range of scenarios that represent different data processing and machine learning workloads. These scenarios can vary in terms of data volume, complexity, processing requirements, and other relevant factors. Then, Bob’s

team selects appropriate performance metrics that align with their goals, such as response time, throughput, and cost. In Bob's context, an important metric called response time reflects the time it takes for their data science tasks, such as running analytics and machine learning models, to be executed successfully i.e. response time. *On-Premises vs. Cloud Comparison* : By measuring response time, throughput, and cost for different scenarios on both on-premises and cloud environments, Bob's team can compare how each configuration affects the response time of each task. *Identifying Bottlenecks* : Longer response time might indicate performance bottlenecks, such as insufficient processing power. Bob's team can analyze these bottlenecks to identify areas where resource optimization is needed. *Optimization* : Through benchmarking and experimentation, the team can find resource configurations that result in shorter execution times. This optimization enhances the efficiency of task completion.

Moreover, performance modeling significantly helps Bob and his data science team in the challenge of optimizing resource allocation and striking the right balance between on-premises and public cloud resources. *Identifying Optimal Configurations* : Performance modeling allows Bob's team to simulate and analyze the behavior of their analytics and machine learning models under different configurations. By running simulations, they can assess how various combinations of on-premises and cloud resources impact performance metrics like processing time, accuracy, and cost. This helps them identify the optimal configuration that maximizes performance while minimizing costs. *Cost-Efficiency Analysis* : With performance modeling, Bob's team can estimate the cost associated with running their workloads on both on-premises infrastructure and public cloud platforms. By comparing these cost estimates against the expected performance improvements, they can make informed decisions about resource allocation. This analysis helps in optimizing resource utilization and cost-efficiency, ensuring that they get the most value for their investment. *Scenario Evaluation* : Performance modeling allows Bob's team to evaluate different scenarios and workload characteristics. For example, they can assess how well a specific machine learning model performs on-premises for a given dataset size, and compare it to the performance when using cloud resources. This helps them understand which scenarios necessitate the use of the public cloud due to scalability or resource limitations and when on-premises resources are sufficient. *Scalability Planning* : Performance modeling helps in planning for scalability. Bob's team can analyze how well their models can scale on-premises and in the cloud. This is crucial for accommodating growing workloads and ensuring that they can handle increased data volumes and complexity in a cost-effective manner. *Risk Mitigation* : Understanding the performance characteristics of different scenarios through modeling helps mitigate risks. Bob's team can anticipate potential bottlenecks, failures, or unexpected costs associated with resource allocation decisions, allowing them to proactively address these issues.

In summary, performance modeling and benchmarking provide Bob’s data science team with a systematic and data-driven approach to assess, optimize, and make informed decisions about resource allocation, cost efficiency, and performance improvements. This empowers them to achieve the desired balance between on-premises and public cloud resources, ultimately leading to more effective data analytics and machine learning model deployment.

1.3 Research objectives

1.3.1 Objective 1 : Machine learning benchmarking of Spark-based data science projects

Machine learning has become increasingly prominent in data science workflows due to its ability to extract insights and patterns from vast amounts of data. Moreover, machine learning tasks are computationally intensive, requiring substantial computing resources. Accurate performance evaluation of these models is crucial for optimizing resource allocation and ensuring efficient utilization of resources. In addition, the evaluation of machine learning models can provide valuable insights, guiding data scientists in making better decisions and enhancing the overall effectiveness of data science projects. Moreover, Spark is a powerful platform for processing big data analytics [10] [11]. Thus, we propose SparkPerf, a machine-learning benchmarking framework for Spark-based data science projects.

With the increasing complexity and scale of data science projects, the evaluation of the performance of machine learning pipelines has currently become a necessity. Performance evaluation and comparison of Machine Learning models become essential for efficient decision-making regarding resource allocation, and optimization, ensuring data scientists make informed choices that lead to successful and impactful data-driven solutions. However, existing benchmarks often have limitations, such as support only for batch workloads and lack the ability to run user datasets and/or applications or generate synthetic datasets for scaling purposes. To address these limitations, we developed SparkPerf, a machine learning benchmarking framework tailored for assessing Spark-based data science projects. SparkPerf allows users to specify workload characteristics, including customizing workload, and configure Spark cluster settings and infrastructure configuration. The framework employs a transactional workload generator to simulate real-world scenarios and measures performance metrics like throughput, response time, CPU utilization, memory consumption, and so on. These metrics are stored in Prometheus and visualized through Grafana. Through various experiments, we have evaluated SparkPerf’s portability, reliability, customizability, ability to handle user data and applications, and generation of datasets for scaling purposes. These evaluations provide va-

uable insights into SparkPerf's performance and capabilities. The ultimate goal is to enhance decision-making processes and optimize business operations through data-driven insights.

To achieve the objective outlined for the benchmarking platform for Spark, the following plan can be considered :

Requirement Analysis : Conduct a thorough analysis of the requirements for the benchmarking platform. Identify the key features, functionalities, and performance metrics that need to be supported to address the objectives.

Platform Design and Development : Design and develop the benchmarking platform, ensuring it is capable of accommodating continuous and fluctuating workloads. Implement the necessary components to support custom workloads, data, and jobs. Design a user-friendly interface for users to interact with the platform.

Workload Generation : Develop mechanisms to generate realistic and dynamic workloads that mimic real-world scenarios. Consider various workload patterns, such as bursty, periodic, or fluctuating workloads, to assess Spark's performance under different conditions.

Scalability and Sustainability Testing : Design and execute tests to evaluate Spark's scalability and sustainability. This includes measuring the system's ability to handle increasing data volumes, sustained workloads, and extended periods of operation. Collect performance data to analyze Spark's behavior and identify any limitations or bottlenecks.

Customization and Flexibility : Implement features that allow users to run their own workloads, utilize their own datasets, and execute specific jobs on the benchmarking platform. Provide guidelines and tools for users to define and configure their custom workloads effectively.

Performance Metrics : Defining a number of performance metrics to measure Spark's performance consistently. Establish guidelines and methodologies for collecting, analyzing, and reporting benchmarking results. Ensure that the metrics chosen align with the objectives of the benchmarking platform.

Documentation and User Support : Develop comprehensive documentation that explains the platform's functionalities, usage guidelines, and best practices. Provide user support through tutorials, forums, and assistance channels to help users effectively utilize the benchmarking platform.

1.3.2 Objective 2 : Performance modeling of Spark-based data science projects

The second objective is to provide valuable insights and techniques for addressing the challenges of volatility and variability in data science systems, as well as enhancing decision-making processes and optimizing business operations through data-driven insights. We present a study of various machine learning models to estimate the performance of data science projects deployed in Apache Spark, a popular and flexible distributed analytics platform. We demonstrate the process of training such a model, from data collection to training and testing, and we systematically compare the various alternatives to help decision-makers choose the best one.

To address the second objective of providing valuable insights and techniques for addressing the challenges of volatility and variability in data science systems and enhancing decision-making processes through data-driven insights, the following plan can be considered :

- Identify Key Variables : Determine the key variables that contribute to volatility and variability in data science systems. These variables could include data characteristics, workload patterns, resource allocation, system configurations, and external factors impacting system performance.
- Data Collection : Collect relevant data to analyze and understand the impact of volatility and variability on data science system performance. This could involve real-world datasets, synthetic datasets, or simulated data to capture various scenarios.
- Performance Metrics : Define performance metrics that capture the impact of volatility and variability on data science system performance. These metrics could include response time, throughput, resource utilization, accuracy, and robustness.
- Experimental Design : Design experiments to systematically evaluate the performance of data science systems under varying levels of volatility and variability. Define test cases that represent different scenarios and combinations of key variables.
- Performance Evaluation : Execute the experiments and collect performance data. Measure the performance metrics and analyze the results to identify patterns, trends, and correlations between volatility, variability, and system performance.
- Performance Optimization : Develop strategies and approaches to optimize the performance of data science systems under volatility and variability. This could involve workload management techniques, resource allocation strategies, or system tuning to mitigate the impact of changing conditions.

1.4 Scientific contributions of the thesis

Scientific contributions in research typically refer to the novel and innovative aspects of a study that advances the state of knowledge in a particular field. These contributions are not focused on the engineering or practical implementation aspects but on the theoretical or conceptual innovations. Here are some key points to the scientific contributions of this thesis :

Benchmarking and performance modeling with transactional workloads are essential for several reason :

Transactional workloads often involve critical business processes, such as online transactions, financial transactions, and e-commerce operations. Ensuring optimal performance is crucial to meeting customer expectations, maintaining service-level agreements (SLAs), and maximizing efficiency. Transactional systems can experience various bottlenecks, such as database contention, I/O latency, or network congestion. Benchmarking helps pinpoint these bottlenecks, allowing organizations to address them and improve system performance. Consistent and predictable performance is essential to ensure data integrity and avoid issues like data corruption or lost transactions. Benchmarking helps verify that the system can handle transactional workloads without compromising data quality.

Moreover, understanding and analyzing the performance characteristics of transactional workloads is vital in the contemporary technology landscape. With businesses relying on these workloads for essential operations, ensuring optimal resource utilization, fault tolerance, and user experience is imperative. Also, accurate performance modeling allows organizations to plan for future scalability, contributing to cost savings. As the scale and complexity of these workloads continue to grow, ensuring optimal performance becomes increasingly challenging yet essential.

There are several compelling reasons for delving into performance modeling specifically tailored to transactional workloads : Performance modeling provides a systematic approach to optimize the allocation of resources, including compute power, memory, and network bandwidth. Efficient resource utilization can result in cost savings and enhanced system responsiveness. In a world where user expectations for responsiveness and reliability are exceptionally high, performance modeling helps in ensuring that transactional systems consistently meet these expectations. A smooth user experience is crucial for customer satisfaction. Accurate performance modeling facilitates effective capacity planning. Organizations can predict future demands and scale their infrastructure accordingly, preventing potential bottlenecks and system failures during peak usage periods. Furthermore, understanding the performance

characteristics of transactional workloads aids in designing fault-tolerant systems. By modeling how the workload behaves under various conditions, it becomes possible to build resilient architectures that can withstand failures gracefully. Businesses that can deliver superior performance in their transactional systems gain a competitive edge. Performance modeling enables organizations to fine-tune their services, potentially outperforming competitors and attracting a larger customer base.

Benchmarking and performance modeling for Spark-based data science projects with transactional workloads pose several unique challenges. These challenges are related to the specific characteristics of both Spark and transactional workloads. Here are some key challenges and considerations :

- Real-world Variability : Transactional workloads often experience fluctuations in demand, making it challenging to create realistic benchmarks that accurately mimic production scenarios. Real-world variations in user activity, data volume, and system load must be considered.
- Heterogeneous Workloads : Transactional workloads in data science projects can vary widely in terms of data size, complexity, and processing requirements. Designing benchmarks that represent this heterogeneity can be challenging.
- Scalability Testing : Evaluating how a transactional system scales under load can be challenging, especially when dealing with distributed systems or cloud-based infrastructure. Testing scalability often requires significant resources and expertise.

To overcome these challenges when benchmarking transactional workloads, consider the following strategies :

- Workload Modeling : Develop workload models that capture the expected transaction mix, concurrency, and response time requirements.
- Monitoring and Visualization : Employ monitoring tools like Prometheus to capture system metrics and identify bottlenecks.

Software Performance Benchmarking for ML Models : This research highlights a critical gap in existing ML test tools and benchmarks. While many tools focus primarily on evaluating model performance, they often neglect the crucial aspect of the performance of specific implementations. This is a valuable scientific contribution because it recognizes that real-world ML applications not only require accurate models but also efficient software implementations to maintain performance over time. By emphasizing the importance of longitudinal workloads and their impact on resource and time performance, this research addresses a significant research gap and provides a more comprehensive framework for evaluating ML systems.

Statistical Testing for Model Comparison : Another noteworthy scientific contribution is the incorporation of statistical testing for model comparison within our pipeline. This addition is particularly important because it enhances the rigor of model evaluation and selection. Many AutoML tools and traditional benchmarks may lack this statistical rigor, potentially leading to less reliable model choices. By introducing statistical testing, this research is contributing to the development of a more robust and scientifically sound approach to model selection, improving the overall quality and trustworthiness of ML pipelines and methodologies, (which is not the case for AutoML tools for example).

1.5 Structure of the thesis

The remainder of the thesis is organized as follows. Chapter 2 provides a set of related works in this field. Chapter 3 delves into our first objective, which focuses on the machine learning benchmarking framework. In Chapter 4, we explore our second objective, which involves building performance modeling based on machine learning and deep learning models. Chapter 5 offers a comprehensive discussion of our work, and lastly, chapter 6 presents the concluding remarks of the thesis.

CHAPTER 2 LITERATURE REVIEW

2.1 Background

2.1.1 Big data analytics

Big data indicates vast, intricate, and heterogeneous datasets. These datasets are distinguished by the “three V’s : Volume, Velocity, and Variety” [12] [13] [14]. Volume : Big data is characterized by an immense amount of data [15] [14]. It encompasses datasets of such magnitude that they exceed the capacity for efficient processing. or stored using conventional databases or tools [15]. Velocity : Big data is produced, collected, and processed in real-time [15]. Variety : Big data exists in various manifestations and forms [13], including “structured, semi-structured, and unstructured [16] data” [17] in its entirety [13]. Structured data refers to well-defined data [18]. Semi-structured data includes data with structure [11], such as XML [16]. Unstructured data encompasses information that lacks a preset or organized structure [13,16].

Apart from the three V’s, big data is also characterized by other aspects :

To effectively leverage big data, organizations, and researchers use advanced technologies, tools, and methodologies, such as distributed computing frameworks like Apache Spark, data lakes, machine learning algorithms, and data visualization techniques. These enable the storage, processing, analysis, and extraction of Meaningful knowledge or valuable information derived from extensive and intricate datasets.

Utilizing big data analysis helps organizations to make better decisions and gain deeper insights into customer behavior, optimize operations, enhance predictive analytics, support scientific research, and drive innovation in various domains, including finance, healthcare, marketing, logistics, and more [19].

Big data analytics denotes the process of obtaining actionable insights, patterns, and knowledge from large, complex, and diverse datasets known as big data. It includes the utilization of sophisticated analytical techniques, tools, and technologies that can provide important insights for decision-making, problem-solving, and strategic planning [13].

Some key facets of big data analytics are shown below :

Data Storage : Big data analytics often [20] requires specialized storage and management systems capable of handling large volumes of data. This includes distributed file systems, data lakes, NoSQL databases, and other scalable storage solutions that can efficiently store and

retrieve big data. Data Exploration and Visualization : Exploratory data analysis techniques are applied to understand the characteristics and relationships within the data. Visualization tools are employed to present the data in meaningful and interactive ways, enabling analysts and decision-makers to acquire knowledge and make well-informed decisions [21,22]. Advanced Analytics Techniques : Big data analytics involves applying various advanced analytics techniques to uncover patterns, perform statistical analysis, and build predictive models [23]. Scalable Processing and Parallel Computing : Given the extensive magnitude and intricate nature of big data, scalable processing frameworks and parallel computing techniques are used to distribute computations across multiple nodes or clusters [24].

Big data analytics is a critical technique that has gained momentum in recent years by assisting businesses in making informed decisions and improving the customer experience.

Four kinds of big data analytics are as follows “descriptive analytics, diagnostic analytics, predictive analytics, and prescriptive analytics” [25]¹. Descriptive analytics intends to describe historical data to figure out what happened in the past, which involves data aggregation, data mining, and statistical analysis to determine patterns, trends, and insights from large data-sets [26]. Exploratory Analytics aims to discover the reason “Why it happened” by exploring historical data and then detecting the main reason for a specific outcome [27]. Exploratory analytics employs approaches such as regression analysis, correlation analysis, and data visualization to uncover the unexpected relationships between various variables and identify these variables that have an impact on a specific outcome [28]. Predictive analytics predicts what is expected to happen in the upcoming period using extrapolation from the data that is currently available [29]. Predictive analytics is widely used to better understand customers’ requirements and expectations. Prescriptive analytics is concerned with recommending actions based on insights given by descriptive, diagnostic, and predictive analytics. Prescriptive analytics would be the usage of optimization and simulation platforms [25,30].

In this thesis, we applied predictive analytics, and we are going to predict two performance metrics, including response time and throughput. By analyzing gathered data in this way, we can gain insights into how the data analytics system has performed and identify factors that may have contributed to variations in response time and throughput variables. This can help practitioners and software developers make more accurate predictions about future performance and take proactive steps to improve response time and throughput if necessary.

1. <https://hevodata.com/learn/data-analyst-company/>

2.1.2 Apache Spark

“Apache Spark is an open-source distributed computing” [10] framework developed for the purpose of processing and analyzing large-scale datasets” [31] [10]. It offers a programming model and an execution system that enables the streamlining of the processing of large datasets across a cluster of machines to achieve optimal performance and efficiency [32]. Spark is known for its speed, scalability, and support for various data processing tasks, including data science and machine learning [10].

2.1.3 System performnace

Response time :

“Response time is the time taken for a system to respond to a request of a user [1]”.

Throughput :

Throughput refers to the capacity or efficiency of a system or process in terms of the rate or quantity of work accomplished within a specific time frame [1]. It measures the number of tasks, operations, or data units processed or delivered per unit of time. In the context of data science Spark projects, throughput typically refers to the rate at which data is processed or the number of computations performed by Spark within a specific time frame. It provides insights into the system’s capacity to handle workloads efficiently and its ability to handle data expeditiously [33].

Response time and throughput are important performance metrics for evaluating big data systems. Spark is widespread for processing large-scale data analytics, which often involves complex computations and transformations on big data sets [10]. Response time is a crucial metric in analytical scenarios, as it directly affects the time it takes to get insights from data. Analysts and data scientists need timely results for decision-making and insights generation. However, they are not the only metrics available. The choice of metrics focuses on the specific goals of the evaluation, the characteristics of the system, and the requirements of the application. Other metrics that could be relevant for evaluating big data systems include :

Latency

Latency measures the time lag from when a request is started to when the initial response is received [34]. Unlike response time, which may include additional processing, latency specifically focuses on the time to receive the initial result.

Resource utilisation

In addition to assessing resource utilization of big data analytics during task execution, we

can also measure how effectively resources are utilized over time. This metric can help identify periods of overutilization or underutilization, leading to resource optimization.

In fact, the reason behind choosing response time and throughput as important metrics in our data analysis is that these are the only metrics that capture the effect of both workload generator and Spark setting as well as infrastructure at the same time, also Latency is included in response time, as latency is typically a component of response time. Therefore, response time includes latency as well as other components, such as the time it takes for the system to process the request fully, generate a complete response, and send that response back to the requester.

Moreover, the selection of performance metrics including response time and throughput is a result of a collaborative discussion with our industry partner. These metrics were chosen for several reasons : Response time directly reflects the system’s ability to respond to user requests promptly, a critical aspect of user satisfaction. By measuring and optimizing response time, the organization can ensure that its services meet or exceed customer expectations. Throughput, on the other hand, was identified as a crucial metric to assess the system’s efficiency and scalability. Our industry partner aims to handle an increasing volume of transactions as their business grows. Monitoring the number of served requests during a given time provides insights into the system’s capacity to handle concurrent requests and transactions efficiently. It helps in capacity planning, ensuring that the infrastructure can support higher workloads without compromising performance. Indeed, both response time and throughput are highly measurable metrics. They provide clear and actionable data that can be used to identify performance bottlenecks, diagnose issues, and implement targeted optimizations. This aligns with our industry partner’s desire for data-driven decision-making.

2.1.4 Performance model

A performance model is a conceptual or mathematical model that aims to understand, predict, and analyze, the performance of a system, process, or task [35]. It provides insights into the elements that impact performance and helps in making informed decisions to improve efficiency. In the context of resource allocation for data science Spark projects, a performance model would involve understanding and quantifying the relationships between various components that influence the performance of these projects. These components may incorporate computational resources (such as CPU cores, memory, and disk space), data characteristics (volume, velocity, variety), workload patterns, algorithmic complexity, and system configurations.

A performance model typically involves the following components : Input Parameters : These

are the variables or factors that influence the performance of the system. For resource allocation in data science Spark projects, input parameters may include the available computing resources, size and characteristics of the dataset, complexity of the data processing tasks, and the allocation strategy employed. Metrics : These are the quantitative measures used to evaluate performance. Common performance metrics for Spark projects can include execution time, throughput, resource utilization, memory consumption, and scalability. Relationships and Dependencies : The performance model establishes relationships and dependencies between the input parameters and performance metrics.

For example, it may quantify how increasing the number of computing resources improves the Spark job’s execution time or how increasing the dataset size affects memory consumption and scalability. Mathematical Formulation : A performance model often involves mathematical equations, algorithms, or statistical models to represent the relationships between input parameters and performance metrics [35]. These equations can be derived from theoretical analysis, empirical observations, or a combination of both. Validation and Calibration : A performance model needs to be validated and calibrated using real-world data and experiments to ensure its accuracy and reliability. This involves comparing the model predictions with actual performance measurements and making adjustments if necessary.

By using a performance model for resource allocation in data science Spark projects, researchers and practitioners can gain insights into how different resource configurations and allocation strategies impact performance. The model can guide decision-making processes, such as determining the optimal allocation of computational resources to maximize efficiency or identifying potential bottlenecks and performance limitations in the system. Overall, a performance model provides a systematic and quantitative approach to understanding and optimizing the resource allocation process in data science Spark projects, helping to improve the overall performance and productivity of these projects.

2.2 Related work on benchmarking big data analytics

2.2.1 Taxonomy of big data benchmarks :

Current benchmarks can be categorized into three distinct groups based on the range of benchmarking they encompass [36].

Micro Benchmarks : These are small-scale benchmarks that evaluate the performance of specific components, including CPU, and memory [36,37].

End-to-End Benchmarks : These benchmarks evaluate the performance of complete applications, such as database management systems, data warehousing systems, and machine

learning applications. They provide a more realistic assessment of the performance of such systems in real-world scenarios [36, 38].

Benchmark suite : These kinds of benchmarks are collections of diverse “micro and/or end-to-end” benchmarks [16] [36] [39] designed to provide thorough benchmarking solutions. These suites aim to cover an extensive variety of evaluation scenarios. Benchmark suites such as Hibench [8], and BigDataBench [4] are notable examples. This kind of benchmark generates synthetic data sets and workloads that mimic real-world scenarios. They are applied to assess the scalability and performance of big data systems under different workloads and data volumes [36].

2.2.2 Benchmarking Techniques

In this section, we explore several benchmarking approaches that are currently in use.

BigBench [6] is a benchmarking tool designed to assess the performance of big data analytics platforms. It includes a set of complex queries and data sets.

HiBench [8] is a benchmarking tool for Hadoop-based systems. It includes a set of benchmarks that evaluate the performance of systems based on Hadoop in various scenarios.

LinkBench [40] is a database benchmark designed to emulate real-world database tasks in social applications. The authors analyze data from social graphs and its associated database workload from Facebook, highlighting significant statistical distributions, particularly power law distributions, across various areas. They proceed to detail the creation of a benchmark that emulates critical aspects of this database workload and present an evaluation of the MySQL database system’s performance when subjected to this workload. Notably, they have made the benchmark software available as open-source, facilitating its utilization by other researchers for profiling and experimentation with different database systems.

CloudSuite [41] the micro-architectural performance of various scale-out cloud applications and pinpoints the primary causes of inefficiency in terms of both space utilization and power consumption within the instruction retrieval, core micro-structure, and memory system arrangement. The document proposes potential architectural adjustments that may result in more compact and energy-efficient processor designs for data centers in the foreseeable future. Additionally, it explores the benefits of adopting the map-reduce approach for managing extensive analyses, as well as for segregating infrastructure operations from machine learning algorithms.

Gridmix [42] is a benchmarking tool that generates synthetic workloads to assess the performance of big data systems. It simulates a range of workloads, such as machine learning, and

can be used to test the performance of various big data systems, like Apache Hadoop.

SparkBench [7] is a benchmarking suite designed to evaluate the performance of Apache Spark. It includes a set of benchmarks that simulate a range of workloads, such as machine learning and streaming data processing. SparkBench can assist enterprises in identifying performance bottlenecks and optimizing their Spark applications for improved performance and scalability.

BigDataBench [4] is a comprehensive big data benchmarking suite that includes a diverse array of workloads, from machine learning to scientific computing and social network analysis. It is created to analyze the performance of big data systems and applications in different scenarios and with different workloads. BigDataBench can help organizations analyze the performance of various big data systems and applications and make informed decisions when selecting hardware and software.

StreamBench [43] serves as a benchmarking framework aimed at assessing modern distributed stream computing frameworks. The paper delves into the complexities and prerequisites of this field, offering a comprehensive benchmark definition that encompasses elements such as data select, the design of workload suites, and metric proposals. The proposed framework features a messaging system and seven benchmark programs, all while considering fault tolerance and durability. It provides four distinct workload suites, each tailored to different aspects of these systems. The benchmark results are showcased for Spark Streaming, Storm, and various Storm versions, culminating in a summary and forward-looking suggestions

Here are a few potential limitations to consider : *Limited workload coverage* : Many benchmarking tools are developed to test a particular type of workload or system configuration, which may not be representative of the user’s actual workload or system configuration. This can result in less precise evaluations of a system’s performance. *Difficulty in configuring and using* : Some benchmarking tools can be difficult to configure and use, which may discourage users from using them or lead to inaccurate benchmarking results if the tool is not configured correctly. *Limited input data* : Many benchmarking tools use static input data, which may not accurately represent the dynamic and changing nature of real-world data. This could result in misleading performance evaluations if the input data characteristics are significantly different from those of the user’s actual data.

Table 2.2 provides a comparison between our framework, SparkPerf, and other existing platforms.

In conclusion, while benchmarking tools have some benefits for evaluating the performance of different systems, there are several limitations to consider when selecting a platform. These

TABLE 2.1 Overview of the related work on performance modeling with respect to the developed, type of workload, set of input variables, targeted output variables, and whether a work has considered feature importance (The key difference of our work is utilizing response time and throughput metrics to capture the combined impact of transactional workload generators, Spark settings, and the underlying infrastructure simultaneously, therefore, the key difference lies in our case study.

Prior works	Models	Type of Workload		Input variables					Output variable(s)	Feature Importance
		Batch	Transactional	users	Size of data	Executor config	Spark workers	VMs		
Javaid [25]	Linear regression (LR), Random forest, Gradient Boost Machine, Feedforward neural network	✓			✓				Runtime	✓
Singhal [44]	Multi linear regression, SVM	✓			✓		✓		Runtime	
Luo [45]	SVM, Artificial neural network	✓			✓				Runtime	
Maros [46]	Linear regression, Random forest, Decision tree	✓			✓		✓		Runtime	
Oliveira [47]	Decision tree	✓				✓			Runtime	✓
Myung [48]	Convolutional neural	✓			✓				Runtime	
Myung [49]	Deep learning	✓			✓				Memory usage	
Cheng [50]	AdaBoost	✓				✓			Runtime	
SparkSim	Linear regression, Random forest, Decision tree, AdaBoost, XGBoost, LSTM and MLP		✓	✓	✓		✓	✓	Response time and throughput	✓

limitations include limited workload coverage, difficulty in configuring and using, resource-intensive requirements, and limited input data. It is of great importance to carefully take into account the aforementioned limitations and select a benchmarking tool that is appropriate for the specific needs and use cases of the user.

2.2.3 Benchmarking Machine learning models

Given the recent rise in the popularity of machine learning and the integration of corresponding applications in larger systems, there has been a lot of focus on practitioners and researchers on the development and deployment of ML applications. AutoML [51] refers to a collection of various platforms that enable the automation of several steps and phases for the training, testing, validation, and final deployment of machine learning models in final applications. At the same time, AutoML tools can be potentially used for benchmarking the performance of the models, but this is not their primary focus as they put more emphasis on the accuracy of the model itself and less so on its runtime performance and resource consumption.

For example, the MLPerf project [52] provides a standardized set of benchmarks for training and inference tasks, covering a broad spectrum of platforms and systems. The benchmark [52], focuses on evaluating the performance of specific machine-learning algorithms or libraries in different environments.

This paper [53] evaluates the performance of eleven machine learning methods in predicting the performance of four scientific applications. The study concludes that machine learning methods can model the performance of scientific applications precisely, but the selection of the approach relies on the particular application and the characteristics of the platform.

The AMLB AutoML Benchmark [54] provides an extensive examination of 9 well-known AutoML frameworks, examining their performance in different classification and regression tasks. The intention of the work is to prevent common errors encountered when comparing AutoML frameworks and employ a comprehensive analysis to assess the distinctions between these frameworks. The findings of the comparison are presented for each task, and the respective advantages and disadvantages of each framework are discussed.

The paper [55] provides a comprehensive survey and benchmark of popular AutoML frameworks using real-world datasets. It covers the entire process of synthesizing an automatic ML pipeline, including pipeline structure determination, algorithm selection, and fine-tuning. Additionally, it conducts an extensive evaluation of six AutoML frameworks.

SparkPerf is designed to address several limitations that exist in current benchmarking tools

for machine learning model training in Spark. These limitations primarily revolve around the scope of benchmarking, the lack of customization options, including only batch workload generators, and the difficulty in deploying and running benchmarking experiments. SparkPerf aims to offer a comprehensive, customizable, and easy-to-deploy platform for benchmarking machine learning model training in Spark. By expanding the scope of benchmarking, providing customization options, and streamlining the deployment process, SparkPerf empowers researchers and practitioners to effectively evaluate and optimize the performance and efficiency of Spark-based ML pipelines.

2.3 Related work on performance modeling of big data analytics

Hernandez et.al [56] have built some regression models for the estimation of the completion time of Spark workloads using a number of the system- and application-level metrics. The features employed are specifically connected to parallelism, like executor cores, the number of jobs, and CPU load. However, the models were trained to maximize one goal, which was local optimal performance.

Wu et al. [57] introduced a logistic regression statistical model to make a prediction of performance metrics regarding the execution time of a single-threaded job. However, their proposed model did not factor in the impact of parallel jobs.

Ipek et al. [58] built a deep learning model to have a prediction of the running time of an application using the software and hardware features. The issue though with neural networks is that they cannot be interpreted. This is incompatible with our goal of aiding users while using big data platforms.

A machine-learning-based performance model was proposed by Javaid et al. [25] in which varying data volume, a range of Spark configurations, and different sets of features for training the model. Because these techniques include a wide parameter space, they necessitate a significant amount of previously collected execution data to be fully beneficial. However, acquiring such information may not always be possible.

Cheng et al. [50] presented a performance model based on machine learning models specifically Adaboost methods for Apache Spark, to predict the application’s execution times based on input configurations.

The Random-forest method for Hadoop is proposed by Bei et al. [59] to modify the parameters of Hadoop tasks that are executed repeatedly and/or have a long duration. For each MapReduce task phase, RFHOC uses a random forest approach to train an ensemble model. Each model takes a number of Hadoop parameters as input, and its result is a performance

prediction.

Ahmed et.al [60] explored the application of both machine learning approaches as well as analytical models in predicting the performance of big data projects. Using the interpolation and extrapolation method along with cross-validation, they thoroughly analyzed the performance of analytical models and machine learning regressors. They found out that the analytical model can make accurate predictions regarding runtime even beyond the range of the available training data.

Didona et al [61] introduced a hybrid approach with a combination of analytical models, and ML models to predict future workloads. The objective is to minimize the training time of machine learning models while enhancing the accuracy of the analytical models. The authors integrate these models using techniques such as kNN. They evaluated the performance of the process using a platform comprising two open-source software.

In summary, there are a few key differences between our research and existing studies. Firstly, we analyzed a different type of workload and looked at a more diverse range of critical parameters. Additionally, we used a distinct statistical analysis method to select the best models. Our workload generator is an open transactional workload, which means that requests are sent continuously, as opposed to batch workloads like machine learning and analytical queries. This makes our approach more reflective of a real-world environment. We also took into account several parameters such as data size, workload size, Spark topologies, and underlying infrastructure, which we consider to be crucial in understanding performance. Table 2.1 displays a comparison between our work and existing models. To save space on the page, we have abbreviated the names of some of the machine learning models we used, including Linear Regression (LR), Random Forest (RF), Decision Tree (DT), and Gradient Boost Machine (GBM).

TABLE 2.2 Comparison of benchmarking techniques with SparkPerf.

Benchmark		Hib-ench	cloud-suite	Bigdata Bench	Link Bench	Stream Bench	Spark Bench	Big Bench	Grid Mix	SparkPerf
Workload	Batch	✓	✓	✓	✓	✓	✓	✓	✓	
	Trans- actional									✓
Ability to run user's data										✓
Ability to run user's application										✓
Ability to generate synthetic data										✓
Config. options	VMs									✓
	Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓
	# of users									✓
	CPU cores	✓								✓
	# workers									✓
Operation	Machine learning						✓	✓	✓	✓
	Statistic	✓		✓			✓	✓		
	database		✓		✓					

3 SPARKPERF : MACHINE LEARNING BENCHMARKING FRAMEWORK FOR SPARK-BASED DATA SCIENCE PROJECTS

3.1 Preamble

The source code of this chapter can be found on the GitHub repository :

<https://github.com/Soude640/Benchmarking>

3.2 Introduction

Given the complexity and scale of modern data science projects, along with the exponential growth of data, it is essential to evaluate the performance and efficiency of machine learning pipelines. Benchmarking plays a critical role in this task due to the vast array of methods, tools, platforms, and architectures available [6], [8]. It allows organizations to meticulously compare and evaluate the available options and identify the ones that deliver superior performance and cost-efficiency. Additionally, benchmarking streamlines the rapid collection of data, enabling prompt comparisons and well-timed decision-making. Moreover, it supports thorough planning for data analytics by assisting organizations in comprehending resource requirements, scheduling analytics tasks, and optimizing budget allocation. By harnessing the power of benchmarking, organizations can make well-informed decisions, optimize the utilization of resources, and achieve highly efficient and effective outcomes in their data analytics endeavors.

However, the current benchmarks available for big data analytics [5–8, 54] focus either on the characteristics of the development or deployment platform, like Hadoop or Hive or on simplistic case studies aiming at giving a general idea about the efficiency and scalability of the platform. More specifically, these benchmarks primarily focus on “batch workloads”, where the entire workload is executed until completion or failure [1], without considering long-term scalability, reliability, and sustainability when dealing with continuous and fluctuating workloads. On the other hand, a transactional workload refers to the execution of multiple small work units, usually emitted by individual and independent simulated users, in an iterative manner throughout the duration of a benchmark [1]. This type of workload creates a longitudinal execution trace, which may correspond closer to the functions of a team or a company, and can capture saturation events or bottlenecks caused by prolonged usage, not evident in the case of batch workloads. In other words, batch workloads are more appropriate in spike or load testing, while transactional workloads may be more appropriate in stress or

endurance testing.

Besides the type of workload, SparkPerf encompasses several key quality properties that are highly valuable for both researchers and practitioners. The most important properties are customizability and flexibility, as SparkPerf allows users to test any application on Spark, using either their own data or sample data provided by the framework. Users have the flexibility to define and customize their workloads according to their specific requirements. This flexibility enables a wide range of testing scenarios, empowering users to evaluate the performance of their applications under different conditions and configurations. The use of Docker containers to deploy a benchmarking experiment streamlines the setup and execution process, allowing for fast and cost-effective testing. This capability enables users to test multiple conditions and gather more information to make optimal planning decisions.

In summary, our work makes the following contributions :

- **A machine learning benchmarking framework :** The machine learning benchmarking framework offers an environment for evaluating Spark-based data science projects.
- The framework includes an open transactional workload generator that allows users to customize and test various workloads, effectively benchmarking machine learning algorithms utilizing the Spark infrastructure.
- Another contribution is the development of workload mixes for system performance evaluation. It involves creating a set of tasks or activities that simulate real-world scenarios and represent typical usage patterns for a given system, or application.
- Customizability and flexibility are the key properties of our benchmarking framework. The framework empowers users to test any application on Spark, allowing them to use their own data or sample data provided by SparkPerf. Additionally, users have the freedom to define and customize their workloads according to their specific requirements. By offering such customizability and flexibility, SparkPerf provides users with valuable insights into the performance of their applications on Spark and supports efficient decision-making in data science projects.
- An additional property of the benchmark is its utilization of Docker containers, which greatly simplifies the deployment and execution of benchmarking experiments. Leveraging Docker containers enables fast and cost-effective testing, as it streamlines the setup process and eliminates the need for complex configuration. This capability facilitates the testing of multiple conditions and provides researchers and practitioners with abundant information to make optimal planning decisions.

3.3 The SparkPerf benchmarking framework

SparkPerf is a benchmark for ML applications developed and deployed with the Apache Spark engine. The main focus of the platform is increased usability and customizability. At its core, it allows testers, experimenters, and researchers to set up and execute benchmarking experiments with ease and speed. The user can pick existing sample codes for popular ML models and processes with sample data or upload their own code with their own data. Most importantly, the benchmark supports longitudinal experiments under open transactional workloads that simulate a continuously varying number of users submitting the ML jobs to Spark for training and/or inference. This supports a large set of realistic case studies relevant to large teams or companies that consist of a number of software and data engineers and data scientists working on different projects. In this section, we outline the basic architecture and implementation of the benchmark, along with its main functional and non-functional features

3.3.1 Functional and Non-Functional Requirements

Functional requirements pertain to the precise capabilities and features essential for a benchmark tool to fulfill its intended purpose, while non-functional requirements refer to the performance and usability characteristics that a benchmark tool must exhibit.

In this section, we discuss the essential requirements of a Big Data or ML benchmark and highlight the additional capabilities and contributions of our benchmark tool beyond these basic needs.

A Big Data or ML benchmark should meet certain fundamental requirements to effectively evaluate and compare different systems and platforms. One crucial requirement is the ability to load user-specific data. This enables researchers and practitioners to assess the performance of their own datasets and evaluate how different systems handle their specific data characteristics. Our benchmark tool recognizes this requirement and provides users with the flexibility to load their own data, ensuring a more realistic and customizable benchmarking environment.

In addition to the basic requirement of loading user-specific data, our benchmark tool offers an additional contribution in the form of synthetic data generation. Synthetic data serves as a valuable asset in benchmarking scenarios, as it allows for controlled experimentation and the generation of diverse datasets that mimic real-world scenarios. In fact, synthetic data generation in a benchmark refers to the process of creating simulated data that mimics the attributes and distribution of real-world user data. This synthetic data is used to sup-

port scalability in the benchmark by providing additional samples beyond the available real data. By providing synthetic data generation, our benchmark tool enhances the versatility of benchmarking experiments, enabling users to evaluate system performance under a wide range of data conditions.

Another class is “RandomModel”. The RandomModel class is a model for generating new data by randomly sampling values from the unique data values in each column of a given DataFrame¹. This model generates each row by choosing a random sample of unique data values for each column. It takes an optional count parameter that specifies the number of rows to generate.

By addressing the fundamental requirement of loading user-specific data and introducing the capability of synthetic data generation, our benchmark tool goes beyond the traditional scope of a basic Big Data or ML benchmark. These additional features empower researchers and practitioners to conduct more comprehensive and realistic benchmarking experiments, leading to better insights and informed decision-making in the evaluation of different systems and platforms. Therefore, in summary, here are some examples of functional and non-functional requirements for the SparkPerf benchmark tool :

Functional Requirements :

— **Load custom code and data :**

One crucial functional requirement is the ability to load user-specific code and data. This enables researchers and practitioners to assess the performance of their own implementations on their own datasets and evaluate how different algorithms or configurations handle their specific data characteristics. In SparkPerf, the user can upload their own code for Spark with its corresponding data, increasing the platform’s customizability. SparkPerf would then automatically deploy the application and its data on its own configurable infrastructure and execute performance experiments for the particular application creating a more personalized experience and results for the user.

— **Support for custom workload :**

SparkPerf workloads are configurable in a variety of ways : from the size of the data to the range within the number of users allowed to fluctuate during the experiment to the mix of ML jobs submitted to Spark. For example, a user can configure that 5-10 users would be submitting files of 100KB to 100MB for three different Spark jobs. This functionality ensures that the benchmarking process captures the complexity and variability of real-world usage scenarios, enhancing the tool’s relevance and applicability

1. <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/>

in practical settings.

— **Support for custom workload :**

Extensibility is one of the basic requirements for benchmarking frameworks. While the control flow and the main benchmarking functionality (i.e., running the experiment) do not need to change, the user should still be able to extend or customize certain parts of the experiment. In SparkPerf, this is achieved in two ways : a) effortlessly incorporate new ML tasks/models for benchmarking, and b) seamlessly introduce new datasets/workloads to conduct experiments. Indeed, our system provides those interfaces or configurations to allow for changing components mainly thanks to the use of Docker. The benchmarked application is deployed in Docker containers and then it can be executed, stressed, and monitored accordingly.

Non-Functional Requirements :

— **Extensibility**

It implies the ability to extend or enhance the system’s functionality by incorporating new components, features, or resources without significant effort or disruption to the existing structure. Therefore, extensibility encompasses the ability to : a) effortlessly incorporate new ML tasks/models for benchmarking, and b) seamlessly introduce new datasets/workloads to conduct experiments. Indeed, our system provides those interfaces or configurations to allow for changing components without a problem. A well-designed system, such as Docker, with appropriate interfaces and configurations, allows for seamless integration and interchangeability of components. Thus, Docker provides a containerization platform that enables us to encapsulate our application and its dependencies known as containers.

— **Usability :**

The usability of big data benchmarks is evaluated based on various characteristics that determine how users experience them. These characteristics encompass a range of factors, including the ease of deploying, configuring, and using the benchmarks, how effectively they benchmark performance, the clarity, and brevity of their performance indicators, the convenient user interfaces; and other important considerations. Ultimately, all of these elements factor into users’ perceptions of how useful and accessible these benchmarks are for their needs. Once again, Docker is the key component to offer increased usability through easy and fast deployment and configurability.

— **Reliability :**

In benchmarking, reliability refers to the ability of the benchmark to produce results in a deterministic manner. In other words, this means that when the conditions or the

configuration of an experiment remain constant, the results of the experiment should remain largely unchanged no matter how many times it is repeated. SparkPerf executes benchmarking experiments in relative isolation, guaranteeing that their results are not affected by extraneous conditions outside of the experimenters' control. This is especially important when results need to be validated statistically, where multiple iterations are necessary.

— **Portability :**

Portability typically refers to the ability of software or a system to be easily moved or transferred between different environments or platforms, such as from a laptop to a server. In the context of benchmarking platforms, portability would indeed involve the capability to run the platform on various machines or systems, including both laptops and servers. Running all components of SparkPerf as containers allows us to deploy benchmarking experiments on any platform, infrastructure, or operating system. This enables researchers and practitioners to conduct benchmarking experiments across different systems, enabling comparisons and evaluations across diverse computing environments. One popular case study in data science and ML is to see whether it is more efficient or cost-effective to run model training and inference tasks on local “on-prem” resources or on the cloud.

3.3.2 Components

The SparkPerf benchmarking platform is designed with multiple tiers in its architecture. As can be seen in Fig. 1, the first layer is the workload generator, which has the responsibility of sending requests to the Spark cluster. The second tier is the Django Server. This tier provides the user interface and configuration information. It provides users with options to configure the workload characteristics, and infrastructure configuration, as well as configuring Spark cluster settings and view performance metrics. Another tier is Flask API : Acting as an intermediary between the Django server and Spark, the Flask API facilitates communication and data transfer between these components. It leverages the computing resources available in the cluster to efficiently execute tasks generate performance metrics, and store them in the Prometheus database. Prometheus² is a monitoring tool used to collect and store performance metrics. It gathers data from various components, including the workload generator and Spark cluster, to provide insights into the system's performance.

The results are then displayed in a customizable Grafana dashboard, enabling users to visually analyze and compare performance across different Spark configurations, different infrastruc-

2. <https://prometheus.io/>

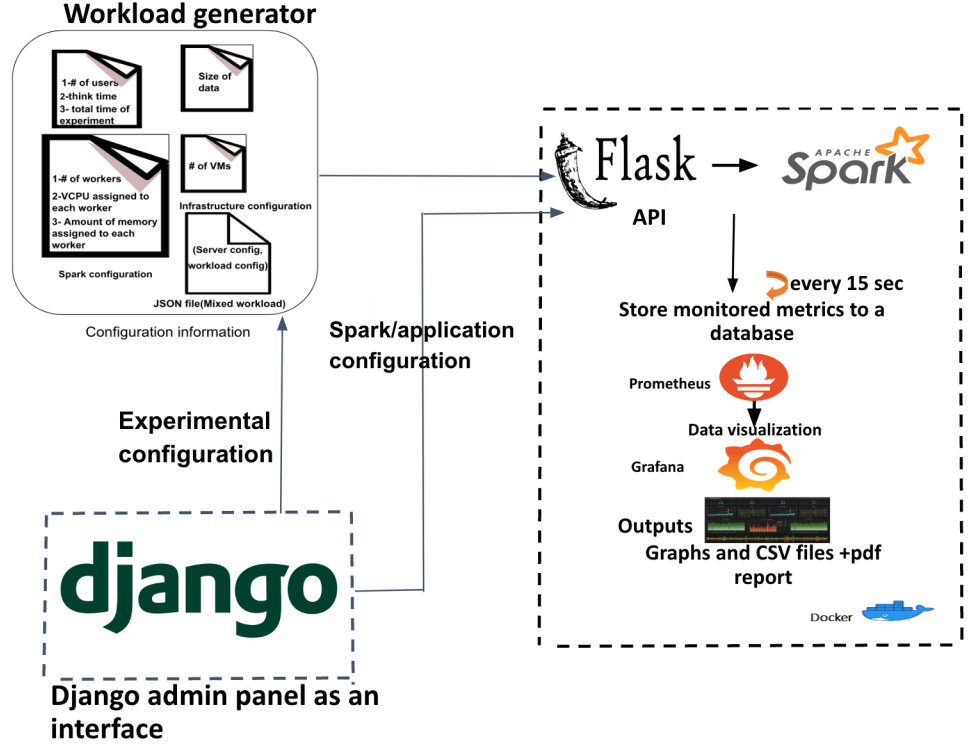


FIGURE 3.1 The SparkPerf component structure

ture configurations as well and different workload characteristics. In fact, Grafana³ is a data visualization platform that creates customized dashboards for displaying performance metrics. It retrieves data from Prometheus and presents it in a visually appealing and easily understandable format. Thanks to the use of containers, deploying the components is quite straightforward. The use of containers greatly simplifies the deployment of components, making it an effortless process. Indeed, the architecture involves a series of components that work together to generate a workload, process it in a distributed manner, and collect performance metrics. The Django server provides the user interface and configuration information, Flask API acts as an intermediary between the Django server and Spark, and then the image is built, and all the components including Flask, Spark, Prometheus, and Grafana, are up and running with the use of Docker compose and then the workload generator will be run. Eventually, all the performance metrics will be monitored by Prometheus and shown on Grafana's customized dashboard. We elaborate on each component in the following subsections.

3. <https://grafana.com/>

User interface

The Django server is used as a user interface to provide configuration information for the benchmarking experiment. In the Django admin panel, the user can specify the various parameters of the workload, load a custom application and/or dataset, pick from the preset ones, and configure the experiment infrastructure (Figure 3.2). Next, we present in detail what the user can configure through Django. The Django serves as an interface for the user to specify the benchmarking scenario, such as the number of users, the types of requests to generate, and other parameters. We created a Django admin panel, which is an automatic admin interface and can read configuration information from our model to provide a quick interface. The Django admin panel is a built-in web-based interface that allows us to manage our framework⁴. It provides a user-friendly interface for providing configuration information. In the Django admin panel, the user can specify the various parameters of the workload, load a custom application and/or dataset, pick from the preset ones, and configure the experiment infrastructure (Figure 3.2). Next, we present in detail what the user can configure through Django.

labs In the context of SparkPerf, a benchmarking experiment is simply called a “lab”. In Django, the user can configure labs, through the corresponding section, by conveniently providing input in a series of forms. The parameters that can be configured for each lab are described next.

Number of users :

In the context of a transactional workload, the number of users refers to the number of simulated users that are sending requests to a Spark application, and their interactions with the system are used to generate a load on the system. Users are active in the system for about a specified time interval, which can be specified on another unit(time of activeness of each user), and are allowed to send another request when they get the response to their previous request. The experimenter provides a range of users and the benchmark will vary the number of users within this range to simulate a continuous and fluctuating workload. According to the objectives of SparkPerf, the users correspond to developers or data scientists who use a shared Spark infrastructure concurrently for different applications with different intensities. This allows for planning at a global level and not per application or per project.

Size of data :

In the context of benchmarking Apache Spark, the size of the data is an important confi-

4. <https://www.djangoproject.com/>

Django administration

WELCOME, QARI. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home » Lab » Labs

Start typing to filter...

APP

Applications [+ Add](#)

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

LAB

Datas [+ Add](#)

Labs [+ Add](#)

MIXED_LAB

Mixed labs [+ Add](#)

Select lab to change

ADD LAB +

Search

Action: [Go](#) 0 of 9 selected

ID	NAME	IS ACTIVE	SERVICES COUNT	APPLICATION SPARK WORKERS COUNT	WORK LOAD USERS COUNT	WORK LOAD DATA SIZES	WORK LOAD USERS DURATION	WI
10	server_3	✓	1	2,2	5,10	mlp_100MB.csv	900,600	10
9	server_2	✓	1	2,2	5,10	Iris_gRMFfNF.csv	900,900	50
8	server	✓	1	4,4	5,10	Iris_gRMFfNF.csv	900,900	50
7	mlp_custom	✓	1	4,4	5,10	mlp_8KB.csv	900,600	50
6	decision_tree_custom	✓	1	3,3	10,5	Iris.csv	900,600	50
5	XGBoost_classifier_custom	✓	1	3,3	10,5	Iris.csv	900,600	50
3	clustering	✓	1	3,3	10,5	dummy	300,900	1
2	xgboost	✓	1	2,3,4,2	5,10,3,5	g1.csv	300,600,300,600	1
1	mlp_classifier	✓	1	2,3,4,2	5,10,3,5	mlp_8KB.csv	300,600,300,600	1

9 labs

FIGURE 3.2 Django admin panel as an interface

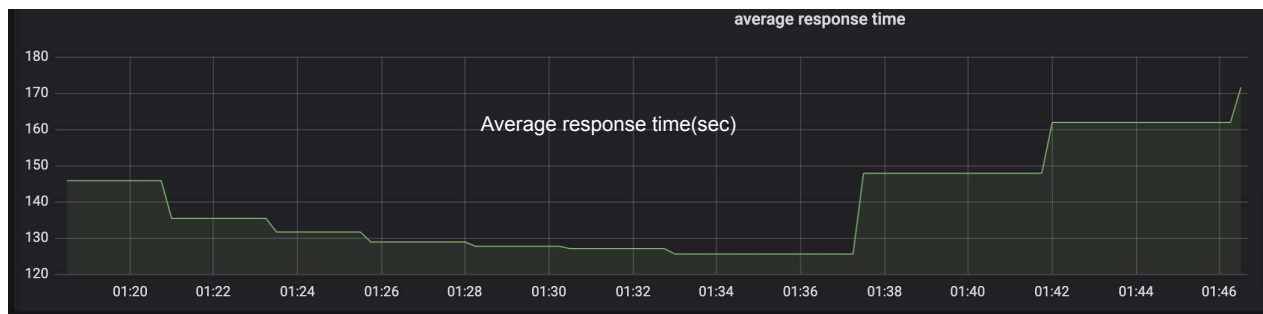


FIGURE 3.3 A view of Spark Grafana Dashboard for the response time metric

guration parameter that can affect the performance of the Spark cluster. Data size refers to the volume of data that is used in the benchmarking process, and it is typically specified in terms of the overall size of the input dataset or the size of individual partitions within the dataset. Larger data sets can help identify performance bottlenecks that may not be visible with smaller data sets, and they can also help measure the scalability of Spark’s processing capabilities as the size of the cluster is increased.

Number of VMs :

This parameter configures the size of the infrastructure in terms of the number of virtual machines (VMs) used to host the Spark infrastructure. Once again, this parameter can be used to evaluate the scalability and the cost of the underlying infrastructure. The exact value of this parameter depends on the number of VMs dedicated by the experimenter to SparkPerf, but in our prototype implementation, the maximum number is 2. Thus, the user can run their experiment on one or two virtual machines on the cloud. For one server, the user needs to identify a fixed number of workers. When the user identifies two servers to run their experiment, a 2-node Docker Swarm cluster is created.

Total time of experiment :

Given that SparkPerf supports open transactional workload, the workload is continuously sent to the Spark application and a time-based condition is necessary to terminate the benchmarking experiment. This condition is set by the experimenter as the total time of the experiment.

Think time of each user :

This parameter is necessary to simulate an interactive system. The premise is that a user (e.g., a data scientist) would submit an ML application to Spark. Once Spark responds with the results and potentially the performance of the model, the data scientist may want to go back and change the model to increase its performance. The time it takes for the data scientist to process the application’s response and submit a new job request is considered the “think time” of the user. This parameter is used to increase the realism of the simulated users.

Infrastructure configuration :

Here the experimenter can configure Spark in terms of the number of CPU cores and memory allocated to each worker. They can also specify the number of Spark workers, which remains fixed for the duration of the experiment.

Data With respect to data, SparkPerf offers three options : i) load custom user data, ii) use preset datasets, and iii) generate a synthetic dataset.

In the first option, SparkPerf allows experimenters to load their own dataset, under the premise that this can enable data-specific conclusions. The specific format or complexity of data may create special conditions with respect to performance and resource consumption. However, in this case, the experimenter is responsible for adapting the benchmarked application to correctly load and use the custom dataset. Currently, SparkPerf does not offer the automatic load of the custom dataset in any application.

In the second option, the experimenter can use a preset dataset. SparkPerf currently supports two datasets popularly used in ML applications. The first dataset is the Yelp dataset, sourced from Kaggle⁵, which holds substantial relevance for evaluating machine learning algorithms in the domain of sentiment analysis and recommendation systems. The dataset contains mostly textual semi-structured datasets (5.2 million user reviews for 174,000 businesses in 11 metropolitan areas) and it is appropriate for natural language processing or classification models. The second dataset is the MNIST handwritten digit database [62]. The MNIST handwritten digit database is a dataset that is extensively utilized in the domain of machine learning and computer vision. It is a collection of grayscale images of handwritten digits (0-9) that are 28x28 pixels in size. The dataset consists of two parts : a training set comprising 60,000 images, and a test set containing 10,000 images [62]. The images are accompanied by corresponding labels indicating the digit depicted in each image [62]. This dataset serves as a benchmark for developing and evaluating algorithms that classify and recognize handwritten digits. Compared to loading a custom dataset, this option offers the possibility to compare against other models and applications in literature and in practice that have been trained or tested in these datasets.

In addition to providing predefined datasets like the MNIST dataset and Yelp dataset, the SparkPerf benchmarking platform also offers the capability to load and utilize user-specific datasets. This feature allows users to work with their own data. Thus, by supporting the loading of user datasets, the SparkPerf platform offers a high level of flexibility and customizability. Users can upload their data to the platform, ensuring that the benchmarking experiments are conducted on the specific datasets relevant to their research questions and objectives. In other words, the ability to load user datasets expands the platform’s applicability beyond predefined datasets, empowering researchers, data scientists, and practitioners to evaluate and benchmark their algorithms and models by applying real-world dataset that is specific to their use cases.

5. <https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset/versions/6>

Application With respect to the application, the experimenter has two options. In the first option, the experimenter can select predefined applications. Currently, SparkPerf offers a few custom implementations of popular ML jobs, including a feature selector, a neural network classifier, and an XGBoost classifier. All jobs are implemented in Python. Alternatively, the platform allows the experimenter to pick from a larger collection of implemented jobs from the Spark repository, including logistic regression, K-means, and Decision tree classification tasks⁶.

In the second option, the experimenter can upload their own code. Compared to the previous option, the experimenter is responsible for the proper implementation of their application, which needs to be a proper Spark job that can run on a regular Spark cluster. SparkPerf performs no validation of the source code or the proper execution of the job and as a consequence exceptions may be thrown.

Workload mixes In the main scenario we have assumed for SparkPerf, multiple data scientists or developers would be using a common Spark infrastructure to train or test their ML applications. It is naturally expected that all these users would submit different jobs, both in terms of the ML model and in terms of data. To better capture this variability, SparkPerf gives the possibility of defining workload mixes.

In SparkPerf, experiments involving workload mixes are considered a special class, called “mixed labs”. Mixed labs are configured similarly to regular labs with one exception. Instead of one application and one dataset, the experimenter defines a set of applications and a set of datasets. Then, they also define a distribution over the applications and over the datasets. For example, the experimenter can specify an XGBoost classifier, a K-means clustering, and an APRIORI algorithm with a distribution of [50, 30, 20], meaning that 50% of the active users will be submitting classification jobs, 30% clustering jobs and 20% association rule jobs. Similarly, the user can define datasets of 100MB and 400MB with a distribution of [60,40], meaning that 60% of the users will use data files of 100MB and 40% of the users will use files of 400MB. After validating the configuration, the framework orchestrates the execution of the workload mix within the benchmarking platform. It coordinates the necessary resources, initiates the required processes, and manages the overall workflow to ensure smooth and efficient execution.

After activating the workload__mixes experiment, a message will be shown on the screen, that in order to see the result, you need to go to the provided link, which is the Grafana dashboard for Spark, Docker cluster, and Docker host. The link is port :3000 of Grafana. Port 3000 is

6. <https://github.com/apache/spark/tree/master/examples/src/main/python/mllib>

the default port used by Grafana for its web interface. When we run the Grafana server on our local machine or on a server, it starts listening to port 3000 by default. We can access the Grafana web interface by typing the URL “http ://localhost :3000” in our web browser. If we are accessing Grafana on a remote server, we can replace “localhost” with the IP address or hostname of the server. Additionally, port 8000 can be used in a Spark cluster depending on the configuration and usage. In our case, it may be used for Spark’s web UI that provides a dashboard for monitoring Spark application and cluster status. The Spark web UI can be accessed by pointing a web browser to the IP address or hostname of the Spark master node followed by port 8000 (e.g., http ://spark-master-ip :8000).

Workload Generator

The Workload Generator is a key component of the benchmarking platform that is responsible for generating requests. It allows users to simulate different types of workloads and measure the performance of the system under varying conditions. We used a transactional workload generator to generate synthetic transactional workloads for testing and benchmarking distributed systems. Regarding a big data analytics platform particularly Spark in our case, a benchmark workload simulates a number of users with the purpose of assessing the performance of Spark clusters and applications across diverse conditions, such as different infrastructure configurations, Spark topologies, and data sizes.

After activating the Django form, the configuration information is sent to the workload generator by the Django server. Subsequently, the Django server passes the configuration file to the workload generator. Our Flask application retrieves its Spark/application configuration from Django, including the memory and the number of CPU cores of its workers from Django. Therefore, flask obtains the configuration information from the Django server and converts the applications, a user has built on the Django server into APIs. These APIs are used by Spark to create sessions for the applications. Eventually, the workload generator sends requests to the API created by Flask to start generating load.

Additionally, when we deal with a mixed workload, the input of the workload generator that Django sends to it is different from a standard experiment. It comes in the form of a JSON file, which users create through the Django Framework API.

The frequency of user count changes can be adjusted to align with specific workload scenarios. For instance, it can be configured to follow a pattern such as 5, 10, or 15. This pattern signifies that initially, the number of users starts at 5, then increases to 10, and finally reaches 15. By customizing the workload generator in this manner, it becomes possible to mimic the fluctuations in user activity that occur in real-world scenarios.

Additionally, the workload generator allows for fine-grained control over the rate at which each user sends requests. This is achieved through two mechanisms : specifying the active time duration for users within the system and incorporating think times between consecutive requests. The workload generator allows for the customization of user count changes to simulate various user concurrency levels.

Building Container image

After the Django form is activated, the process starts. First, the provided configuration information will be forwarded to the Flask API. When the user triggers the Activate button, the Django Admin Panel activates the Flask API to handle the request. In this process, Flask is used to handle the request received from the Django form. It receives configuration information from the Django server and forwards it to the other components for further processing. The Transaction Workload Generator receives the specifications for the transaction workload and sends them to the Spark cluster for processing. Then, the resulting performance metrics are sent to Prometheus and Grafana for visualization.

When the lab is activated, the provided app file will be copied to the right place with the Flask app, and then the image will be built. Then Docker Compose will run Flask, Spark, Prometheus, and Grafana. Then the script of the workload generator will be run. To run user applications, the user needs to implement the application in Python format and follow the requirements of our application.

Data visualization

The last component is data visualization including Prometheus and Grafana.

Prometheus, being an open-source monitoring system, is capable of retrieving metrics from multiple sources and persisting them in a dedicated database that organizes the data based on timestamps, known as a time-series database, and facilitates querying and alerting. Its main functionality is to obtain the status of the monitored component at predetermined intervals.

Grafana⁷ is a popular open-source platform for monitoring and visualization of time-series data. It provides a variety of pre-built dashboards and plugins for monitoring various systems and applications. It empowers users to generate and visualize interactive and customizable dashboards, offering real-time insights into diverse data sources.

Typically, there are multiple steps involved in setting up Prometheus and Grafana, including

7. <https://grafana.com/>

setting up and configuring the Prometheus server, configuring data sources and dashboards in Grafana, and installing and configuring any necessary integrations. In Grafana, every dashboard has a corresponding JSON configuration file within our project, allowing us to add any type of dashboard we need. By default, Grafana is able to read configuration files and display the dashboards that have been added there. It is worth mentioning that monitoring is done every 15 seconds. Indeed, Prometheus sends its data every 10 seconds, but we have configured it every 15 seconds. Hence, we define the scrape interval (Prometheus.yaml) to 15 seconds from the configuration file (Prometheus.yaml). The file used by the Prometheus server to specify how to collect metrics from various sources can define the targets that Prometheus should scrape, along with any relevant metadata. The metrics can be updated more frequently with a shorter scrape interval, resulting in more precise and current information about the system being monitored. However, more frequent scraping can result in more network traffic produced by Prometheus and put more strain on the system being monitored. Thus, a scrape interval of 15 s is a reasonable trade-off between data accuracy and performance for the system. Grafana is an open-source visualization platform⁸ that enables the creation of dashboards and graphs to display the data collected by Prometheus.

The workload generator would then need to be configured to extract performance metrics to Prometheus at regular intervals. Once the metrics are collected by Prometheus, they can be visualized in Grafana using pre-built dashboards or custom dashboards created by the user. This allows for real-time monitoring of the Spark cluster and applications. By monitoring performance metrics with Prometheus and Grafana, we can identify issues in real time, and take proactive measures to optimize the Spark cluster and applications for maximum performance. This can lead to improved query performance, reduced resource utilization, and more efficient use of computing resources. Hence, integrating a transactional workload generator with monitoring tools like Prometheus and Grafana can provide useful insights into the performance of Spark applications, and help us optimize our system for maximum efficiency. The link to our repository for the source code of the benchmarking platform is shared⁹. We created three dashboards for interpreting results including :

- 1- Spark
- 2- Docker container
- 3- Docker host

Figure 3.3 demonstrates the screenshot of the dashboard for 147 MB of data. Grafana provides various views and components to help users analyze and present data in meaningful

8. <https://grafana.com/>

9. <https://github.com/Soude640/Benchmarking>

ways. Here is the description of the Grafana dashboard view :

Dashboard View : The dashboard view provides an overview of multiple visualizations and panels on a single page. It allows us to arrange and organize different graphs, tables, and other components in a layout that suits our needs. Dashboards can be customized with various templates, annotations, and time range selectors. The flexibility and extensibility of Grafana make it a versatile tool for visualizing and exploring data from various sources, enabling users to gain insights and make data-driven decisions.

Table 3.1 shows the list of metrics monitored by Prometheus described next :

1. *Number of users* : The number of active users fluctuates throughout the experiment. This number should be within the range specified by the experimenter during the configuration. This metric gives a high-level view of the load generated by the application.
2. *Number of workers* : The number of Spark workers that process the job requests submitted by the users. If the experimenter has specified so, this value may fluctuate to reflect the scalability of Spark.
3. *Average response time* : The average time taken by Spark to complete a job request of a user. This is the main performance metric indicating the efficiency of the application. In the case of mixed workloads, there is an average response time per application, but overall data files and data sizes are used for this application.
4. *Number/rate of successful requests* : This corresponds to the throughput of the application and it is the second important metric to determine the efficiency of the application. If there are no problems, e.g., bottlenecks, misconfiguration, or failures, the number of successful requests should be the same as the number of requests submitted.
5. *Number/rate of failed requests* : The number of failed requests may be an indication of a problem. Eventually, the number of failed requests plus the number of successful requests totals the number of sent requests, which is another metric for the load received by the application.

Besides metrics related to the execution of the experiment, Prometheus also gathers data about the provisioning and consumption of resources, such as CPU, memory, storage, and network bandwidth, to meet the requirements of applications or services running on a system. These metrics can be useful to improve the management of the resources or how these relate to the general performance of the system.

Data visualization and reporting :

The last component of SparkPerf is the visualization of monitoring data to aid in the final reporting of the executed experiments. Visualization is based on Prometheus¹⁰, an open-source instrumented platform for visualization of live monitoring data and the creation of dashboards for better control of the monitored systems. Once the metrics are collected by Prometheus, they can be visualized in Grafana using pre-built dashboards or custom dashboards created by the experimenter. The dashboards can be inspected while the experiment is running to identify problems or behavior patterns at runtime or after the experiment has ended for a post-mortem analysis. Through Grafana, we can export CSV files with the monitored data as well as snapshots of the dashboards as visualizations of the monitored traces (see Figure 3). In the current implementation, SparkPerf provides three custom dashboards corresponding to the three monitored layers of the system : i) Spark (and the deployed applications), ii) Docker container (for all containers deployed for the application), and iii) Docker host (for the VMs that host the Docker containers)

In summary, the primary objective of a big data analytics benchmarking platform is to facilitate developers (including researchers and practitioners) in conducting experiments to evaluate the performance of big data analytics systems and their different configurations. Additionally, it enables the comparison of these systems, aiding in the assessment of their relative capabilities. The fundamental workflow of using our benchmark tool is as follows

- The user has the flexibility to select the desired SparkPerf benchmarking platform and configure the infrastructure based on specific requirements. This includes specifying parameters such as the number of CPU cores, the total memory allocation, and workload settings, including the number of concurrent users, data size, and the number of workers to be employed.
- The workload generator produces a stream of users continuously.
- The Django server is responsible for sending Spark/application configuration to Flask and experimental configuration to the workload generator.
- The end-user has the ability to monitor the progress of the experiment through a real-time dashboard known as Grafana. This dashboard provides visualizations of the experiment’s performance information, which can also be exported in formats such as CSV. The dashboard actively tracks input-related metrics, including the number of transactions sent by the workload generator, and performance metrics from the Spark cluster, including successful requests and error rates.

system.

10. <https://prometheus.io/>

TABLE 3.1 Monitoring performance metrics and resource utilization

Monitored metrics relevant to performance and resource utilization		
System provisioning	Spark	#of users (pure number)
		Average response time(sec)
		#of successful requests (pure number)
		Rate of success requests (percentage)
		#of failed requests(pure number)
		Rate of failed requests(percentage)
		Rate of increase success response (percentage)
Resource provisioning	Docker container	Container CPU usage(percentage)
		Container memory usage(GB)
		Container cached memory usage(MB)
		Container network input(B)
		Container network output(kB)
		CPU load(percentage)
		CPU cores(pure number)
		Memory load(percentage)
		Used memory(GB)
		Storage load
		Used storage(GB)
	Docker host	CPU usage(percentage)
		Memory usage(GB)
		IO usage(Read :(Bs), Write(KBs))
		Network usage(Bs)
		swap usage
		Swap IO(Bs)
		Uptime(sec)
		CPU idle(percentage)
		CPU cores(pure number)
		Available memory
		Free swap
		Free storage(GB)

3.4 Case studies

In this section, we provide a demonstration and evaluation of how SparkPerf can potentially be used through a series of case studies, which is an established validation method in software engineering research [63]. More specifically, we contact 2 case studies to validate the portability, customizability, and extensibility properties of SparkPerf. In all case studies, we assume the role of a random experimenter, whose objective is to answer a research question pertaining to one of the non-functional requirements. In the context of each case study, we discuss how SparkPerf can be used to set up and run the benchmarking experiment and how the results output by SparkPerf can be interpreted to answer the research question.

- For portability, we aim to find the limit of the on-prem infrastructure and find it easy to install SparkPerf on any environment rather than a local or remote server.
- For customizability and extensibility, we aim to understand if SparkPerf’s components of the experiment can be extended without modification to the overall control flow and benchmarking functionality.

3.4.1 Experimental Setup

In all two case studies, we have deployed SparkPerf on an OpenStack environment on virtual machines running Ubuntu Server 22.04 with 64GB RAM, 16 vCPUS, and 20GB hard disk space. In the portability experiment, we had to explore a different deployment environment, so we also used a personal laptop running Ubuntu Desktop 20.04.5 LTS with 16GB RAM and 8 CPU cores.

In this section, we provide a demonstration and evaluation of how SparkPerf can potentially be used through a series of case studies, which is an established validation method in software engineering research [63]. More specifically, we contact 3 case studies to validate the portability, customizability, and extensibility properties of SparkPerf. In all case studies, we

TABLE 3.2 Characteristics of experiment 1 and 2 regarding reliability case study

Exp	# of users	# of workers	Size of data	Total time	# of VMs	#of CPU cores for each worker	Amount of memory of each worker	Application
1	5,10,3,5	2,3,4,2	1 G	40 min	1	4	14 GB	XGBoost classifier
2	5,15	4	147 MB	10 min	1	4	14 GB	Feature selection

TABLE 3.3 Characteristic of the experiment with workload mixes regarding reliability case

Server configuration				workload configuration											
# of VMs	# of workers	# of CPU	Memory of each worker	# of users	Total time	app names	Think time	user percentage	size of data			size of data percentage			type of data
2	8	4	10	20	3600	feature selection	30	60%	120 (K)	150 (M)	1 (G)	50%	30%	20%	regression
						MLP classifier	30	40%	8 (K)	100 (M)	-	70%	30%	-	Iris

TABLE 3.4 Measuring Average and Standard Deviation
Reliability of Experiment 1

Iter.	Avg response time(s)	Throughput (req/s)
1	478.554	12.61
2	478.653	12.61
3	478.832	12.61
4	478.912	12.61
5	478.354	12.61
6	478.871	12.61
7	478.682	12.61
8	478.453	12.61
9	478.876	12.61
10	478.763	12.61
Average	478.690	12.61
Standard deviation	0.187	0

TABLE 3.5 Measuring Average and Standard Deviation for Testing Reliability of Experiment 2

Iter.	Avg response time(s)	Throughput (req/s)
1	31.472	0.0698
2	31.689	0.0698
3	31.391	0.0698
4	31.775	0.0698
5	31.563	0.0698
6	31.559	0.0698
7	31.493	0.0698
8	31.684	0.0698
9	31.512	0.0698
10	31.421	0.0698
Average	31.555	0.0698
Standard deviation	0.0919	0

assume the role of a random experimenter, whose objective is to answer a research question pertaining to one of the non-functional requirements. In the context of each case study, we discuss how SparkPerf can be used to set up and run the benchmarking experiment and how the results output by SparkPerf can be interpreted to answer the research question.

3.4.2 Case study 1) Portability

Alex is a project manager for the data science team of a large insurance company. The company has two options in terms of infrastructure to run data science projects, including training and inference of ML models : a local “on-prem” infrastructure and a public cloud. Naturally, the on-prem infrastructure offers lower marginal costs, but also lower computational capacity. On the other hand, the public cloud offers much higher computational capacity for a higher cost. Therefore, Alex needs to address this research question :

What are the performance limitations of the on-prem infrastructure, at which point do we need to change to the public cloud ?

This research question epitomizes Alex’s academic pursuit to investigate the inherent limitations of his company’s on-premises infrastructure. By gauging the potential of cloud infrastructure as a means to overcome or compensate for these limitations, Alex aims to

explore the extent to which the performance boundaries of the local infrastructure can be extended.

1) Setup and initial configuration

In order to answer the research question, Alex plans on comparing various jobs and data sizes on the on-prem and public cloud infrastructures. His main objective is to understand underperforming or terminated experiments that can be effectively executed in the public cloud. Alex proceeds to easily install SparkPerf through docker-compose both on the on-prem server as well as on a remote VM on the public server. Then, he prepares the configurations presented in Table 3.6 through the Django interface. Two Spark jobs are compared, an MLP classifier and a Clustering algorithm, in pairs. In the first pair for MLP, a fairly large data size (400MB) is used and in the second pair, a smaller data size (100MB) is used. In the clustering example, the intent is to see the limits of the on-prem infrastructure, both in terms of users and data size, and if it can be surpassed by the remote VM.

2) Results

As can be seen in Table 3.6 during the execution of the MLP classifier, the on-prem server exhibits almost double the response time of the remote server. Depending on the potential cost, Alex may find this as acceptable or if it is cost-efficient he may choose to deploy this type of job on the public cloud. In addition, we can see that in the case of the clustering algorithm, the on-prem server exhibits comparable response time, but for fewer users and with much smaller data files. In fact, with a few more experiments, Alex found that the on-prem could not run the larger experiment at all, which led to the decision to deploy the job on the cloud.

3)Case study discussion

Overall, the case study demonstrated how easy it is to install SparkPerf on any environment with no other configuration required, besides the one for the experiments. This allows for extensive comparisons between different environments that cover a variety of research questions, one of which was demonstrated here.

3.4.3 Customizability and extensibility

Sara is a Ph.D. student and for her research, she has developed a custom implementation of the Random Forest classification algorithm. In order to evaluate the performance of her

TABLE 3.6 The experimental characteristics of on-prem and remote server setups along ge response time

Infrastructure	Experiment Characteristics							Average response time
	number of users	number of workers	Size of data	number of CPU cores (per workker)	Amount of memory (per worker)	Type of Spark's job	Think time	
On-prem	5-10	2	400 MB	3	4 G	MLP classifier	10s	820 s
Remote server	5-10	2	400 MB	4	8 G	MLP classifier	10s	435 s
On-prem	5-10	2	100 MB	3	4 G	MLP classifier	10s	265 s
Remote server	5-10	2	100 MB	4	8 G	MLP classifier	10s	144 s
On-prem	3-4	2	400 MB	3	4 G	Clustering	10s	484 s
Remote server	5-8	2	1 G	4	8 G	Clustering	10s	462 s

TABLE 3.7 Measuring Standard deviation for testing reliability of workload mixes for Average response time metric

Avg response time	Feature selection(app1)			MLP classifier(app2)	
	Average of 5 Iterations	120 KB	34.912	8 KB	48.527
		150 MB	36.928	100 MB	52.946
		1G	37.165		
	Average of 5 Iteration	120 KB	34.763	8 KB	48.469
		150 MB	36.794	100 MB	52.901
		1G	37.112		
Standard deviation 120 KB			0.0745	Standard deviation 8 KB	0.029
Standard deviation 150 MB			0.067	Standard deviation 100 MB	0.0225
Standard deviation 1 G			0.0265		

algorithm, she wants to compare it with an existing algorithm, XGBoost, as a baseline. One issue that Sara has to deal with is that while she has a dataset, it is not large enough to stress the system and evaluate its performance or its scalability. Eventually, Sara wants to answer the following research question :

RQ2 : *What is the performance of my custom Random forest classifier compared to the XG-Boost classifier ?*

1)Setup and Initial Configuration

Sara installs SparkPerf on a remote server according to the configuration shown in Table 3.9. Sara wants to test her custom algorithm using the Iris data available in the preset datasets of SparkPerf. However, the available data file is only 5KB, which is not suitable for performance and scalability testing. Therefore, Sara uses the SparkPerf data generation tool to increase the size from 5KB to 150MB.

2)Results

Sara uses the Grafana dashboard to monitor and analyze the experimental results. Using Grafana data she produces the graphs shown in Figure 3.4 to compare the performance of her custom XGBoost classifier algorithm against the Random forest classification algorithm in terms of response time and throughput. The results show that the Random Forest algorithm is more efficient both in terms of throughput and response time and it scales better as the number of users increases. Sara is able to use SparkPerf's Grafana module to export the graphs and publish them in her Ph.D. dissertation to support her research.

3)Case Study Discussion

Overall, the case study highlights how a user can seamlessly integrate a custom Spark ML application and generate synthetic data into the platform to facilitate further exploration concerning performance and scalability.

3.5 Reliability evaluation

Reliability in our context refers to the degree of consistency and dependability in the results obtained from our experiments. It assesses whether our benchmarking framework, SparkPerf, consistently produces similar outcomes across multiple runs of the same experiment.

TABLE 3.8 Measuring Standard deviation for testing reliability of workload mixes for throughput metric

Through-put	Feature selection(app1)			MLP classifier(app2)	
	Average of 5 iteration	120 KB	0.0685	8 KB	0.0889
		150 MB	0.0364	100 MB	0.0267
		1G	0.0282		
	Average of 5 iteration	120 KB	0.0685	8 KB	0.0889
		150 MB	0.0364	100 MB	0.0267
		1G	0.0282		
	Standard deviation 120 KB		0	Standard deviation 8 KB	0
Standard deviation 150 MB		0	Standard deviation 100 MB	0	
Standard deviation 1 G		0			

TABLE 3.9 Characteristic of XGBoost classifier and Random forest

# of users	# of workers	Size of data	total time of experiment	of VMs	number of CPU cores for each worker	Amount of memory of each worker	Application	Think time
5,15,20	3	150 MB	15 min	1	4	14 GB	XGBoost classifier	50 s
5,15,20	3	150 MB	15 min	1	4	14 GB	Random forest classifier	50 s

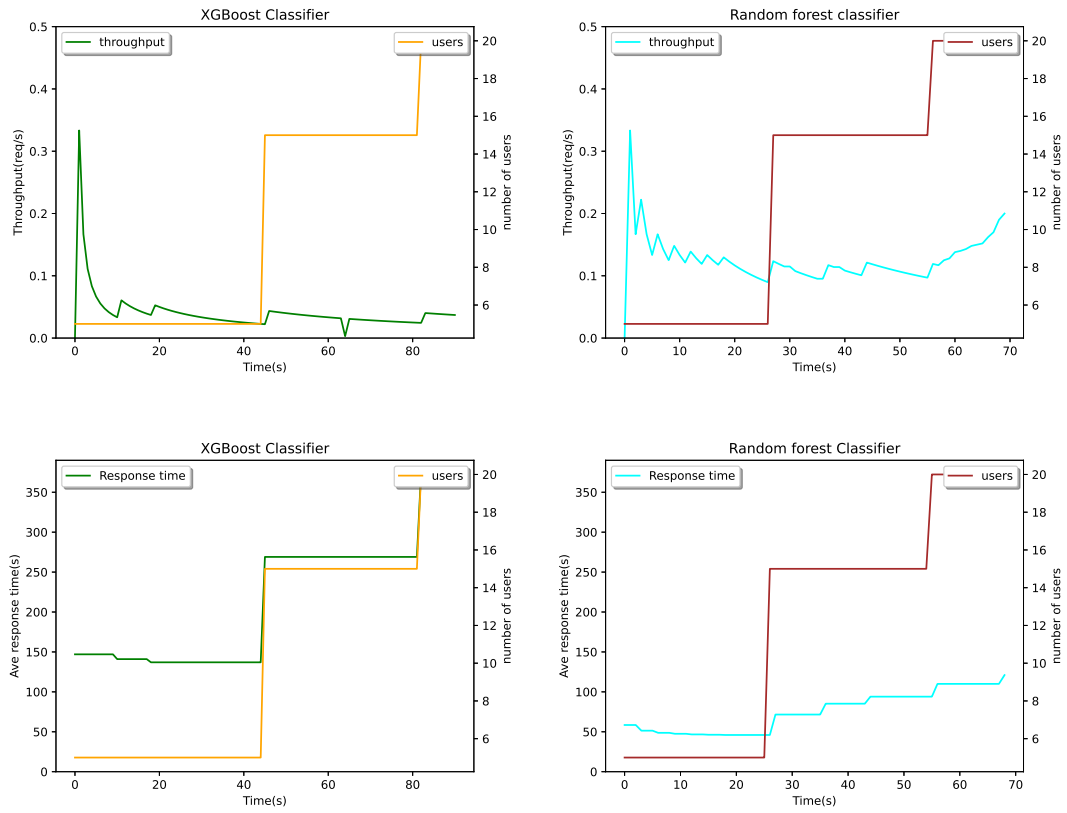


FIGURE 3.4 Comparative Analysis (a) XGBoost classifier (b) Random forest in terms of throughput and (c) XGBoost classifier (d) Random forest classifier in terms of response time

A reliable framework would yield results with low variability and high consistency, while an unreliable one would exhibit significant variation in the outcomes.

To evaluate the reliability of the SparkPerf benchmarking platform, we conducted an experiment aimed at assessing its performance under different scenarios. When conducting experiments to evaluate the reliability of a benchmark, the following areas can be the focus of our evaluation : We can follow these steps :

Set up the experiment environment : First, we need to define the specific workload or task to be benchmarked. This involves configuring the necessary infrastructure, including configuring the computing resources such as the number of servers, CPU cores, memory, and storage capacity, as well as infrastructure configuration, Spark settings, and workload characteristics.

Run the experiment multiple times and calculate the average throughput and response time : Then, execute the benchmark task 10 times, each time recording the average of two metrics throughput and ave response time measurements.

Calculate the standard deviation : Standard deviation is a statistical metric employed to measure the extent of diversity or spread within a dataset¹¹. In our context, we calculated standard deviations to understand the extent to which the results from each of the ten runs of an experiment differ from the average result. A higher standard deviation indicates greater variability, suggesting less reliability in the measurements, while a lower standard deviation indicates more consistency and higher reliability. To assess the variability and consistency of the benchmark results, calculate the standard deviation. This involves measuring the difference between each individual throughput or response time value and the average throughput or average response time.

In the first experiment, we simulated a workload by having a specified number of users sending requests to the system for approximately 40 minutes. The experiment focused on evaluating the performance of the Spark job, specifically the XGBoost classifier, with varying numbers of workers and users. The characteristic of our experiment is shown in the table 3.2. Each run involved executing the Spark job and capturing measurements of response time and throughput as mentioned before.

The second experiment is done for an application like feature selection.

Finally, the last experiment is done to evaluate the reliability of the workload mixes.

After conducting the experiments, we calculated the average of two metrics average response time and throughput across the 10 runs. This allowed us to determine the typical performance of the SparkPerf platform under different user and worker combinations. Additionally,

11. <https://www.scribbr.com/statistics/standarddeviation>

we computed the standard deviation to assess the variability in the results and gauge the platform’s consistency and reliability. By performing these calculations, we obtained meaningful insights into the reliability of the SparkPerf benchmarking platform. The standard deviation helps identify any variations or fluctuations in the measurements. In fact, when the condition of an experiment remains constant, the result should be consistent and stable, regardless of the time the experiment is conducted. These findings contribute to a comprehensive assessment of the platform’s reliability and suitability for handling diverse workloads and user scenarios. The result is shown in Tables 3.4 and 3.5, 3.7, and 3.8. The characteristic of the experiment is shown in Table 3.3.

Also, it is worth mentioning that the small standard deviation of the throughput metric is related to the fact the system is not being saturated. When the standard deviation of throughput is small, it indicates that the system is operating in a relatively stable and predictable manner. Here is a more detail on this concept :

System Saturation : In the context of system performance and resource utilization, “saturation” refers to the point at which the system is fully loaded or operating at its maximum capacity. When a system is saturated, it typically experiences delays, increased queuing, and reduced performance because it cannot handle additional workload efficiently.

Throughput and Arrival Rate : Throughput measures the rate at which a system successfully processes and completes requests or tasks [1]. The arrival rate is the rate at which requests or events arrive in the system. When the throughput matches or closely matches the arrival rate, it suggests that the system is efficiently processing incoming requests without significant delays or queuing [1].

Standard Deviation of Throughput : The standard deviation of throughput is an indicator of the variability in the throughput values over time. A small standard deviation indicates that the throughput values are relatively consistent and close to the average throughput.

3.6 Strength and limitation

Benchmarking ML pipelines allows researchers and practitioners to assess the performance of various approaches and techniques within the pipeline. By systematically comparing different pipeline configurations, it becomes possible to identify the most effective combination of techniques and algorithms for a given task or dataset.

Our primary objective is to evaluate the performance of a new algorithm developed by the researcher. The benchmarking process using Spark provided a valuable platform for this assessment. Regardless of the algorithm being tested, Spark’s benchmarking capabilities al-

lowed for seamless integration and evaluation. The benchmark could easily compile and load the new algorithm, ensuring a fair and standardized comparison. This enabled us to conduct a comprehensive analysis, comparing the performance of the new algorithm against other established algorithms. The flexibility of the benchmarking framework offered by Spark allowed us to assess the efficacy of the new algorithm with ease, providing valuable insights into its potential superiority. The ability to include any algorithm within the benchmarking framework enhanced the credibility and applicability of the results obtained. By employing Spark as the benchmarking tool, we were able to validate the effectiveness of the new algorithm, paving the way for further advancements in the field.

In addition to its benchmarking capabilities, SparkPerf shows its versatility by providing the ability to generate synthetic data for evaluation purposes. This feature is particularly valuable in our study as it allowed us to assess the performance of the new algorithm across a wide range of scenarios and data distributions. By generating synthetic data, we are able to simulate various real-world conditions and thoroughly test the algorithm's robustness and effectiveness. This capability not only ensured a comprehensive evaluation but also enabled us to explore the algorithm's performance boundaries beyond the limitations of existing datasets. The ability to generate synthetic data within the Spark benchmarking framework significantly enhanced the reliability and generalizability of our findings, making them more applicable to real-world scenarios. As a result, we gained deeper insights into the strengths and limitations of the new algorithm, further supporting its potential as an innovative solution in the field.

Furthermore, one of the notable advantages of using SparkPerf as a benchmarking tool is its portability. Our benchmarking framework can be easily deployed and utilized across different environments and systems. This portability ensures that the evaluation of the new algorithm is not limited to a specific platform or infrastructure. Researchers can seamlessly transfer their benchmarking setup to different environments, including cloud-based systems, or even local machines, without significant modifications or compatibility issues. The ability to maintain consistency in the benchmarking process across various environments enhances the reliability and reproducibility of the results. The portability of SparkPerf's benchmarking framework empowers researchers to conduct evaluations in diverse computing environments, ensuring the algorithm's performance is assessed under realistic conditions and providing a strong foundation for broader adoption and practical implementation.

3.7 Conclusion

This thesis introduces SparkPerf, a benchmarking framework designed for assessing the performance of data science projects based on Apache Spark. SparkPerf incorporates a configurable workload generator capable of simulating concurrent users over a given timeframe. The framework demonstrates its efficacy by successfully loading user applications onto the Apache Spark platform, thereby demonstrating its compatibility with diverse application types. Furthermore, the framework exhibits the platform's ability to generate synthetic data, providing a valuable resource for rigorous testing, thereby contributing to a comprehensive understanding of Spark's performance and its suitability for diverse data processing tasks. By employing a transactional workload generator that simulates real-world scenarios, the framework measures performance metrics such as the successful number of requests in a given time as throughput, average response time, CPU utilization, memory consumption, and so on. These metrics are stored in Prometheus, and visualized through Grafana, a customizable dashboard. We have defined a variety of experiments to evaluate SparkPerf and its special characteristics. Evaluating portability, reliability, and the ability to load users' data and applications, as well as generating synthetic datasets are crucial aspects of benchmarking that we consider. These evaluations can provide valuable insights into the performance and capabilities of SparkPerf.

CHAPTER 4 SPARKSIM : A PERFORMANCE MODEL FOR RESOURCE ALLOCATION OF SPARK-BASED DATA SCIENCE PROJECTS

4.1 Preamble

The code can be found here :

<https://github.com/Soude640/Simulation-based-system>

4.2 Introduction

Data science and big data analytics have evolved into crucial components for enterprises for gaining meaningful insights from data, but also to evaluate and optimize their own system performance. Besides the benefits, the voluminous data can introduce scalability and orchestration problems. Moreover, with the increasing complexity of data analytics platforms, system administrators encounter the challenge of configuring and fine-tuning a substantial number of nodes, some of which are outfitted with various CPU and memory capacities. This translates into additional overhead to budget for human and equipment resources. Any significant deviation in planning for data engineering and analytics tasks can prove detrimental for companies in terms of cost, productivity, loss of revenue, and scheduling.

Performance modeling can help estimate the best resource allocation for various goals, such as lowest resource consumption, highest throughput, and so on. Performance models can be used in various capacities, such as to estimate the initial resource allocation, before the first deployment, or to adjust the allocation when there are performance issues. In the latter case, the models can act as a planning module, like in a MAPE (monitoring, analysis, planning, execution) self-adaptive framework [64], which can quickly simulate and estimate the impact of a number of adaptive actions and select the best one. Such a model needs to be accurate and be able to receive a variety of inputs to test for complex scenarios.

Prior research in performance modeling is generally classified into two types : analytical models for generating a task flow [44] and model-based machine learning methods [45]. Analytical models are considered a white-box approach, based on mathematical equations [65], while machine learning (ML) models are black-box models, which tend to favor broader applicability over accuracy, by using a limited number of parameters to promote simplicity [25]. Nevertheless, recent studies showed that for predicting the performance of big data analytics, ML models are almost as accurate, if not more, compared to simple analytical models [60].

Furthermore, it is worth noting that while previous studies employed batch workloads, our study focuses on an open transactional workload. Batch workloads are good for estimating the performance of a system for a given load, but they do not account for longitudinal loads that may cause long-term saturation. Models trained on transactional workloads can be more suitable for self-adaptive systems that need to guarantee the performance of a system continuously.

In this work, we present *SparkSim*, a platform that leverages machine learning to build performance models for data analytics performance prediction. SparkSim includes a systematic pipeline to configure, train, test and validate various machine learning models, and a final model for the estimation of data-analytics performance. We consider both traditional ML models like AdaBoost and deep-learning models like LSTM and MLP. Our work focuses specifically on Apache Spark¹, a popular and highly capable big data analytics platform. Compared to other platforms, Spark’s advantage lies in its ease of deployment anywhere (on-premises or on the cloud) as it does not directly depend on the storage infrastructure [10]. The research conducted in this study is driven by the specific requirements and demands of our industry partner. The main assumption is that a data science team performs a large number of data tasks, model training, testing, and deployment, continuously with varying data sizes and configurations. It is also assumed that multiple data scientists will need to access the infrastructure simultaneously, thus creating the conditions for a high workload. This implies that there will be a concurrent demand for accessing and utilizing the infrastructure’s resources. It is worth noting that this workload will not remain constant but will vary continuously over time, reflecting the dynamic nature of the tasks and requests from the data scientists. Therefore, the infrastructure must be capable of dynamically and continuously accommodating and efficiently managing this high and ever-changing workload to ensure smooth and uninterrupted data processing and analysis for all concurrent users.

In summary, our work makes the following contributions.

- **Performance model creation pipeline :** We present a systematic methodology to train, evaluate and compare multiple machine learning and deep learning performance models. The comparative framework includes steps to train and tune a variety of models and then compare their relative performance using Spearman’s ranking coefficient and statistical ranking. The pipeline is structured and can be easily customized and reproduced for a variety of performance problems in a variety of systems, not only for Spark.
- **Machine learning performance estimation :** We present the results for a number

1. <https://spark.apache.org/>

of different ML-based performance models for Spark-ML and we choose the one that best estimates performance in terms of response time and throughput.

- **Exploring important features :** We further analyze the results of the performance model by identifying the important features that significantly influence the performance of the Spark analytics platform. From a planning perspective, these features can be used in to better allocate resources, calibrate the application or the infrastructure, and potentially lead to optimized deployments of ML applications on Spark.

The rest of the chapter is structured as follows. Section 4.3 shows the experiment design, and section 4.4 explains the performance modeling. Section 4.5 demonstrates model validation. Section 4.6 describes the results and analysis. Section 4.7 presents the discussion and Section 4.8 concludes the thesis.

4.3 SparkSim model creation pipeline

Figure 4.1 illustrates the workflow of our SparkSim framework. The first step is the data collection consisting of a workload generator that simulates data scientists submitting data analytics jobs to a Spark topology, which is being monitored regularly to gather performance data. The generated and monitored data is then used to train a number of different regression-based machine learning models. Finally, the accuracy of the various models is compared and a selection is made based on a number of different quality metrics.

4.3.1 Workload Design

Setting up the Spark application

Spark has been selected as the data analytics solution for the following reasons. Spark’s high scalability makes it a compelling choice, allowing for the processing of large and complex datasets and computations that may not be feasible or efficient with other technologies [66]. Moreover, Spark is widely recognized for its speed, due to its ability to data processing, in-memory and distribute computations over a cluster of machines [67].

In this work, we assume Spark jobs that perform different kinds of machine learning algorithms, including feature engineering, classification, and clustering. Classification algorithms involve algorithms like XGBoost, Random Forest, and Multi-Layer Perceptron (MLP). Moreover, clustering algorithms including K-Means are considered. Classification and clustering can be highly demanding in terms of computation resources, response time, and throughput in comparison with feature selection. In fact, incorporating different kinds of machine lear-

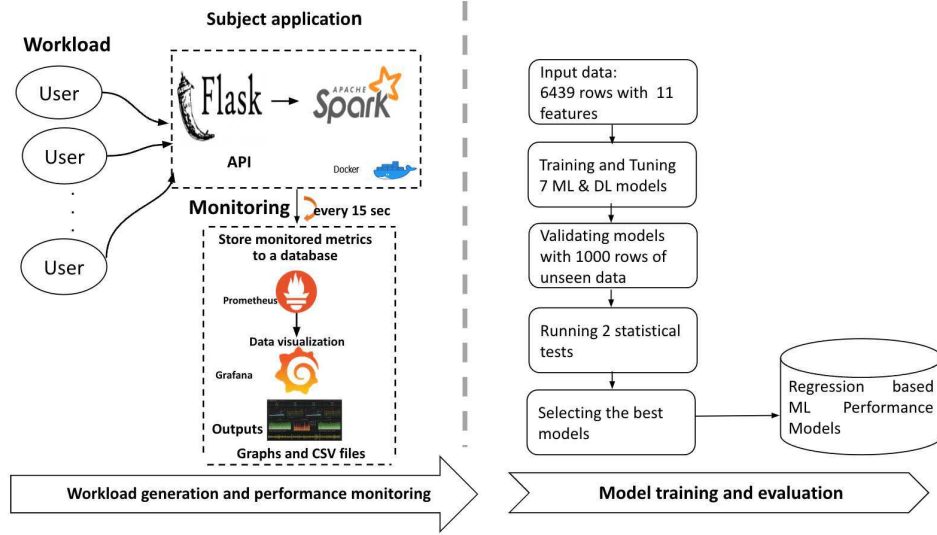


FIGURE 4.1 The SparkSim workflow including workload generation and performance monitoring, model training, and the performance model

ning algorithms as Spark jobs is crucial for several reasons, including robust generalization. The Spark jobs are developed in Python, using Pyspark, and the MLlib Spark library for machine learning.

The goal of workload design is to cover a range of scenarios that are likely to be encountered in real-world performance modeling. Different numbers of workers, data sizes, different numbers of users, different types of machine learning tasks, and different deployment environments reflect the diversity of conditions under which our model or system might operate.

In the scenario we examine here, this Spark job is supposed to be used concurrently by a number of users, each of whom submits a file of varying sizes to be analyzed by the job. The workload is modeled as an *open transactional workload* [68]. This type of workload better captures the motivating example with the varying number of concurrent data scientists. On the other hand, a transactional workload simulates continuously arriving requests, as opposed to a fixed number of requests. In the former case, the experiment ends within a given amount of time [1]. In the latter case, the experiment ends after all requests have been received and served. Once again, the transactional workload better simulates the continuous traffic to our Spark cluster [68], [1].

TABLE 4.1 Description of the Features and Output variables.

Feature Type	Variable	Description	Values
Spark topology	W	Number of workers	1-8
	CW	VCPU(numbers of CPU cores) assigned to workers	3-32
	MW	Amount of memory assigned to workers in GB	4-80
Cloud infrastructure	VM	Number of virtual machines	1-2
	MVM	Total memory of Virtual Machines in GB	16-128
	CVM	Number of CPU cores of Virtual Machines	16-32
Size of data	SD	Data file submitted as a Spark job	5KB-8KB-120KB-100MB-147MB-150MB-1G
Type of ML task	TML	Type of Spark job	Feature selection, K-means, XGBoost, Random forest, MLP
Deployment environment	On-prem	Machine learning models is hosted and executed on a local on-premises server	0-1
Size of workload	U	Number of users	3-20
	th	Think time of each user in seconds	10,50
Output variables	Rt	Average response time in seconds	0.662-810.88
	X	Requests per seconds	0.011-11.33

Feature selection

The input variables for our performance model are shown in Table 4.1. These parameters included the size of the Spark topology in terms of the number of Spark workers, and number of VCPUs and memory assigned to each worker, the size of the cloud infrastructure in terms of the number of Virtual Machines, and the total number of CPU cores and memory per VM. The input parameters also included the size of the data, and the number of users, varying per experiment, the type of machine learning tasks as the Spark job, and the deployment environment, which is considered as the environment, in which the experiments are done. The output variables monitored to train the models were the average response time per successfully executed job and the number of served requests per second.

4.3.2 Experiment execution and feature monitoring

Experiments to collect data were conducted with SparkPerf, a machine learning benchmarking framework for Spark applications [69]. The experiments are shown in Table 4.2.

Setting up the data collection process

Docker was used to deploy both Spark and the subject application. Docker provides the desired efficiency and ease in deploying the application and scaling it if necessary. Therefore, we preferred it to increase our capacity to perform multiple experiments and gather as much data as possible quickly to train our models. Nevertheless, we provided a range of configurations for Docker, in terms of CPU, memory, and number of VMs available to the containers, so that its impact is also included in the performance models. After all, the adaptivity of Spark and Docker has been studied before [70, 71].

The different Spark workers were deployed in separate Docker containers. Our experiments used a varying number of Spark workers and VMs on which Spark workers are deployed to allow our performance model to incorporate scaling data and allow it to operate as a planning module for a self-adaptive system. Finally, a Flask server was used as an API gateway to allow users to submit jobs to the Spark cluster.

Feature monitoring

We leverage the Prometheus² and Grafana³ platforms to have a comprehensive view of the system performance, including Average response time, number of successful requests, and so

2. <https://prometheus.io/>

3. <https://grafana.com/>

TABLE 4.2 Experimental Data for Machine Learning Model Training and Model Validation

Experiment type	Application	Number of workers	Number of vms	Size of data	Number of users
Remote server	Feature selection	1-2-3-4	1-2	100 KB, 147 MB, 1 G	5-20
	mixed workload (MLP, feature selection)	8	2	120 KB, 150 MB, 1 G, 8 K, 100 MB	20
	MLP	2	1	400 MB	5-10
	K-means	2	1	1 G	5-8
	Random forest	3	1	100KB	5-15-20
	XGBoost	3	1	100KB	5-15-20
	MLP	2	1	100KB	5-10
	MLP	4	1	1G	5-10
On-prem	MLP	2	-	100 MB	5-10
	MLP	2	-	400 MB	5-10
	k-means	2	-	400 MB	3-4

on. Prometheus is an open-source monitoring approach. Its main functionality is to obtain the status of the monitored component at predetermined intervals. Grafana⁴ is an open-source platform for visualizing data that empowers users to generate and view dynamic and customizable dashboards that offer analytics and insights across diverse data sources. Moreover, Grafana integrates with many popular data sources and platforms, including Prometheus. There are multiple steps involved in setting up Prometheus and Grafana, including setting up and configuring the Prometheus server, configuring data sources and dashboards in Grafana, and installing and configuring any necessary integrations. It is worth mentioning that monitoring is done every 15 seconds. Indeed, Prometheus sends its data every 10 seconds, but we have configured it every 15 seconds. Specifically, we define the `scrape_interval` (Prometheus.yaml) to 15 seconds from the configuration file. The metrics can be updated more frequently with a shorter scrape interval.

The link to our repository for the final dataset and the source code for building the models⁵ is shared.

We built two different models for our target variables, including a response time model and a throughput model, based on the aforementioned algorithms. The predictant variable from our monitoring data, average response time, is used to build a response time model. However, we do not have any data for the throughput metric, so we need to calculate this metric from our data. As we monitor data on the number of successful requests from our monitoring tool, we can easily calculate the metric, as throughput quantifies the system's capacity to handle a specific quantity of requests within a defined time frame [72].

4. <https://grafana.com/>

5. <https://github.com/Soude640/Simulation-based-system>

We monitored the performance metrics from 3 sources : 1) Spark cluster : the collected metrics would be the average response time, the number of successful requests, the number of failed requests, and so on 2) Docker container resources, including container CPU, memory usage, and container network input/output. 3) Docker host : The consumed resources of the host, in terms of CPU and memory utilization, network, and I/O usage. Data from the Spark cluster is used directly for building the machine learning models, including average response time and number of served requests as target variables, and the number of users and number of workers as independent variables. While the other two help us assess the performance when Spark encounters issues. The Docker container and host metrics provide insights into the environment in which our Spark jobs are executing. While they can be valuable for troubleshooting and optimization, our primary goal is how Spark jobs perform within this environment, and when we face some anomalies and issues (saturation event), we can analyze the resource utilization of the Docker container and Docker host to find the point. In fact, monitoring these metrics allows us to identify potential performance bottlenecks and resource constraints in real-time. By closely tracking container CPU and memory usage, we can optimize resource allocation and prevent resource contention issues that could impact job execution times. Monitoring metrics like container CPU and memory usage of Docker container and Docker host helps ensure that Spark jobs are effectively utilizing available resources without overloading containers or hosts. This can lead to improved performance by preventing resource shortages that could otherwise slow down job execution.

4.3.3 Experimental setup

We conducted different experiments on an OpenStack⁶ cloud platform. “OpenStack is an open-source cloud computing infrastructure software⁷”. The infrastructure of the experiment and software version is presented in Table 4.4. The experiments were performed with different data sizes for different numbers of workers on a dedicated VM, provisioned with 60 GB of RAM and 16 cores.

The experiments were conducted on a single VM for three different data sizes : 100 KB, 147 MB, and 1 GB, 400 MB, 5 KB, and 120 KB. The first set of experiments involved testing various Spark topologies with 1, 2, 3, and 4 workers for the 100 KB, 147 MB, and 1 G datasets. The second experiment covers the evaluation of workload mixes, including the execution of two distinct types of Spark jobs : one involving feature selection and the other employing an MLP classifier. The experiment is conducted on two virtual machines, with each machine

6. <https://www.openstack.org/>

7. <https://www.openstack.org/>

accommodating four workers. Additionally, the experiment involved data of varying sizes, spanning a range from 120 KB to 150 MB, 1 GB, 8 KB, and 100 MB. The third experiment is done for an MLP classifier with 2 workers on a single VM, for the 400 MB of data, and so on. For all these experiments, we leverage remote server infrastructure OpenStack VM. The experiment also includes data sizes of 147 MB with 4 workers with 2 VMs.

In addition, 3 experiments as shown in the table 4.2 are done locally(on-rem), including MLP classifier and K-Means with 2 workers targeting data sizes of 100 KB and 400 MB.

In total, 22 experiments were performed. After analyzing this collected data, we find that for 147 MB of data, between 1, 2, and 3 workers, there is a natural progression of response time and throughput. Response time constantly decreases and throughput consistently increases. However, when we go with 4 workers, we see that response time and throughput are worse than with 3 workers. With 4 workers we start to hit the limit of the available memory in the host (91.6% total memory use). This can be a case of saturation, causing the performance to decrease a bit.

We collected a dataset comprising 6439 rows and 11 columns. For the first set of experiments, the dataset is a combination of three different data sizes, each processed with a different number of Spark workers. Indeed, this is a combination of 3 different data sizes with different numbers of workers of Spark. The inclusion of three different data sizes implies that the dataset encompasses varying levels of data volume or complexity, allowing for the exploration of how Spark’s performance and scalability are influenced by different workloads. Additionally, the utilization of different numbers of Spark workers indicates that the dataset was processed with varying levels of distributed computing resources, enabling the examination of how Spark’s performance is impacted by the allocation of different numbers of workers to handle the data processing tasks.

4.4 Performance modeling

We create performance models for predicting Spark application performance, and utilize data collected from performance measurements to train the model. We leveraged SciKit-learn [73] for this purpose. The hyperparameters used for each model are shown in table 4.3. The hyperparameters are tuned based on a random search technique⁸, in which a set of hyperparameters is randomly sampled from a distribution and tested. For XGBoost, we used Grid search⁹. In grid search, hyperparameters are defined and tested exhaustively using a pre-defined grid of parameter values.

8. https://scikit-learn.org/sklearn.model_selection.RandomizedSearchCV.html

9. https://scikitlearn.org/stable/sklearn.model_selection.GridSearchCV.html

TABLE 4.3 Hyper-parameter Tuning for the Different Models

Model	Hyper-parameters	Values
Decision tree	Max depth	40
Random Forest	Estimators	12
	Max depth	50,100,150
	Max features	auto,sqrt
XGBoost	Gamma	[0, 0.1, 0.2]
	Estimators	[100,200,300]
	Max depth	[3,5,7]
	Learning rate	[0.01,0.1,0.2]
	Subsample	[0.8, 1.0]
LSTM	Loss	MSE
	Numbers of hidden layers	2
	Maximum Epoch	300
	Batch Size	64
	Learning rate alpha	0.0001, 0.05
	Dropout	0.2, 0.2
MLP	Solver	sgd, adam
	Activation function	tanh, relu
	Max iteration	50, 100
	Learning rate	Constant, adaptive
	Learning rate alpha	0.0001, 0.05
	Hidden layer size	150,100,50

In the following, we outline the machine learning-based performance model and deep learning models of our Spark application. Based on the data provided by the previous step, we can build different regression-based machine learning models as well as deep learning models. As our targeted variables are continuous, regression-based machine learning algorithms are explored. The features used for training the models are the number of users, size of data, number of Spark nodes, number of VMs, amount of memory of workers, total memory of VMs, number of VCPUs, type of machine learning tasks, on-premises deployment, and think time of each user. Table 4.1 demonstrates our features, target variables, and the ranges of all the variables.

Overview of the models

We employed different regression-based machine learning models (ML). Regression-based ML approaches are the supervised learning methods for modeling and predicting continuous, numeric variables, like our response variables [74].

Linear Regression. Linear models are easily updatable with the inclusion of new data, however, even if the proper interaction terms are added, they frequently lack the flexibility to detect more complicated patterns. This can also be clearly observed in our linear regression model [75].

Adaptive Boosting (AdaBoost) regression. Adaboost’s fundamental foundation is to turn a series of weak learners [50] into strong ones. In addition, Adaboost is relatively robust to over-fitting and does need to make any assumptions about the independent variables.

XGBoost Regression. XGBoost can be a suitable algorithm when dealing with big data. However, if the parameters are not precisely tuned, this can lead to overfitting. So, the parameters should be determined properly [76].

Decision Tree Regression. It can be considered a decision-making method [77]. Despite being complex on some trees, they can be simply interpreted. It would be fairly fast, even on large datasets with multiple predictors, and it would be possible to include continuous independent variables in the same model. Nevertheless, decision trees have a high degree of variability; that is, a small change in the data can substantially affect the tree by varying the splitting rules.

Random Forest Regression. One of the Ensemble ML algorithms used for regression analysis would be Random Forest. It employs the bagging technique [78] to select the features for the best node split and construct multiple decision trees, and then it averages the values of all decision trees to predict accuracy.

LSTM Regression. “LSTM (Long Short-Term Memory), a specialized type of Recurrent Neural Network (RNN), is commonly used to resolve gradient vanishing issues during extended sequence training and is well-suited for time series forecasting tasks ” [79] [80].

Multi-layer perceptron (MLP) Regression. A multi-layer perceptron regressor is a kind of neural network, that serves as the foundation for an artificial neural network, to train the data based on the perceptron. MLP is based on a three-layer model, including input, output, and hidden layers [81].

4.4.1 Model evaluation

We use statistical significance tests, including Scottknot effect size, and Spearman rank coefficient. and also error measures to assess the result of our approaches.

Performance evaluation metrics

The effectiveness of a regression model’s capability to estimate the dependent variable from the independent variables is evaluated using performance metrics¹⁰. The performance evaluation metrics employed in our regression analysis are as follows. The first used metric is the Mean Absolute Error (MAE), which is “the average absolute difference between the actual and predicted values” [82]. The second metric is the Root Mean Square Error (RMSE). In fact, RMSE is the square root of the error, which is used to evaluate the average size of the predictions’ errors in the same units as the dependent variable. MSE is another metric considered, which is Mean squared error [83]. Moreover, R^2 indicates the extent to which the model accurately matches the data with a value between 0 and 1, where a greater number indicates that the model fits the data better [83].

10. <https://www.enjoyalgorithms.com/blog/evaluation-metrics-regression-models>

TABLE 4.4 Experimental Configuration of the VM and Spark Cluster

VM Configuration	
VCPUs	16 cores
Main memory	60GB
Storage	20GB
Software	
Spark	3.2.1
Docker	20.10.14

Spearman’s rank correlation

Spearman’s rank correlation test [84] measures how similar (correlated) two data distributions (e.g., actual and predicted samples) are.

Scottknott

Another statistical test is the ScottKnott Effect Size Difference (ESD)¹¹. Scott-Knott is a statistical approach for doing post-hoc pairwise comparisons of R-squared. The test is frequently used in data analysis and experimental design, which was developed by Richard J. Scott and M. A. C. Knott [85]. The process involves first sorting the R-squared of the groups being compared in descending order, and then applying a clustering algorithm to group them into subgroups with similar R-squared. The number of subgroups is picked based on a heuristic method involving the use of a minimum significant difference. Finally, the subgroups are ranked based on their R-squared values, and statistical tests are done to compare the R-squared of the subgroups to each other [85]. Scott-Knott is extremely helpful in situations where there are many groups being compared like in our case and the differences between them are small, making it challenging to identify which groups are considerably different from each other. Thus, the method aims to identify significant differences in the R-squared of various regression models and determine whether these differences are practically meaningful [85]. Once we have collected multiple observations for each model, we can analyze and compare their distributions. We obtained multiple observation metrics for each model’s metric through cross-validation by performing multiple rounds (10 times) of cross-validation. Each round involves partitioning the data into different subsets, resulting in multiple evaluation metrics for each model.

4.5 Model Validation

Model validation is a critical phase in machine learning to assess the performance and reliability of a trained model. We validated the model through unseen test data and considering feature importance.

Unseen test data

Model validation with distinct test data can be used to evaluate the performance of a trained model on new, unseen data [86]. The intention is to make sure the model can generalize

11. <https://github.com/klainfo/ScottKnottESD>

properly and make predictions correctly on data that it has never seen before. The test set is utilized to assess the performance of the machine learning model [87], while the previously trained dataset is utilized to train the model. This is imperative as it helps avoid overfitting, which happens when a model is extremely complex and does not perform well on the new data.

The distinct test data includes 1000 rows of data and includes applications such as MLP classifiers, feature selection, and K-Means with different sizes of data, including 8 KB to 1 G for different numbers of users ranging from 5 to 20. The distinct test data is obtained from the previously mentioned experiment. The test data is obtained through a process of random sampling from the input data. Random sampling involves selecting a subset of data points in a way that each point has an identical probability of being chosen [88].

Feature importance

Feature importance is a critical aspect in machine learning models, that helps us find which feature contributes the most to the prediction of the models [89]. We used the permutation feature importance [90] to find out which features in our regression models have more importance for the targeted variables. An important feature is considered to be the one that results in the greatest drop in performance [91].

4.6 Results and Analysis

Table 4.5 shows the performance of the different performance models for the target variables of response time and throughput. As shown in Table 4.5, MLP and LSTM outperform other models based on R^2 score and Spearman correlation for both the response time model and throughput model. With respect to the mean squared error(MSE), root mean square error (RMSE), and mean absolute error (MAE), it becomes apparent that the LSTM model outperforms the other response time models. and for the throughput model, MLP is the best. MLP and LSTM are different types of neural network architectures with varying levels of complexity and capacity to capture different patterns in the data. LSTM is specifically designed for sequential data like in our case and can capture temporal dependencies. As the data exhibits time-dependent patterns that influence response time, LSTM's ability to capture temporal relationships might lead to better performance in response time metrics. Different metrics focus on different aspects of model performance. R-squared score and Spearman correlation measure the overall fit and monotonic relationship, respectively, while RMSE and MAE focus on prediction errors. A model might perform well in terms of fit (R-squared,

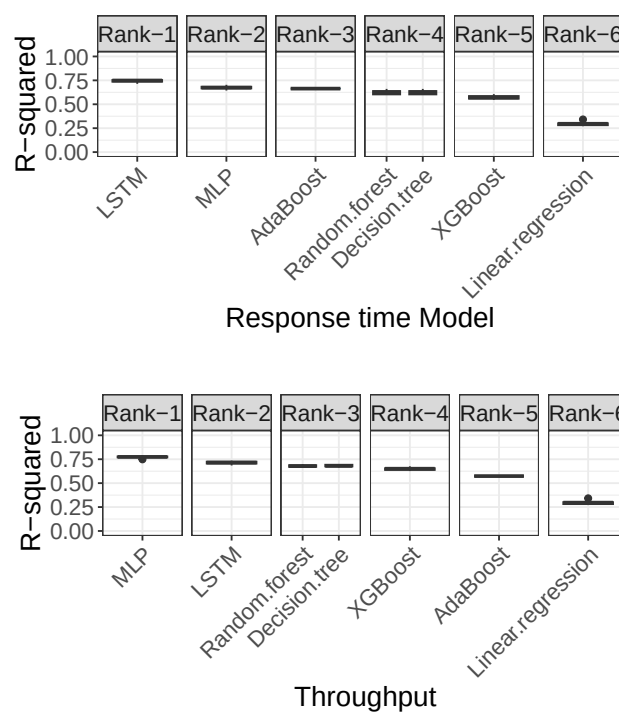


FIGURE 4.2 ScottKnot Test for Comparing ML Models Regarding Top Model as Response Time and Bottom as Throughput

TABLE4.5 Comparison of Different Performance Evaluation Metrics and their corresponding Spearman rank coefficient : The top table shows the results of the Response Time Models, and the bottom table shows the results of the Throughput Models, both on the validation data.

Model's name	R^2 score	MSE	MAE	RMSE	Spear coef.
Linear Regression	0.341	7312.39	55.82	85.50	0.10
AdaBoost	0.663	3733.05	27.24	61.10	0.67
XGBoost	0.595	3721.22	23.13	61.38	0.55
Decision Tree	0.657	3875.69	23.71	62.25	0.61
Random Forest	0.658	3795.23	23.21	61.60	0.66
LSTM	0.740	1408.86	17.48	37.55	0.87
MLP	0.735	2935.31	27.06	54.17	0.81
Model's name	R^2 score	MSE	MAE	RMSE	Spear coef.
Linear Regression	0.211	12354.8	201.31	89.81	0.285
AdaBoost	0.573	0.270	0.153	0.520	0.677
XGBoost	0.675	0.670	0.322	0.818	0.739
Decision Tree	0.694	0.195	0.108	0.453	0.640
Random Forest	0.687	0.204	0.112	0.452	0.680
LSTM	0.704	0.194	0.114	0.441	0.795
MLP	0.789	0.155	0.137	0.394	0.819

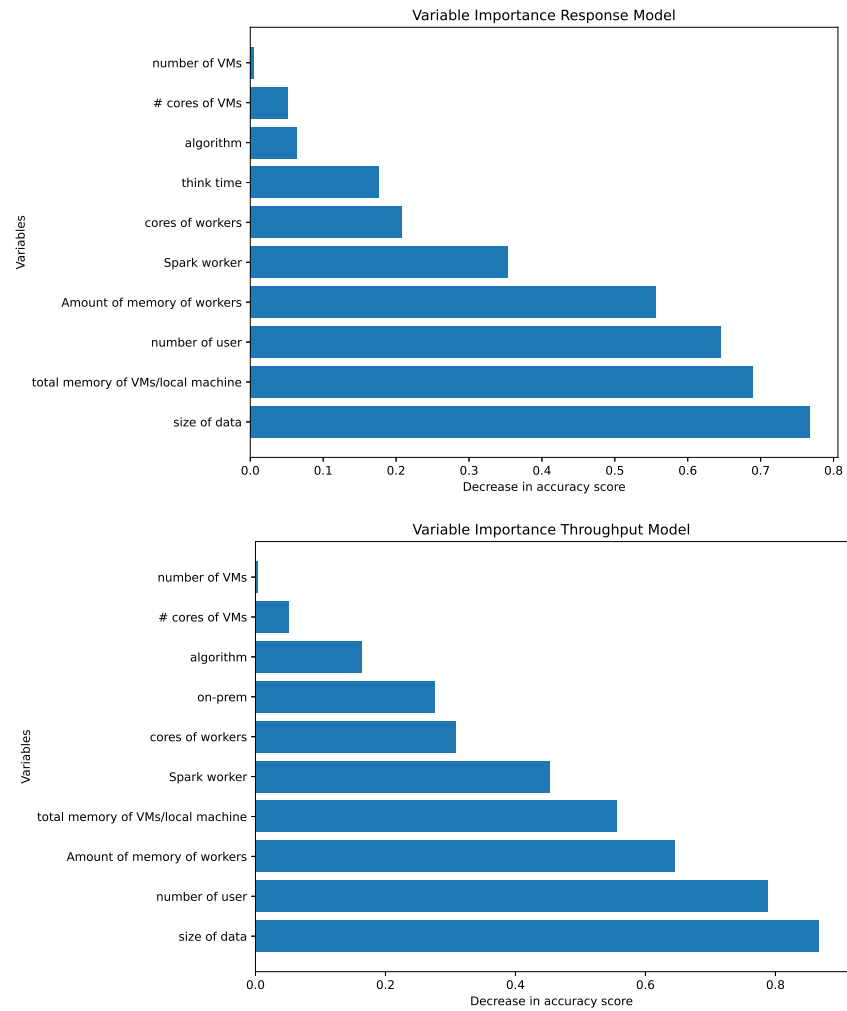


FIGURE 4.3 ScottKnot Test for Comparing ML Models Regarding Top Model as Response Time and Bottom as Throughput

correlation) but still have larger prediction errors (RMSE, MAE) due to its response to specific data points. With respect to the throughput model, the relationship between the independent features and the dependent variable (throughput) is highly non-linear, MLP's flexibility might allow it to capture those patterns better, resulting in higher R-squared and Spearman correlations.

Fig. 4.2 demonstrates the ranking of different models based on the Scott Knott test [92]. The deep learning models including LSTM and MLP are the first and second best models for the response time and MLP and LSTM for throughput models, which means the models are the best-performing ones.

Table 4.5 indicates a relationship between predicted values and actual values (indicated by Spearman's rank coefficient) for response time and throughput models. As can be seen in 4.5, LSTM and MLP seem to perform better than other performance models for our dataset.

LSTM and MLP are neural network architectures capable of capturing complex relationships within the data. If the relationship between the input features and the output variables (response time and throughput) is intricate and non-linear, these models might excel in capturing those patterns. LSTM, in particular, is designed to handle sequential and time-dependent data. If data exhibits temporal dependencies that influence response time and throughput, LSTM's architecture could be particularly effective in capturing those dependencies. Neural networks like LSTM and MLP can automatically learn relevant features from the data. This ability to learn meaningful representations might be helping LSTM and MLP models capture patterns that shallower models like random forests and decision trees struggle with. This can be advantageous when dealing with high-dimensional data or when interactions between features are important.

Linear regression has a poor performance in predicting response time and throughput compared with other models. In fact, linear regression supposes a linear correlation between the independent variables known as input features and the output variable. If the true relationship is more complex, linear regression might struggle to capture it effectively. Moreover, linear regression might not be able to effectively model interactions between features, which could be crucial in predicting response time and throughput.

Our deep learning-based performance models (LSTM and MLP) can accurately predict the response time and throughput of the Spark application, and outperform other classical counterparts.

For interpreting the results of permutation feature importance, we found that the five most important features are related to the size of data, number of users, amount of memory of

workers, total memory of VMs, and Spark workers for both response time and throughput model (Fig. 4.3).

The determination of the most important features related to the size of data, number of users, amount of memory of workers, total memory of VMs, and Spark workers for both the response time and throughput models can provide valuable insights into the factors that significantly have an impact the performance of our models. One potential reason for the **size of data** being the most important one is that larger datasets often require more processing time, as they involve more computation and potentially more data shuffling. This could lead to increased response times and affect response time and throughput. **Number of Users** : The number of users accessing the system can impact the demand for resources. Higher user activity can lead to increased job submissions, and resource utilization, affecting both response time and throughput. **Amount of Memory of Workers** : Memory availability is crucial for caching data and intermediate results. Adequate memory can considerably enhance Spark job performance as it decreases the need for disk Input/Output [93]. **Total Memory of VMs** : The collective memory across all VMs affects how much data can be cached in memory. More memory allows for efficient data processing and reduced reliance on disk storage. **Spark Workers** : The number of Spark workers directly influences parallelism and the ability to process tasks concurrently. More workers can lead to improved throughput and potentially shorter response times.

Practitioners can use the information obtained from feature importance to guide resource allocation decisions. Allocating sufficient memory and adjusting the number of Spark workers based on the anticipated data size and user demand can help optimize performance. Moreover, understanding the impact of data size and user load on performance can inform capacity planning efforts. Practitioners can estimate the resource requirements for different scenarios. When facing larger datasets or higher user loads, practitioners can adjust the system's resources accordingly. This might involve provisioning additional memory, increasing the number of workers, or optimizing VM memory. In addition, when facing performance issues, practitioners can focus on optimizing the features that have the most impact.

Features related to the size of data, number of users, amount of memory of VMs, and total memory of VMs/local machine are the most important in explaining the performance of the Spark applications.

TABLE 4.6 Wilcoxon signed-ranked test for measuring the significance of top 5 features

Metric	Model	Feature Set	Value	P-value
R-squared	LSTM	All features	0.74	0.1
R-squared	LSTM	Top 5 features	0.65	
R-squared	MLP	All features	0.78	0.1
R-squared	MLP	Top 5 features	0.72	
MSE	LSTM	All features	1408.86	0.1
MSE	LSTM	Top 5 features	2930.05	
MSE	MLP	All features	0.155	0.1
MSE	MLP	Top 5 features	0.228	
RMSE	LSTM	All features	37.552	0.1
RMSE	LSTM	Top 5 features	54.13	
RMSE	MLP	All features	0.394	0.1
RMSE	MLP	Top 5 features	0.474	
MAE	LSTM	All features	17.48	0.1
MAE	LSTM	Top 5 features	30.07	
MAE	MLP	All features	0.137	0.1
MAE	MLP	Top 5 features	0.156	

4.7 Discussion

Benefits of performance modeling. Performance modeling for data analytics applications, such as Spark, can provide multiple benefits. It helps to understand how these applications behave under various conditions and enables us to identify bottlenecks and areas for improvement. Performance modeling can also predict potential issues and aid in capacity planning, optimizing cluster size and hardware requirements for a given workload. Furthermore, performance modeling can help reduce the total cost of ownership by identifying cost-effective cluster configurations and hardware specifications. Additionally, the Spark performance model can be used on building simulations for resource scheduling and estimation, like in our case, predictive maintenance, and self-adaptive systems. Therefore, we can use our Spark performance model in order to have a prediction of Spark’s application performance under different configurations. Considering the scenario where we are executing spark analytics. At a certain point, the number of users increases (e.g., from 10 to 15), leading to elevated CPU utilization, decreased latency, and reduced throughput. Consequently, it becomes imperative to identify the underlying issue. In the planning phase, a recommended approach would involve running the model for the specific user count while maintaining a fixed output. This entails determining the appropriate file sizes corresponding to the given user count, thereby enabling an assessment of the required number of spark workers to achieve the desired response time or throughput. A well-informed strategy would involve scaling up the number of

workers from the current level to a higher value, thus resolving the problem.

By using regression machine learning models and deep learning models to predict the performance of Spark applications, we can save time and resources by avoiding the need to run the applications under different configurations to determine their performance.

There are existing works for Spark performance modeling, like [94]. One reason for choosing machine learning models over analytical models in Spark performance modeling is their versatility and adaptability to different contexts and applications. Machine learning models are particularly suited for handling complex data that may not be easily managed by analytical models. This stems from the fact that machine learning models can automatically find complex patterns and relationships in the data, whereas analytical models typically require assumptions and simplifications about the data. Another consideration is flexibility, as machine learning models are more flexible than analytical models since machine learning models can be trained on a broader spectrum of data types and formats, allowing enterprises to apply machine learning models to predict performance across a variety of use cases and applications.

Significance of the important features. The importance of feature analysis is intricately linked to the planning phase. It entails an evaluation of which configuration parameters yield the greatest influence over the model’s predictions or outcomes. Consequently, upon identifying these significant features, we have the potential to retrain the model with only the top five, excluding the others. This will guide us to a more streamlined model.

As a result, we start training the model with these selected five features, including the size of data, number of users, amount of memory of workers, total memory of VMs and spark workers, and using the Wilcoxon rank test [95] to compare the performance metrics after and before this removal. The result for the Wilcoxon test is shown in table 4.6. As can be seen, the observed statistic does not provide strong evidence against the null hypothesis. This suggests that there is no notable distinction between the two groups under comparison. The comparison metrics are R-squared, RMSE, MSE, and MAE. The result demonstrates that the difference between performance metrics is small so we can go with just 5 features. using a reduced set of features offers practical advantages in terms of computational efficiency, model performance, interpretability, and system resource utilization. It simplifies the data processing pipeline, reduces monitoring overhead, enhances model training and inference speed, and facilitates easier model maintenance. In addition, it makes deploying the model at runtime much more efficient as it saves on storage, memory and CPU due to the lower complexity of the model. This minimizes the impact of the model on cost, infrastructure use and interference with the managed application.

Uniqueness of SparkSim. To the best of our knowledge, there have not been any machine learning models for data analytics applications, specifically Spark systems, that consider computing capability, size of workload, size of data, and the Spark topologies simultaneously as input features in building machine learning performance models with considering open transactional workload. The main difference between our ML models and the existing ones would be our workload generator. The majority of existing models used batch workload, which does not cover every facet of big data systems. For example, it does not cover transactional workloads, which are increasingly important in modern big data systems. Furthermore, they might not accurately capture the complexity of practical workloads, and they have limited scalability. These kinds of workloads are designed to be run out of the box, which limits the ability to customize models to specific use cases or scenarios. Therefore, we have experimented with an open transactional workload over an extended period, in which the infrastructure may still carry the impact of the previous value of the workload, which would be better for testing the endurance of the system. So, this is the first difference between our ML models and the existing ones. Another difference would be in conducting statistical tests to opt for the best final models. The ones that outperform others. Also, the existing ML models for the Spark data analytics applications just evaluated the models with a specified dataset and did not validate them on separate test data.

4.8 Conclusion

In this thesis, we explored the challenge of predicting the performance of large-scale data analytics platforms in the presence of uncertainty and variability. Varying certain configuration parameters of data analytics applications, including computing capability, size of the input data, configuration parameters, and the size of the workload, type of machine learning task for Spark job, definitely have an impact on the performance of a data analytics system, such as response time and throughput. In this work, we design a performance modeling approach to predict the response time and throughput of Spark applications using multiple dimensions of metrics related to Spark topology, cloud infrastructure, input data size, workload size, and type of machine learning task. Various regression-based machine learning models were used to build such performance models. Our evaluation demonstrates that MLP and LSTM outperform other response time and throughput models in predicting performance metrics. We also found that the features related to the size of data and number of users, and total memory assigned to each worker, and the total memory of VMs play the most important role in explaining the performance of Spark applications. Our performance modeling approach can be leveraged by data scientists to determine the appropriate Spark configurations for their

applications or be leveraged by self-adaptive systems to automatically scale the computing resources for varying workloads.

CHAPTER 5 DISCUSSION

The discussion section further explores and contextualizes the findings and contributions of this study. In this section, we aim to assess the implications and importance of our research in the domain of performance modeling and benchmarking and the need to integrate them and have a decision support system.

The key findings will be explored, and outline the contributions for practice and future research. Furthermore, we will provide recommendations for further investigation and refinement of the proposed approach. Through this discussion, we seek to deepen our understanding of the research outcomes and their broader implications within this domain.

We assume that there is a company that leverages data science and machine learning for various tasks, including prediction and forecasting. Given the nature of these endeavors, a significant degree of uncertainty naturally arises during the model development and training process. The company frequently faces the challenge of optimizing the performance of its services, which are highly dependent on complex configurations and fluctuating workloads. Consequently, repetitive actions become quite apparent. First, experimenting with various configurations and workloads is time-consuming and resource-intensive. Second, they often lack accurate long-term insights into how changes in configurations impact performance over time. Third, deploying and executing real-world experiments can be risky and disruptive to their services.

To address these challenges and make data-driven decisions, the company has decided to implement a decision support system by integrating benchmarking and performance modeling, which enables them to efficiently compare configurations and are able to predict long-term performance, so the company can make informed decisions on configuration changes before actual deployment. Thus, it can be used to estimate how the system will perform without executing actual experiments.

In fact, the performance model can be replaced with our subject application, in image 3.1 in Chapter 3, and then the benchmarking becomes a decision support system. The transformation of a benchmark into a decision support system involves enhancing its capabilities beyond its traditional role of merely measuring performance or comparing systems. The benchmark initially collects performance data from various experiments, helping assess the system's behavior under different conditions. However, rather than stopping at data collection, the benchmark starts to analyze and interpret this data to derive meaningful conclusions about system performance by replacing the subject application with a performance model.

5.1 Integration the benchmarking with development environment : enhancing decision support

The integration of benchmarking with a performance model is an approach used to optimize system performance, evaluate different configurations, and make informed decisions. With benchmarking, we can run a variety of experiments on a system or application to measure its performance under specific workloads or conditions. These benchmarks serve as a performance baseline and provide valuable insights into how the system performs. On the other hand, a performance model takes input parameters such as workload characteristics, system configurations, and resource allocation as inputs and predicts the system's performance metrics (e.g., response time, throughput). It shows how a system behaves under different configurations, workloads, and parameters. It enables the prediction of system performance without physically executing the experiments. Therefore, integrating benchmarking and a performance model can offer valuable insights and alleviate the necessity for extensive infrastructure deployment, enabling a holistic approach to performance evaluation. Thus, it becomes possible to simulate and analyze the performance of a given system or process without the need for execution in a real-world setting (Figure 5.1).

For instance, when faced with a question involving configuration information for two scenarios A and B, the experimenter does not need to undertake the entire experiment, which would typically require more time and resources to attain higher accuracy. The integration of benchmarking and performance models, allows experimenters to predict system behavior under various configurations and workloads without actually executing them, providing valuable insights into how the system can perform in different settings, and which configuration would be better in terms of desired performance metrics.

Therefore, we can leverage the model as a symbolic executor, allowing us to obtain the desired output by inputting the necessary parameters. In fact, symbolic execution is a technique [96] used to explore and reason about all possible paths of a program's execution by symbolically representing inputs, variables, and program states rather than using concrete values. This integration enables us to employ the benchmark as a decision support system. This integration supports decision-making related to system optimization, resource allocation, and configuration choices. They aid in making informed choices about how to enhance system performance and efficiency while minimizing costs and risks.

By running the model as a symbolic executor, we can provide input parameters to the model symbolically and obtain output predictions without the need for infrastructure or executing the actual application. This approach allows for faster analysis and decision-making based

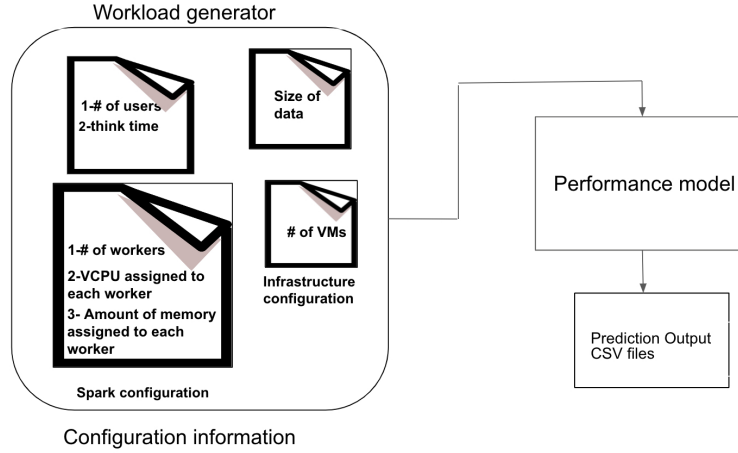


FIGURE 5.1 Integration approach components

on the symbolic execution of the model.

In essence, the model acts as a comprehensive simulator, facilitating a thorough assessment of parameterization. In fact, for the SparkPerf benchmarking platform, we needed the Docker containers to be deployed and then run the experiment, and then get the results. So integrating the model into the benchmark would mean we do not need any kind of infrastructure.

Notably, during the system's design phase, we lack access to runtime parameters such as CPU consumption, RAM consumption, and network usage. However, we possess information about the specific application being employed. For instance, we can specify the application type, such as random forest or clustering, as well as the desired data size, whether in megabytes or gigabytes. Consequently, we input this data into the model alongside resource requirements like the number of workers, amount of RAM, and CPU cores. By executing the benchmark with the integrated model, we retrieve vital performance measurements such as average latency and throughput.

Thus, it is possible to extend our work by giving all of this configuration information like the number of Spark workers, number of users, size of data, and so on to the model running the benchmark using the model, and then getting the response, which would be our average response time and our average throughput. Using this framework, we can explore various scenarios through what-if analyses by quickly manipulating the model's input parameters. Notably, while our regular experiments typically last approximately 30 minutes, the model's execution time is significantly reduced to mere seconds. Therefore, we can connect the workload generator to the model. In other words, we can have a time series prediction scenario where the workload sends inputs to the model, and we have a time series of predictions.

Hence, even when considering a substantial number of data points over time, each execution at every time point requires only milliseconds. Consequently, if we were to examine 100 data points, the total execution time would amount to only a few seconds.

Here is the summary of the key points :

- Integration of the benchmark and the performance model :

By integrating the model as a symbolic executor within the benchmark, the system can operate without requiring infrastructure deployment. The model takes the input parameters and generates the desired outcome, eliminating the need for running experiments and obtaining results through Docker containers.

- Parameterization of the performance model :

When designing the system, model input variables are required to be determined [1]. While runtime parameters such as CPU consumption, RAM consumption, or network usage may not be available during the system design phase, information about the type of application being run and the size of the data can be provided. This allows the model to estimate performance metrics like response time and throughput.

- What-if scenario analysis :

The model enables conducting what-if scenarios quickly. By adjusting the resources, such as the number of workers, amount of RAM, or CPU cores, and replaying the scenarios using the model, one can assess the impact on performance. This analysis helps in making informed decisions about resource allocation or scheduling strategies, aiming to achieve lower latency or optimize data scientist workflows.

- Rapid model execution :

Compared to regular experiments, the model execution time is significantly reduced to a few seconds. Even with multiple data points over time, the model's execution time remains within a few seconds. This speed enables quick exploration of various scenarios and facilitates efficient decision-making.

To utilize our performance models in the proposed approach, we can follow these steps :

- Replacement with the Spark application :

Replacing the trained machine learning model with the application and then the benchmark system act as a decision-support system. Therefore, we enable the model to take the place of the actual application. This integration allows the model to replace the actual application and provide performance predictions based on input configurations and workloads.

- Input parameter selection :

Determining the input parameters needed for the machine learning model should include relevant features such as data size, algorithm type, resource allocation, or any other factors that influence performance, and then providing these parameters to the model for generating performance predictions.

- Performance prediction and analysis :

Utilize the integrated machine learning model to predict performance metrics based on the given input parameters. The model analyzes the input configurations and workloads to generate predictions for metrics like response time, throughput, or resource utilization. Analyze these predictions to gain insights into the expected performance outcomes and assess trade-offs between different configurations.

- Decision-making and optimization :

Acting benchmarking framework as a decision-support system and the impact of different input configurations or resource allocations on performance metrics can be evaluated, optimizing decisions by considering trade-offs between factors such as throughput, resource utilization, accuracy, or cost can be achieved.

By replacing the application with the model, the model replaces the actual application during the benchmarking process. This integration allows for the generation of performance predictions based on input configurations and workloads, providing a basis for decision-making and optimization. It is imperative to validate the accuracy of the model's predictions by comparing them with the actual application's performance, ensuring its suitability as a decision support system within the benchmarking framework. Our benchmarking framework provides an environment and a systematic framework to gather data from a variety of experiments based on the Spark platform.

Thus, this is a potential extension of our work. Enhancing data synthesis techniques and integrating the benchmark with development environments or AutoML platforms present opportunities for further refinement and integration with existing workflows, thereby expanding the system's capabilities and adaptability.

These impacts and potential contributions highlight the significance and relevance of the proposed approach within the context of performance modeling and analysis.

5.2 Impact and contribution of the work

To highlight the impact and potential contributions of the proposed approach. Here's an academic rephrasing that incorporates our points application scenarios :

The proposed approach demonstrates several notable impacts and potential contributions :

Substituting the performance model with the application : The model is replaced with the application of data science projects, and so the benchmark becomes a decision support system. This approach aims to provide rapid decision-making capabilities by leveraging the model’s ability to generate output metrics, such as response time, CPU utilization, and throughput, based on user configurations and workload requests. This concept allows for faster decision-making without the need to execute the entire experiment.

Integration with a self-adaptive system, which can be considered the impact of the integration approach : The proposed approach paves the way for creating a self-adaptive system by replacing the analysis and planning modules with the model during runtime. This integration enables on-the-fly analysis when problems occur, facilitating real-time assessment and decision-making. The purpose of this approach would be to combine the model with an analysis and planning module to create a self-adaptive system. In this system, the model could be replaced with the analysis and planning module at runtime to perform on-the-fly analysis when a problem occurs.

Decentralization and customization, which is the contribution and impact of our benchmarking framework. The ability to upload custom models and leverage domain-specific data promotes data decentralization. This aspect empowers users to adapt the system to their specific requirements, enhancing personalization and accuracy.

Streamlined experiment configuration : The abstraction of experiment configuration simplifies the experimental process, reducing complexity and technical barriers. This abstraction enables users to focus on experimental design and analysis, streamlining the overall workflow.

Realistic performance assessment : The use of an open transactional workload, as opposed to a batch workload, allows for the evaluation of longitudinal cost and performance. This consideration provides organizations with insights into the long-term implications and efficiency of running experiments, contributing to informed decision-making, which is the contribution of both our performance model and benchmarking framework.

5.2.1 Representativeness of data

In this section, we would like to discuss data representativeness within the contexts of Spark, Docker, and machine learning. Ensuring that the data we work with accurately reflects the real-world scenarios and challenges faced in Spark-based data science projects is fundamental to the reliability of our analyses and models. Similarly, in the Docker environment, data representativeness plays a pivotal role in creating consistent and reproducible application containers that mimic production conditions. Moreover, when it comes to machine learning,

the quality and representativeness of the training data directly impact the effectiveness and fairness of the models we build.

The representativeness of data, in the context of Spark and Docker, refers to the extent to which the dataset accurately reflects the real-world phenomena or the problem we are trying to model or analyze. It implies that the data we are working with is a faithful and unbiased representation of the underlying population or system.

While the importance of data representativeness in the SparkPerf framework has been outlined, it is crucial to understand how we have addressed this issue to ensure the reliability and effectiveness of our benchmarking framework built with it. First is **diverse data sources selection** : We have taken care to select diverse and relevant data sources for benchmarking within SparkPerf. **Workload Configurations** : Since SparkPerf benchmarks different machine learning tasks under various workload configurations, we may have carefully designed these configurations to simulate different scenarios. For example, we might have configured workloads to mimic low and high data volumes, different feature sets, and varying levels of data complexity. **Data for Spark cluster** In our specific context, we have applied different settings for Spark, including the specific number of CPUs assigned to each worker, the amount of memory allocated to each worker, and the number of virtual machines (VMs), a representative of data could serve several important functions : With different Spark settings, it is crucial to determine the optimal resource allocation for our specific workload. A representative dataset can help us fine-tune parameters like the number of CPUs and memory allocation for each worker to maximize performance and efficiency. Moreover, we defined different machine learning models as Spark jobs. **Data for Model Building** : As mentioned before, we conducted experiments to gather data for building machine-learning models for Spark-based projects. In this context, representativeness of data means that the data collected during these experiments accurately reflects the data distribution, patterns, and challenges that real Spark-based data science projects encounter. It ensures that the machine learning models trained on this data are robust and effective in practical scenarios. To ensure the representativeness of data for model building in Spark-based projects, several key actions can be taken. We carefully plan and design the data collection process to align with the specific objectives and challenges of Spark-based data science projects, including considering factors including data sources, data types, and the volume of data to collect. Make sure the data sources chosen are representative of the real-world scenarios our models will encounter.

5.2.2 Generalization of findings :

A key goal in machine learning is to build models that not only perform well on the training data but also generalize effectively to unseen data. Data representativeness is crucial for achieving this generalization. If the training data does not represent real-world data accurately, the models may fail to generalize and provide useful insights when applied in actual projects. Incorporating different ranges of input features is crucial for robust generalization. In fact, input features from diverse application domains provide a more comprehensive and representative dataset helping in training models that can generalize well across various tasks. Moreover, models trained on a broader spectrum of data are less likely to be skewed toward a specific domain, resulting in more accurate performance evaluations.

CHAPTER 6 CONCLUSION

6.1 Summary of works

The primary objective of this work is to develop a performance modeling and benchmarking framework specifically tailored for Spark Data Science Projects. Therefore, the research work has two primary components, including a benchmarking framework for evaluating the Spark-based machine learning pipeline and the second component is a performance modeling based on machine learning models to predict two performance metrics, including response time and throughput. The framework aims to provide researchers and practitioners with a powerful tool for evaluating the performance of Spark-based data science projects.

The benchmarking framework enables researchers and practitioners to thoroughly evaluate the efficiency and effectiveness of Spark-based machine learning pipelines. It provides a systematic approach to measure and analyze the pipeline's performance under different conditions and configurations, ensuring robust and reliable results., and offering valuable properties for researchers and practitioners. Its key features include customizability and flexibility, allowing users to test any Spark application with their own data or provided samples. Users have the flexibility to define and customize workloads according to their specific requirements, enabling evaluation under different conditions and configurations.

On the other hand, performance modeling can help estimate optimal resource allocation for different objectives, such as response time, resource consumption, or throughput. These models can assist in initial resource allocation and adjustment in case of performance issues. Various regression-based machine learning and deep learning models were applied to build such performance models. Thus, based on the experimental evaluation done through benchmarking, we are able to build a performance model based on machine learning and deep learning.

By conducting experimental evaluations and benchmarking, we can gather data on the performance of our performance models of data science projects across different scenarios and datasets. This data is then used to train our performance model.

Moreover, the performance model can be integrated into benchmarking for having a decision-support system.

Overall, the Performance modeling, benchmarking framework, and decision support system provide the specific needs of Spark Data Science Projects, empowering researchers and practitioners with a powerful toolset for effectively evaluating and optimizing the performance

of Spark-based data science tasks. Our benchmarking framework provides an environment and a systematic framework to gather data from a variety of experiments derived from the Spark platform and our performance modeling represents a model to explore how well it will perform on new, unseen data before deploying it in a real-world setting, and finally, the decision support system can help organizations in making informed and effective decisions.

6.2 Limitation

The potential limitation of SparkPerf is its generalizability.

- Limited Generalizability

While SparkPerf has shown its applicability through two case studies involving various ML models, a potential limitation lies in its generalizability. However, the platform already supports and has been tested on many different models successfully.

- Specific to Apache Spark Ecosystem

The specialization of SparkPerf for the Apache Spark ecosystem may be considered a limitation but limited only to its implementation. However, the benchmark’s design can potentially be applied to other MapReduce platforms with some modifications, and the use of containers can make it easy to port SparkPerf on other platforms as well.

- Monitoring and Configuration Focus

Currently, SparkPerf’s monitoring and configuration primarily concentrate on performance and resource consumption metrics concerning the workload and deployment infrastructure. It does not take into account parameters that are specific to the machine learning model and its implementation, including hyperparameters. This may limit the platform’s ability to provide insights into model-specific performance characteristics.

The potential limitation of our machine learning models can be :

- Limited Generalization :

The model may perform well within the range of values seen during training, but it might struggle to generalize to unseen data that falls outside of that range. For instance, if the model was trained on data with a specific range of the number of online users or VMs, it may not accurately predict response time and throughput for scenarios with significantly different numbers of users or VMs. To mitigate this problem we collect a diverse and representative dataset during training that covers a

wide range of input values.

In fact, generalization of findings refers to the process of extending the conclusions and insights drawn from a specific set of data or a particular study to a broader context. It involves making inferences or predictions that go beyond the specific data or sample under investigation and applying them to a larger population or a wider range of scenarios.

Scope Expansion : When we conduct a study or experiment, we typically collect and analyze data from a specific group, sample, or set of conditions. Generalization involves expanding the scope of our conclusions to encompass a larger or more diverse population, setting, or situation.

Inference : To generalize findings, we infer that the patterns, relationships, or trends observed in our data are not unique to the specific cases studied but are likely to hold true in similar situations. **Statistical Significance :** In quantitative research, generalization often involves assessing the statistical significance of our findings. If our results are statistically significant, it suggests that the patterns observed are likely to hold true beyond our sample, increasing confidence in generalizing the findings. To mitigate the generalization issue in our work, we consider a validation set to evaluate the performance of a trained model.

6.3 Conclusion

In this thesis, we developed a machine learning benchmarking framework and performance modeling approach to evaluate the performance of Spark-based data science applications. The framework's effectiveness is demonstrated through its successful loading of user applications onto the Apache Spark platform, confirming its customizability with a wide range of application types. Additionally, the framework's capability to generate synthetic data enables users to conduct rigorous testing and evaluation. Hence, we utilized multiple dimensions of metrics obtained from our experiments through the benchmarking tool, including Spark topologies, cloud infrastructure, input data size, and workload size to build the performance models. The models we employed were a combination of regression-based machine learning and deep learning techniques. The evaluation of these models revealed that Random Forest, Decision Tree, and XGBoost models performed exceptionally well in predicting response time metrics, outperforming other models. On the other hand, the AdaBoost and LSTM models excelled in predicting throughput metrics.

REFERENCES

- [1] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems benchmarking : for scientists and engineers*. Springer, 2020, vol. 1.
- [2] U. M. Fayyad, A. Candel, E. Ariño de la Rubia, S. Pafka, A. Chong, and J.-Y. Lee, “Benchmarks and process management in data science : Will we ever get over the mess ?” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 31–32.
- [3] A. Watson, D. S. V. Babu, and S. Ray, “Sanzu : A data science benchmark,” in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 263–272.
- [4] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, “Bigdatabench : A big data benchmark suite from internet services,” in *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)*. IEEE, 2014, pp. 488–499.
- [5] W. Xiong, Z. Yu, Z. Bei, J. Zhao, F. Zhang, Y. Zou, X. Bai, Y. Li, and C. Xu, “A characterization of big data benchmarks,” in *2013 IEEE international conference on big data*. IEEE, 2013, pp. 118–125.
- [6] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, “Bigbench : Towards an industry standard benchmark for big data analytics,” in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, 2013, pp. 1197–1208.
- [7] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, “Sparkbench : a comprehensive benchmarking suite for in memory data analytic platform spark,” in *Proceedings of the 12th ACM international conference on computing frontiers*, 2015, pp. 1–8.
- [8] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, “The hibench benchmark suite : Characterization of the mapreduce-based data analysis,” in *2010 IEEE 26th International conference on data engineering workshops (ICDEW 2010)*. IEEE, 2010, pp. 41–51.
- [9] M. Assefi, E. Behraves, G. Liu, and A. P. Tafti, “Big data machine learning using apache spark mllib,” in *2017 ieee international conference on big data (big data)*, pp. p.p 3492–3498(2017).
- [10] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, “Big data analytics on apache spark,” *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

- [11] M. Capuccini, “Enabling scalable data analysis on cloud resources with applications in life science,” Ph.D. dissertation, Acta Universitatis Upsaliensis, 2019.
- [12] M. Y. Santos, J. Oliveira e Sá, C. Costa, J. Galvão, C. Andrade, B. Martinho, F. V. Lima, and E. Costa, “A big data analytics architecture for industry 4.0,” in *Recent Advances in Information Systems and Technologies : Volume 2 5*. Springer, 2017, pp. 175–184.
- [13] S. Sagiroglu and D. Sinanc, “Big data : A review,” in *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, 2013, pp. 42–47.
- [14] Y. A. Qadri, A. Nauman, Y. B. Zikria, A. V. Vasilakos, and S. W. Kim, “The future of healthcare internet of things : a survey of emerging technologies,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1121–1167, 2020.
- [15] T. R. Babu, M. N. Murty, and S. Subrahmanya, “Compression schemes for mining large datasets,” *London : Springer-Verlag*, vol. 10, pp. 978–1, 2013.
- [16] T. Ivanov, “Classifying, evaluating and advancing big data benchmarks,” Ph.D. dissertation, Goethe University Frankfurt, Frankfurt am Main, Germany, 2019.
- [17] T. Rabl, N. Raghunath, M. Poess, M. Bhandarkar, H.-A. Jacobsen, and C. Baru, “Advancing big data benchmarks,” 2014.
- [18] F. Cauteruccio, P. L. Giudice, L. Musarella, G. Terracina, D. Ursino, and L. Virgili, “A lightweight approach to extract interschema properties from structured, semi-structured and unstructured sources in a big data scenario,” *International Journal of Information Technology & Decision Making*, vol. 19, no. 03, pp. 849–889, 2020.
- [19] S. Erevelles, N. Fukawa, and L. Swayne, “Big data consumer analytics and the transformation of marketing,” *Journal of business research*, vol. 69, no. 2, pp. 897–904, 2016.
- [20] P. Raj, A. Raman, D. Nagaraj, and S. Duggirala, “High-performance big-data analytics,” *Computing Systems and Approaches (Springer, 2015)*, vol. 1, 2015.
- [21] D. Fisher, “Big data exploration requires collaboration between visualization and data infrastructures,” in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 2016, pp. 1–5.
- [22] N. Bikakis, “Big data visualization tools,” *arXiv preprint arXiv :1801.08336*, 2018.
- [23] R. K. Chawda and G. Thakur, “Big data and advanced analytics tools,” in *2016 symposium on colossal data analysis and networking (CDAN)*. IEEE, 2016, pp. 1–8.
- [24] G. Gottlob, G. Grasso, D. Olteanu, and C. Schallhart, *Big Data*. Springer, 2013.
- [25] M. U. Javaid, A. A. Kanoun, F. Demesmaeker, A. Ghrab, and S. Skhiri, “A performance prediction model for spark applications,” in *International Conference on Big Data*. Springer, 2020, pp. p.p 13–22(2020).

- [26] P. S. Deshpande, S. C. Sharma, S. K. Peddoju, P. S. Deshpande, S. C. Sharma, and S. K. Peddoju, "Predictive and prescriptive analytics in big-data era," *Security and data storage aspect in cloud computing*, pp. 71–81, 2019.
- [27] F. Savva, C. Anagnostopoulos, and P. Triantafillou, "Explaining aggregates for exploratory analytics," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 478–487.
- [28] S. Allam, "Usage of hadoop and microsoft cloud in big data analytics : An exploratory study," *Sudhir Allam.(2018). USAGE OF HADOOP AND MICROSOFT CLOUD IN BIG DATA ANALYTICS : AN EXPLORATORY STUDY. International Journal of Innovations in Engineering Research and Technology*, vol. 5, no. 10, pp. 27–32, 2018.
- [29] S. Poornima and M. Pushpalatha, "A survey of predictive analytics using big data with data mining," *International journal of bioinformatics research and applications*, vol. 14, no. 3, pp. 269–282, 2018.
- [30] D. Aneato, "Strategies to implement big data analytics in telecommunications organizations," Ph.D. dissertation, Walden University, 2020.
- [31] R. K. Chunduri and A. K. Cherukuri, "Distributed three-way formal concept analysis for large formal contexts," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 141–156, 2023.
- [32] F. Pan, Y. Yue, J. Xiong, and D. Hao, "I/o characterization of big data workloads in data centers," in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware : 4th and 5th Workshops, BPOE 2014, Salt Lake City, USA, March 1, 2014 and Hangzhou, China, September 5, 2014, Revised Selected Papers 4*. Springer, 2014, pp. 85–97.
- [33] B. Furht and F. Villanustre, "Big data technologies and applications," 2016.
- [34] M. K. Frank, *Model-Based Performance Prediction for Concurrent Software on Multicore Architectures : A Simulation-Based Approach*. KIT Scientific Publishing, 2022.
- [35] P. Neumann and J. Kunkel, "High-performance techniques for big data processing," in *Knowledge Discovery in Big Data from Astronomy and Earth Observation*. Elsevier, 2020, pp. 137–158.
- [36] R. Han, L. K. John, and J. Zhan, "Benchmarking big data systems : A review," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 580–597, 2017.
- [37] P. Dreher, C. Byun, C. Hill, V. Gadepally, B. Kuszmaul, and J. Kepner, "Pagerank pipeline benchmark : Proposal for a holistic system benchmark for big-data platforms," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 929–937.

- [38] (2023) Tpc. <http://www.tpc.org/tpce/>. [Online]. Available : <http://www.tpc.org/tpce/>
- [39] T. Ivanov and R. Singhal, “Tutorial on benchmarking big data analytics systems,” in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2020, pp. 50–53.
- [40] T. G. Armstrong, V. Ponnkanti, D. Borthakur, and M. Callaghan, “Linkbench : a database benchmark based on the facebook social graph,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 1185–1196.
- [41] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafae, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, “Clearing the clouds : A study of emerging workloads on modern hardware,” in *Proceedings the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012)*, Mar 2012.
- [42] (2023) hadoop. <https://hadoop.apache.org/docs/r1.2.1/gridmix.html>. [Online]. Available : <https://hadoop.apache.org/docs/r1.2.1/gridmix.html>
- [43] R. Lu, G. Wu, B. Xie, and J. Hu, “Stream bench : Towards benchmarking modern distributed stream computing frameworks,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 69–78.
- [44] R. Singhal and P. Singh, “Performance assurance model for applications on spark platform,” in *Performance Evaluation and Benchmarking for the Analytics Era : 9th TPC Technology Conference, TPCTC 2017, Munich, Germany, August 28, 2017, Revised Selected Papers 9*. Springer, 2018, pp. 131–146.
- [45] N. Luo, Z. Yu, Z. Bei, C. Xu, C. Jiang, and L. Lin, “Performance modeling for spark using svm,” in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 127–131.
- [46] A. Maros, F. Murai, A. P. C. da Silva, J. M. Almeida, M. Lattuada, E. Gianniti, M. Hosseini, and D. Ardagna, “Machine learning for performance prediction of spark cloud applications,” in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. p.p99–106(2019).
- [47] D. de Oliveira, F. Porto, C. Boeres, and D. de Oliveira, “Towards optimizing the execution of spark scientific workflows using machine learning-based parameter tuning,” *Concurrency and Computation : Practice and Experience*, vol. 33, no. 5, p. p.p e5972(2021).
- [48] R. Myung and H. Yu, “Performance prediction for convolutional neural network on spark cluster,” *Electronics*, vol. 9, no. 9, p. p.p1340(2020).
- [49]

- [50] G. Cheng, S. Ying, B. Wang, and Y. Li, “Efficient performance prediction for apache spark,” *Journal of Parallel and Distributed Computing*, vol. 149, pp. p.p 40–51(2021), 2021.
- [51] X. He, K. Zhao, and X. Chu, “Automl : A survey of the state-of-the-art,” *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.
- [52] (2023) Tpc. <https://www.tpc.org/>. [Online]. Available : <https://www.tpc.org/>
- [53] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Damos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, “Mlperf : An industry standard benchmark suite for machine learning performance,” *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.
- [54] P. Gijssbers, M. L. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren, “Amlb : an automl benchmark,” *arXiv preprint arXiv :2207.12560*, 2022.
- [55] M.-A. Zöllner and M. F. Huber, “Benchmark and survey of automated machine learning frameworks,” *Journal of artificial intelligence research*, vol. 70, pp. 409–472, 2021.
- [56] Á. B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero, “Using machine learning to optimize parallelism in big data applications,” *Future Generation Computer Systems*, vol. 86, pp. p.p1076–1092(2018).
- [57] Q. Wu and V. V. Datla, “On performance modeling and prediction in support of scientific workflow optimization,” in *2011 IEEE World Congress on Services*, pp. p.p161–168(2011).
- [58] E. Ipek, B. R. d. Supinski, M. Schulz, and S. A. McKee, “An approach to performance prediction for parallel applications,” in *European Conference on Parallel Processing*, pp. p.p196–205(2005).
- [59] Z. Yu, Z. Bei, H. Zhang, W. Xiong, L. Eeckhout, C. Xu, and S. Feng, “Rfhoc : A random-forest approach to auto-tuning hadoop’s configuration. parallel and distributed systems,” *IEEE Transactions on*, 2015.
- [60] N. Ahmed, A. L. Barczak, M. A. Rashid, and T. Susnjak, “Runtime prediction of big data jobs : performance comparison of machine learning algorithms and analytical models,” *Journal of Big Data*, vol. 9, no. 1, p. 67, 2022.
- [61] D. Didona, F. Quaglia, P. Romano, and E. Torre, “Enhancing performance prediction robustness by combining analytical modeling and machine learning,” in *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, 2015, pp. 145–156.
- [62] Y. LeCun, C. Cortes, C. Burges *et al.*, “Mnist handwritten digit database,” 2010.
- [63] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, vol. 14, pp. 131–164, 2009.

- [64] A. Computing *et al.*, “An architectural blueprint for autonomic computing,” *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.
- [65] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, “Starfish : A self-tuning system for big data analytics.” in *Cidr*, vol. 11, no. 2011, pp. p.p261–272(2011).
- [66] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, “Apache spark : a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. p.p 56–65(2016).
- [67] M. Hamstra and M. Zaharia, *Learning Spark : lightning-fast big data analytics*. O’Reilly & Associates, 2013.
- [68] S. KOUNEV, *SYSTEMS BENCHMARKING : For Scientists and Engineers*. SPRINGER, 2021.
- [69] S. Ghari, M. Fokaefs, and H. Li, “Sparkperf : A machine learning benchmarking framework for spark-based data science projects,” 2023.
- [70] C. Barna, H. Khazaei, M. Fokaefs, and M. Litoiu, “Delivering elastic containerized cloud applications to enable devops,” in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 65–75.
- [71] M. Fokaefs, C. Barna, R. Velede, M. Litoiu, J. Wigglesworth, and R. Mateescu, “Enabling devops for containerized data-intensive applications : an exploratory study,” in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, 2016, pp. 138–148.
- [72] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, “Benchmarking distributed stream data processing systems,” in *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 2018, pp. 1507–1518.
- [73] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn : Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. p.p2825–2830(2011).
- [74] A. Dasgupta, Y. V. Sun, I. R. König, J. E. Bailey-Wilson, and J. D. Malley, “Brief review of regression-based and machine learning methods in genetic epidemiology : the genetic analysis workshop 17 experience,” *Genetic epidemiology*, vol. 35, no. S1, pp. S5–S11, 2011.
- [75] S. Weisberg, *Applied linear regression*. John Wiley & Sons, 2005, vol. 528.
- [76] W. XingFen, Y. Xiangbin, and M. Yangchun, “Research on user consumption behavior prediction based on improved xgboost algorithm,” in *2018 IEEE international conference*

- on big data (Big Data)*. IEEE, 2018, pp. 4169–4175.
- [77] M. M. Hafez, M. E. Shehab, E. El Fakharany, and A. E. F. Abdel Ghfar Hegazy, “Effective selection of machine learning algorithms for big data analytics using apache spark,” in *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016 2*. Springer, 2017, pp. 692–704.
 - [78] N. Altman and M. Krzywinski, “Ensemble methods : bagging and random forests,” *Nature Methods*, vol. 14, no. 10, pp. 933–935, 2017.
 - [79] M. Xu, L. Wang, J. Ren, and S. Poslad, “Use of lstm regression and rotation classification to improve camera pose localization estimation,” in *2020 IEEE 14th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. IEEE, 2020, pp. 6–10.
 - [80] G. Van Houdt, C. Mosquera, and G. Nápoles, “A review on the long short-term memory model,” *Artificial Intelligence Review*, vol. 53, pp. 5929–5955, 2020.
 - [81] X. Feng, G. Ma, S.-F. Su, C. Huang, M. K. Boswell, and P. Xue, “A multi-layer perceptron approach for accelerated wave forecasting in lake michigan,” *Ocean Engineering*, vol. 211, p. 107526, 2020.
 - [82] J. Behera, A. K. Pasayat, and H. Behera, “Covid-19 vaccination effect on stock market and death rate in india,” *Asia-Pacific Financial Markets*, vol. 29, no. 4, pp. 651–673, 2022.
 - [83] D. Chicco, M. J. Warrens, and G. Jurman, “The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation,” *PeerJ Computer Science*, vol. 7, p. e623, 2021.
 - [84] M. Kuhn, K. Johnson *et al.*, *Applied predictive modeling*. Springer, 2013, vol. 26.
 - [85] A. J. Scott and M. Knott, “A cluster analysis method for grouping means in the analysis of variance,” *Biometrics*, pp. p.p 507–512(1974).
 - [86] A. Abdelsalam, “Efficient fpga-based inference architectures for deep learning networks,” Ph.D. dissertation, Ecole Polytechnique, Montreal (Canada), 2019.
 - [87] A. Rantanen, J. Salminen, F. Ginter, and B. J. Jansen, “Classifying online corporate reputation with machine learning : a study in the banking domain,” *Internet Research*, vol. 30, no. 1, pp. 45–66, 2020.
 - [88] S. Basso, A. Ceselli, and A. Tettamanzi, “Random sampling and machine learning to understand good decompositions,” *Annals of Operations Research*, vol. 284, pp. 501–526, 2020.
 - [89] A. Zien, N. Krämer, S. Sonnenburg, and G. Rätsch, “The feature importance ranking measure,” in *Machine Learning and Knowledge Discovery in Databases : European*

- Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part II 20.* Springer, 2009, pp. 694–709.
- [90] A. Altmann, L. Tološi, O. Sander, and T. Lengauer, “Permutation importance : a corrected feature importance measure,” *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.
 - [91] K. Javaid, A. Siddiqi, S. A. Z. Naqvi, A. Ditta, M. Ahsan, M. Khan, T. Mahmood, and M. A. Khan, “Explainable artificial intelligence solution for online retail,” *CMC-COMPUTERS MATERIALS & CONTINUA*, vol. 71, no. 3, pp. 4425–4442, 2022.
 - [92] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “An empirical comparison of model validation techniques for defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.
 - [93] R. Nambiar and M. Poess, “Performance evaluation and benchmarking : Traditional to big data to internet of things,” in *Proc. 7th TPC Technol. Conf.(TPCTC)*, vol. 9508. Springer, 2017.
 - [94] G. Song, Z. Meng, F. Huet, F. Magoules, L. Yu, and X. Lin, “A hadoop mapreduce performance prediction method,” in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. p.p820–825(2013).
 - [95] R. F. Woolson, “Wilcoxon signed-rank test,” *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.
 - [96] T. Margaria and B. Steffen, *Leveraging Applications of Formal Methods, Verification, and Validation : 4th International Symposium on Leveraging Applications, ISO LA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I.* Springer, 2010, vol. 6415.