11th International Conference on Electrical, Electronics and Computer Engineering (IcETRAN), Nis, 3-6 June 2024

Energy-consumption focused multi-objective tuning of Apache Spark

Muhammed Maruf Öztürk
Computer Engineering
Faculty of Engineering and Natural Science, Suleyman Demirel University
Turkey
muhammedozturk@sdu.edu.tr

Valentina Nejkovic
Computer Science
Faculty of Electronic Engineering,
University of Niš, Serbia
valentina@elfak.i.ac.rs

***Abstract*—There exist various ways to optimize big data processing frameworks. The main purpose of these ways is to find optimal configuration settings by applying a set of iterative optimization steps. Apache Spark, which is one of the common big data processing engines, is suitable for multi-objective tuning methods. However, the majority of tuning techniques are adapted either regarding the success of speedup or machine learning methods. Different from preceding studies, this study proposes a new energy-focused optimization algorithm for Spark (EFOS) which is evaluated on the MLlib library of Spark. EFOS is devised by considering the data processing processing abilities of Spark. The findings of the study indicate that Bayes is the most preferable MLlib algorithm to shorten optimization time. The obtained results indicate that unsupervised learning necessitates less time yet more CPU usage compared with supervised learning in the tuning process.**

***Index Terms*—Apache Spark, energy-consumption, tuning.**

I. Introduction

Apache Spark is a cluster computing framework that supports various programming languages including Scala, Java, Python, and R [1]. It provides more than 200 configurable parameters that can be tuned in a scalable manner [2]. Thus, tuning is inevitable for such intricate software systems. To achieve a significant performance improvement, a tradeoff should be found while trying different configurations [3]. To that end, the configurable parameters of Spark are exposed to a specific optimization process. Since manual tuning is a time-consuming and effort-intensive process, automated frameworks are generally preferred to alleviate the burden that stems from optimization. However, these frameworks adopt some objectives including time, speedup, and memory rather than energy consumption [4]–[6].

In a distributed data processing framework, the chosen algorithm should be compatible with the devised architecture [7]. For instance, training data groups retrieved from different cluster machines are individually evaluated to merge them for a specific purpose such as classification. MLlib is a comprehensive machine learning library [8] developed for Apache Spark. It mainly works with Resilient Distributed Data (RDD) data format [9] which is devised for distributed data processing. MLlib provides both data and model-based parallelism that allows users to perform fast operations.

There exist various approaches that can be used for optimizing parameters of Apache Spark: machine learning-based [10]–[13], manual tuning [14], statistical inferences made through log file-based [15]. The main purpose of these approaches is to improve the performance of Apache Spark, thereby leveraging various objective functions. The success of an objective function can be evaluated via a few criteria. Those generally consist of elements such as memory usage and data processing speed except for energy consumption.

Since high energy physics (HEP) experiments have to work with large-scale data sets, they utilize big data frameworks such as Apache Spark. Therefore, in a HEP experiment, data processing throughput may exceed 200TB/s [16]. Although energy consumption, on the other hand, is not a big deal for personal computers, the objective functions of optimization methods should include that criterion in the near future when the data processing speed is close to that of HEP experiments. To achieve a dramatic improvement, some parallel message-passing libraries such as Message Passing Interface (MPI) should be utilized except for the default properties of Apache Spark. A remarkable time saving could be achieved by combining Spark with MPI in the related programming model that is called Blaze [17]. Figure 1 validates the findings obtained via the conjugate gradient (CG) algorithm which exploits matrices. CG is an iterative approach that solves linear equations [18].

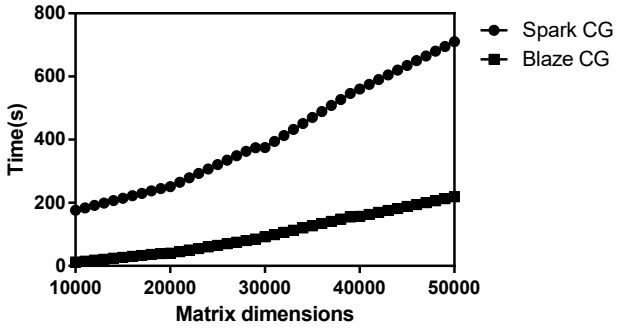

Fig. 1. Spark CG vs. Blaze CG.

Devising optimization methods merely considering speedup becomes superficial when it comes to systems requiring high

energy such as HEP. Instead, there is a need for developing multi-objective methods [19] focusing on various computational resources such as energy and memory consumption.

To fill the research gap detected above, in this study, a new multi-objective optimization algorithm, which considers energy consumption as a major criterion, is proposed. In the first phase, the algorithm traces energy consumption footprints online. Thereafter, the hyperparameter settings are configured through a pre-determined threshold. In the second phase, the configuration is changed depending on the success of the MLlib method to achieve the ultimate hyperparameter set in the iterations. Figure 2 presents the main steps of the proposed method.

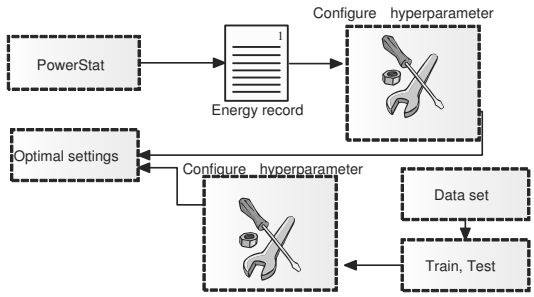

Fig. 2. Overview of the proposed method.

This paper makes the contributions as follows: 1) A new energy-focused multi-objective optimization algorithm is proposed, 2) The responses of the algorithm depending on the energy consumption of MLlib are evaluated, and 3) The relationship between CPU usage and the computational burden is investigated.

The remainder of the paper is organized as follows: Section 2 summarizes the related work. Section 3 elucidates the proposed method. Section 4 presents the results. Last, Section 5 concludes the paper and discusses future studies.

II. Literature review

Inherently, RDD is not suitable for high computational experiments. Because some transactions such as partial wave analysis (PWA) and lattice quantum chromodynamics (LQCD) necessitate a message-passing interface between the processes. There exist various studies performing a transformation on Spark for HEP. In Xia et. al's work [17], a new process planning policy is developed, thereby combining Spark with OpenMPI. They were able to achieve 70% performance improvement compared with the traditional Spark programming model. MPI is remarkably faster than Spark in reading and summarizing operations [20]. On the other hand, in terms of scaling behavior, Spark is a preferable framework.

Loading balance is of great importance to use resources efficiently and save energy [21]. Thus, energy consumption can be reduced by developing algorithms seeking load balancing in big data processing. To that end, Hibench is one of the most well-known benchmark suites employed for testing resource usage [22]. The effectiveness of HEP in distributed data processing is evaluated via evaluating some criteria including CPU usage, speedup, and memory usage [23]. Read/write operation performed on the data type called HDF5 is fundamental for HEP. Therefore, execution time is calculated depending on the number of processes, the data compression format, and the type of transaction (read/write). The obtained findings indicate that chunk size does not have a direct effect on the performance [24].

To what extent the workload is responsible for energy consumption was investigated in the preceding studies. In one of them, a profiling framework was developed for Kafka and Flink to observe the energy consumption depending on the number of nodes and processes [25].

There is a direct relationship between CPU usage and Spark energy consumption. It thus aims to find cases in which CPU frequency becomes high and the execution time is short. However, that configuration should be carefully done by considering the number of CPU cores. In doing so, 60% of energy saving may be achieved [26]. Preceding studies have shown that smart grid architectures have stability in terms of energy profiling. It was detected that gradient-boosted decision tree has an accuracy equal to or greater than those of Random Forest and CNN. Changing design models is an alternative way to reduce energy consumption [27]. A design model namely Visitor was applied in Java and C++ codes [24]. It was able to reduce energy consumption by up to 7%. That result points out that refactoring-based techniques may open new avenues for energy-saving methods.

The tasks allocated for Spark jobs may be planned by creating specific clusters considering workload distribution to reduce energy consumption dramatically. Shi et al. [28] clustered Spark nodes, thereby utilizing historical data usage. To achieve remarkable energy saving, task scheduling-based workload planning should be conducted on large-scale data sets. Otherwise, task-based scheduling becomes much more time-consuming. In addition to this, instead of calculating the energy consumption of all the data sets, the prediction may be performed by exploiting a specific sub-instance group [29].

III. Methods

A. Energy measurement

In this section, the tools preferred in measuring energy consumption and which one is chosen for the experiment are explained.

Intel Power Gadget: is a profiling tool that can used both in macOS and Windows operating systems [30]. The main advantage of this tool is its ability to trace CPU usage, frequency, temperature, and power (W) through a graph that can be converted to a .csv file. On the other hand, Intel Power Gadget is not compatible with Linux systems.

PowerStat: is developed on Linux and can record energy consumption as watt [31]. This tool supports the Running Average Power Limit (RAPL) interface of Intel, thereby utilizing two types of registers including MSR_PKG_Energy_Status3 and MSR_PP0_Energy_Status4. It mainly relies on telemetry and thermal control algorithms.

PowerTop: is an alternative profiling framework that works on Linux systems [32]. Firstly, it requires a calibration that finds a compromise between the machine and profiling. After that, energy consumption data is presented in various formats encompassing idle, device, and tunable. Further, it provides HTML reporting.

Perf: is a specific program that can be executed on machines having Intel processor and Linux operating system [33]. Energy consumption data is saved as joule along with the measurement count. Some components including RAM, GPU, and core can be evaluated in terms of energy consumption.

In the experiment, PowerStat is preferred for evaluating the threshold optimization value of energy consumption. The main reason PowerStat is chosen is its advantageous speed in read/write operations since it performs on .txt files. Further, it allows users to determine time periods employed for measuring energy consumption.

B. Proposed algorithm

EFOS takes a .txt file produced by PowerStat. An initial hyperparameter value h_1 is given to the algorithm for starting optimization. Step 7 takes an energy record from the file to configure the hyperparameter. Steps 8-11 calculates the local optimal h_2 depending on the threshold energy e_t. Step 14 assigns an arbitrary performance value, which is assigned to an arbitrary performance 70% of accuracy that should be exceeded during T iterations. Steps 15-23 calculate the second local optimal h_3, thereby utilizing a function $reAssign$ which either increases or decreases the local optimal by controlling the change of the hyperparameter. Last, Step 24 returns the ultimate performance measure along with the final value of the hyperparameter. To create experimental codes, $sparklyr$ and $sparktf$ libraries of R were utilized. RDD conversion and other preprocessing steps were also coded with R.

1: **Input:** energy file (.txt), inital value of hyperparameter (h_1), D (Data set), iteration (T), threshold energy (e_t)
2: **output:** optimal hyperparameter, ultimate performance
3: $D_t \leftarrow createFolds(D)$
4: $file \leftarrow read(energyfile)$
5: $L \leftarrow length(file)$
6: $h_2 \leftarrow h_1$
7: $S_d \leftarrow file[sample(1:L,1)]$
8: **while** $S_d < e_t$ **do**
9: $h_2 \leftarrow reAssign(h_2)$
10: $S_d \leftarrow file[sample(1:L,1)]$
11: **end while**
12: $h_3 \leftarrow h_1$
13: $i \leftarrow 0$
14: $result \leftarrow 0.7$
15: **while** $i < T$ **do**
16: $model \leftarrow train(D_t)$
17: $result \leftarrow test(model, D_t)$
18: **if** $(result >= max)$ **then**
19: $h_3 \leftarrow reAssign(h_3)$
20: **end if**
21: $h_{optimal} \leftarrow mean(h_3, h_2)$
22: $i \leftarrow i + 1$
23: **end while**
24: **return** $result, h_{optimal}$

Algorithm 1: EFOS.

IV. Experimental Setup

A. Datasets

Table I presents the experimental data sets. The $Dense$ data set was developed for improving neural network models in classification[1]. It has 30 numerical features along with a label compatible with binary classification. $Microsoft$ includes instances collected from a security contest and it is publicly available[2]. It has 1804 numerical features (187.83 MB) that can be used for classification experiments. $Payload$ is collected from a research project conducted by Politecnico di Milano University[3]. It has 31 numerical features which have low churn (+10,-1) compared to the other data sets. $Santander$ is generated for transaction prediction of users[4]. It has 140 numerical features in which the majority of them are floating-point numbers.

TABLE I
MLlib data sets utilized in this work.

Name	Size range	Category
Dense	175000 (instances)	Classification
Microsoft	10868 (instances)	Classification
Payload	130529 (instances)	Classification
Santander	200000 (instances)	Classification

B. Experimental Settings

The case study comprises five algorithms and their hyperparameters along with search space as follows:

- deep neural network:max_iter:10:100-step size:1, tol:1e-10:1e-06-step size:0.0001, step_size:0.0001:1-step size:0.01.
- kmeans:max_iter:10:100-step size:1, tol:1e-10:1e-06-step size:0.0001, k:2:5-step size:1.
- logistic regression:max_iter:10:100-step size:1, threshold:0.1:0.5-step size:0.0001.
- naive bayes:smoothing:1:5-step size:1, thresholds:0.1:0.5-step size:0.0001.
- random forest:max_depth:5:10-step size:1, thresholds:0.1:0.5-step size:0.0001, num_trees: 5:20-step size:1

C. Performance measures

For deciding the optimal configuration, Algorithm 1 employs accuracy measure as follows:

[1]https://www.kaggle.com/c/dense-network/data?select=train.csv
[2]https://www.kaggle.com/muhammad4hmed/malwaremicrosoftbig
[3]https://zenodo.org/record/5731597
[4]https://www.kaggle.com/datasets/lakshmi25npathi/santander-customer-transaction-prediction-dataset

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \quad (1)$$

where true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are the elements of confusion matrix used in calculating classification performance. On the other hand, total energy T_E can be formulated as follows:

$$T_E = \sum_{i=1}^{m} P_i.\triangle t \quad (2)$$

in which P represents a unique energy consumption (watt) recorded for a specific time period $\triangle t$. For m different power usages, the computational burden of CPU is:

$$CPU_{usage} = \sum_{i=1}^{m} U_i \quad (3)$$

in which U denotes the identical CPU usage of a specific task. The ultimate optimization burden O_{burden} can be calculated as in Equation 4.

$$O_{burden} = T_E + CPU_{usage} \quad (4)$$

V. Results

Optimal settings found by the proposed method are presented in Table II. It is worth noting that each algorithm relies on specific settings depending on the data set group. However, some hyperparameters such as max_iter accurately reflect the type of machine learning algorithm. For instance, GLM requires a lot of number of iterations but this case does not greatly contribute to the computational burden. On the other hand, a Deep neural network inherently depends on complex calculations due to the number of hidden layers. In this respect, it necessitates less number of iterations.

TABLE II
MLlib data sets utilized in this work.

Method	Optimal settings
Deep neural network	max_iter:55, tol:1e-6, step_size:0.003
Bayes	smoothing:2, thresholds:0.3
Random Forest	130529 (instances)
GLM	max_iter:82, threshold:0.4
Kmeans	max_iter:45, tol:1e-04, k:3

Figures 3-7 present a time analysis of the proposed method. Bayes is the most preferable technique in that it does not exceed 600 seconds up to 100 iterations as seen in Figure 3. In this algorithm, the data sets are divided into two groups depending on the size. Further, there is no threshold for time in increasing iterations. Figure 7 shows the time required for the execution of the proposed method in five MLlib algorithms. Since the Santander data set is the largest, it takes a long time to complete optimization. Separate lines are evident in Random Forest (Figure 6) and Deep Neural Network (Figure 7). On the other hand, this is not the case for GLM (Figure 4). 100 can be considered a boundary value for computational burden since time starts dramatically increasing after that value. Clustering takes less time compared to classification as shown in Figure 5 which presents a clear churn in time results.

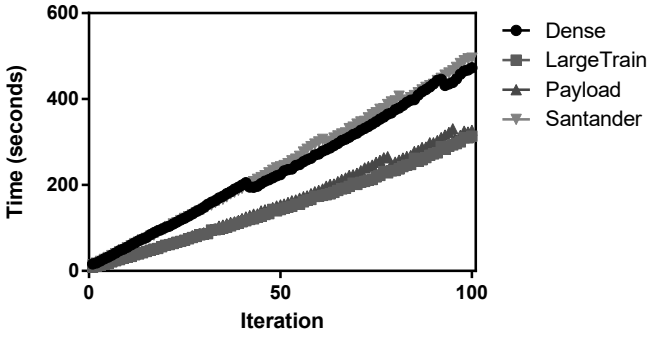

Fig. 3. Tuning time analysis of Bayes.

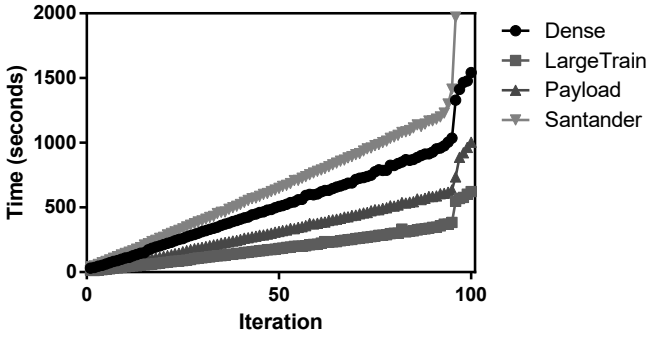

Fig. 4. Tuning time analysis of GLM.

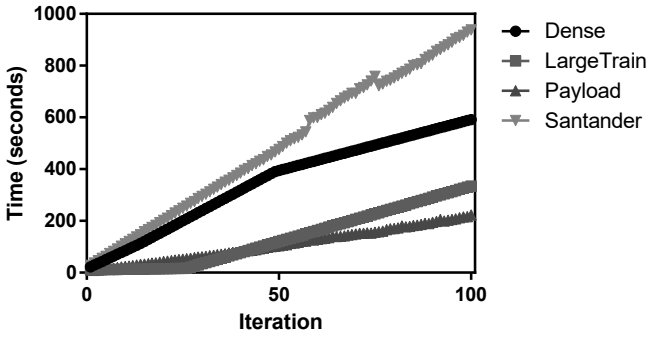

Fig. 5. Tuning time analysis of Kmeans.

Figure 8 shows CPU usage rates of the MLlib algorithms employed in the tuning. It is worth noting that Kmeans has the highest rate (up to 90%) which resulted in a remarkable computational burden. On the contrary, deep neural network, which is a supervised algorithm, has not exceeded 35%. Likewise, Random Forest is the second eligible method in terms of CPU usage. It can be concluded that unsupervised

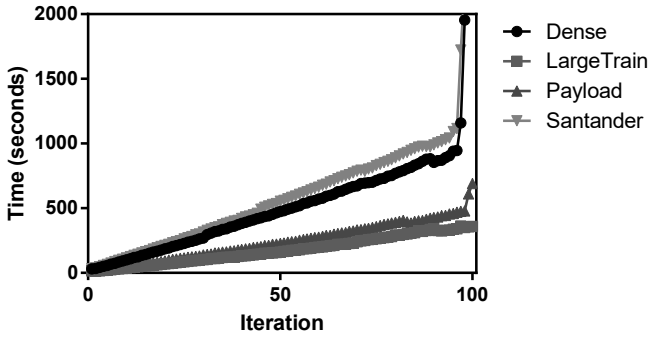

Fig. 6. Tuning time analysis of Random Forest.

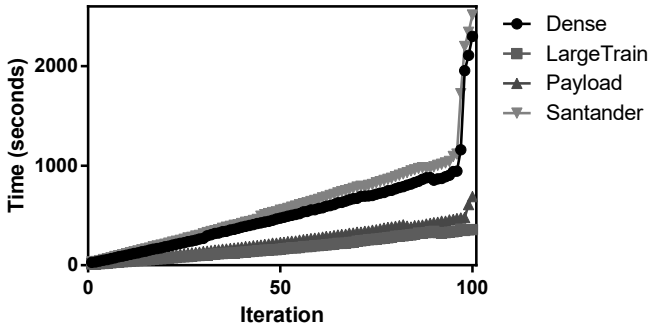

Fig. 7. Tuning time analysis of deep neural network.

methods perform more cost-effectively in Apache Spark when it comes to configuring hyperparameters.

VI. Discussion

To make a general discussion, it is inevitable to evaluate energy consumption along with CPU usage. Table III presents the ultimate optimization burden which includes mean energy consumption recorded during the tuning. GLM requires more energy than the other algorithms due to its large computational time. Unsurprisingly, Bayes consumes the lowest energy since it establishes a set of tuning processes that is pruned via a result-based inference design. Despite the fact that deep neural networks are very complex to run, they require reasonable energy (13kW) compared to the alternatives.

TABLE III
O_{burden} DETAILS OF THE EXPERIMENTAL ALGORITHMS.

Method	T_E	$CPU_{usage}\%$	O_{burden}
Deep neural network	13kW	10.2	23.2
Bayes	12kW	19.8	31.8
Random Forest	39kW	9	48
GLM	48kW	10.4	58.4
Kmeans	17kW	54.2	71.2

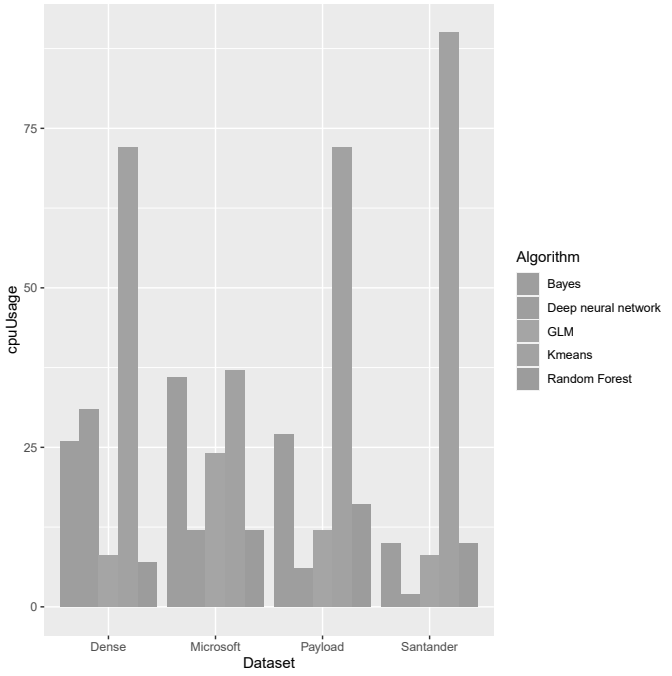

Fig. 8. CPU usage analysis of the MLlib algorithms on the experimental data sets. The measurement was performed with proc.time function of R base library. The record was obtained via Google Console cloud computing environment having Intel(R) Xeon(R) CPU @ 2.20GHz, 14 GB RAM, 4 CPU(s).

VII. Conclusion

We utilize energy footprint and accuracy to generate a multi-objective optimization algorithm for Apache Spark. This method instantly checks energy records to shrink hyperparameter space. Then, it seeks the highest performance measure depending on the predefined iteration. To validate the effectiveness of the proposed method, four classification data sets are exploited by five MLlib algorithms. The experimental findings bolster the claim that inference-based algorithms such as Bayes remarkably shorten tuning time in big data processing frameworks. Although unsupervised methods are not concerned with the features of classification data sets, they create a significant CPU burden according to the findings.

The future agenda of this paper encompasses the following avenues: 1) The energy measurement tools are either devised for specific operating systems or CPU types. A flexible energy profiling tool may be developed for Apache Spark, 2) A deductive decision-making mechanism is needed before tuning big data systems. There are various hyperparameter optimization techniques but few are effective in reducing data processing time. In this context, an elimination algorithm could be used to guide practitioners.

References

[1] 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.

[2] G. Cheng, S. Ying, B. Wang, and Y. Li, "Efficient performance prediction for apache spark," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 40–51, 2021.

[3] G.-M. Park, Y. S. Heo, and H.-Y. Kwon, "Trade-off analysis between parallelism and accuracy of slic on apache spark," in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2021, pp. 5–12.

[4] D. B. Prats, F. A. Portella, C. H. Costa, and J. L. Berral, "You only run once: spark auto-tuning from a single run," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2039–2051, 2020.

[5] N. Nguyen, M. M. H. Khan, and K. Wang, "Towards automatic tuning of apache spark configuration," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 417–425.

[6] D. Nikitopoulou, "Ml-driven automated framework for tuning spark applications," 2020.

[7] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 71–91, 2020.

[8] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The journal of machine learning research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[9] Y. Park, B. Tak, and W.-S. Han, "Qaad (query-as-a-data): Scalable execution of massive number of small queries in spark," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–26, 2023.

[10] C. Dünner, T. Parnell, K. Atasu, M. Sifalakis, and H. Pozidis, "Understanding and optimizing the performance of distributed machine learning applications on apache spark," in *2017 IEEE international conference on big data (big data)*. IEEE, 2017, pp. 331–338.

[11] Y. Guo, Z. Zhang, J. Jiang, W. Wu, C. Zhang, B. Cui, and J. Li, "Model averaging in distributed machine learning: a case study with apache spark," *The VLDB Journal*, vol. 30, pp. 693–712, 2021.

[12] K. Li, L. Deng, Y. Lu, and J. Wu, "Improve spark-based application performance using bayesian hyperparameter optimization," in *2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS)*. IEEE, 2019, pp. 703–707.

[13] R. Tooley, "Auto-tuning spark with bayesian optimisation," 2021.

[14] M. A. Rahman, J. Hossen, and C. Venkataseshaiah, "Smbsp: a self-tuning approach using machine learning to improve performance of spark in big data processing," in *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*. IEEE, 2018, pp. 274–279.

[15] T. Marlaithong, V. C. Barroso, and P. Phunchongharn, "A hyperparameter tuning approach for an online log parser," in *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE, 2021, pp. 1036–1040.

[16] M. Migliorini, R. Castellotti, L. Canali, and M. Zanetti, "Machine learning pipelines with modern big data tools for high energy physics," *Computing and Software for Big Science*, vol. 4, pp. 1–12, 2020.

[17] L. Xia, W. Sun, X. Liu, G. Sun, and X. Jiang, "Blaze: A high performance big data computing system for high energy physics," in *Journal of Physics: Conference Series*, vol. 2438, no. 1. IOP Publishing, 2023, p. 012012.

[18] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[19] Y. Tian, L. Si, X. Zhang, R. Cheng, C. He, K. C. Tan, and Y. Jin, "Evolutionary large-scale multi-objective optimization: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–34, 2021.

[20] S. Sehrish, J. Kowalkowski, and M. Paterno, "Spark and hpc for high energy physics data analyses," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 1048–1057.

[21] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: a survey on software technologies," *Cluster Computing*, vol. 26, no. 3, pp. 1845–1875, 2023.

[22] H. Li, H. Dai, Z. Liu, H. Fu, and Y. Zou, "Dynamic energy-efficient scheduling for streaming applications in storm," *Computing*, vol. 104, no. 2, pp. 413–432, 2022.

[23] V. E. Padulano, "Distributed computing solutions for high energy physics interactive data analysis," Ph.D. dissertation, Universitat Politècnica de València, 2023.

[24] S. Lee, K.-y. Hou, K. Wang, S. Sehrish, M. Paterno, J. Kowalkowski, Q. Koziol, R. B. Ross, A. Agrawal, A. Choudhary *et al.*, "A case study on parallel hdf5 dataset concatenation for high energy physics data analysis," *Parallel Computing*, vol. 110, p. 102877, 2022.

[25] K. Govind, G. Pierre, and R. Rouvoy, "Studying the energy consumption of stream processing engines in the cloud," in *IC2E 2023-11th IEEE International Conference on Cloud Engineering*. IEEE, 2023, pp. 1–9.

[26] S. Maroulis, N. Zacheilas, and V. Kalogeraki, "A framework for efficient energy scheduling of spark workloads," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 2614–2615.

[27] N. Mostafa, H. S. M. Ramadan, and O. Elfarouk, "Renewable energy management in smart grids by using big data analytics and machine learning," *Machine Learning with Applications*, vol. 9, p. 100363, 2022.

[28] W. Shi, H. Li, J. Guan, H. Zeng *et al.*, "Energy-efficient scheduling algorithms based on task clustering in heterogeneous spark clusters," *Parallel Computing*, vol. 112, p. 102947, 2022.

[29] H. Dou, X. Wei, K. Wang, Y. Zhang, P. Chen, and Y. Huang, "Eftuner: A bi-objective configuration parameter auto-tuning method towards energy-efficient big data processing," in *Proceedings of the 14th Asia-Pacific Symposium on Internetware*, 2023, pp. 292–301.

[30] "Intel power gadget," https://github.com/intel/pcm, accessed: 2024-01-30.

[31] "powerstat," https://manpages.ubuntu.com/manpages/xenial/man8/powerstat.8.html, accessed: 2024-01-30.

[32] "powertop," https://github.com/fenrus75/powertop, accessed: 2024-01-30.

[33] "perf," https://www.man7.org/linux/man-pages/man1/perf.1.html, accessed: 2024-01-30.