STARGAZER: A Deep Learning Approach for Estimating the Performance of Edge-Based Clustering Applications

Breno Dantas Cruz*, Arnab K. Paul†, Zheng Song*, Eli Tilevich*
{*Software Innovations Lab, †Distributed Systems and Storage Laboratory}, Virginia Tech
{bdantasc, akpaul, songz, tilevich}@cs.vt.edu

***Abstract*—As a solution to the sensor data deluge, edge computing processes sensor data by means of local devices. Many of these devices are resource-scarce in terms of the available processing capabilities and battery power. To achieve the required design trade-offs of edge applications, developers must be able to understand the performance and resource utilization of data processing algorithms. An increasing number of edge-based applications use machine learning (ML) as their key functionality. However, the performance and resource utilization of ML algorithms remain poorly understood, thus hindering the system design of edge-based ML applications. In addition, developers often cannot access real-world edge-based test beds during the design phase. To address this problem, we present an approach for estimating the performance of edge-based ML applications, with a particular application to clustering. To that end, we first comprehensively evaluate the performance and resource utilization of widely used clustering algorithms deployed in a representative edge environment. Second, we identify which properties of these algorithms are correlated with their performance and resource utilization. Finally, we apply our findings to create STARGAZER, a Deep Neural Network that given a clustering algorithm's computational load and input data size, estimates how this algorithm would perform and utilize resources in an edge-based application. Our tool provides viable decision-making support for addressing the multifaceted design challenges of edge-based ML applications.**

***Keywords*-Edge Applications, Machine Learning, Clustering Algorithms, Performance Estimation, Deep Neural Network**

I. INTRODUCTION

Machine learning (ML) has become an important building block for modern computing applications. As mobile, IoT, and wearable sensors collect ever-increasing volumes of data, ML models need to be continuously trained on that data. The resulting sensor data deluge renders cloud-based processing impractical, due to the privacy concerns and transfer constraints of wide-area networks. These realities give rise to edge-based ML training, as it eliminates network transmission bottlenecks and helps ensure data privacy[1]. Because edge resources are limited, edge-based ML applications must achieve high performance while utilizing resources efficiently. To that end, developers must understand how their source code translates into the performance and resource utilization of deployed edge-based ML applications [1], [2]. In this work, we focus on clustering as one of the most widely used ML technique. We put forward a new approach that estimates an edge-based clustering application's performance and resource utilization from its source code.

The performance and resource utilization of edge applications are hard to estimate. Therefore, developers cannot easily estimate how their edge-deployed code would perform under different workloads, a particularly important issue due to the resource scarcity of edge environments [3]. Specifically, to estimate the performance of edge-based clustering, developers need to understand not only the intrinsic properties of clustering algorithms, but also the expected resource utilization trade-offs. In addition, due to cost limitations, reduced repeatability, and lack of controlled environments, developers cannot always access real test beds [4].

The design of edge-based applications can substantially impact their performance and resource consumption [5]. The design process is driven by multiple considerations, with each design choice leading to the computational power, available memory, and energy scarcity trade-offs. For example, consider designing a mobile app that employs ML for facial recognition. The choice of an ML algorithm affects not only how much processing power and memory the app would utilize, but also how much energy it would consume. Failing to recognize such trade-offs can negatively impact the user experience.

To accurately estimate an edge application's performance, developers must understand how the underlying algorithm's characteristics would affect the resulting performance and resource utilization. The most performance-affecting characteristics of the edge-application source code are computational complexity and input data size, which must be linked to performance and resource consumption (e.g., CPU usage, memory utilization, energy consumption, and task execution time) [3], [6].

Use Case: Collecting and Processing Volcanic Data

Consider a developer designing an application that collects and clusters volcanic data to predict eruptions [7]. Multiple energy harvesting devices are located at the base of a volcano to collect their region's sensor data (e.g.,

[1]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5271

temperature, pressure, humidity, vibration, etc.). Each device is located at a hard-to-access location, with limited network access and energy resources. Due to resource constraints, the software and hardware stacks are optimized to minimize memory utilization, CPU usage, and energy consumption. In addition, for budgetary reasons, it would be infeasible to set up a test bed for the application under design. Given the source code of a clustering algorithm and the estimated size of the data to be collected at the edge nodes, the developer needs to be able to estimate the energy consumption, task duration, CPU usage, memory utilization, transactions per second, and block reads per second of the application once it is deployed in the target environment.

Contributions

This paper discusses our approach for estimating the performance of edge-based clustering applications. After presenting the background and related work (Section II), we describe an empirical study with an edge-based clustering application, whose four application variants differed in their algorithms (Section III). We used a state-of-the-art approach for ascertaining each clustering algorithm's asymptotic complexity [8]; we identified the correlations between the collected metrics via multiple linear and polynomial regression (Section IV). We used the collected metrics to train a Deep Neural Network (DNN) to be able to estimate the performance and resource utilization of an ML application under design (Section V). Finally, we discuss our conclusions and future work directions (Section VI).

The contributions of this paper are as follows:

1) We put forward a methodology for estimating the performance of edge-based ML applications.
   - We empirically evaluate four popular clustering algorithms deployed at the edge, in terms of their performance and resource utilization.
   - We identify how the observed performance and resource utilization correlate with each other, computational complexity, and data set size.
2) We introduce STARGAZER, a tool that reifies the methodology above.

II. Background & Related Work

We describe different approaches for estimating the performance of edge-based systems and computational complexity as well as explain regression models and deep learning in the context of our work.

A. Deep Learning and Performance Estimation

Deep Learning [9], [10] (DL) is an ML methodology that uses artificial neural networks (ANN), inspired by the structure of neurons in the human brain. DNNs have become indispensable tools in fields including image and video recognition, natural language processing, etc. DNNs combine the advantages of DL and neural networks and can solve nonlinear problems more satisfactorily compared to conventional ML algorithms. Prior work has applied ML techniques to estimate the performance of edge devices [11] and applications [12], [13]. DL has been applied widely to solve problems in multiple domains [14]–[16]. DL methods can deliver performance and accuracy greater than those of ML ones [17], [18]. STARGAZER's prediction heuristic is driven by DL to estimate the performance of devices executing clustering algorithms in the edge.

The performance of edge applications has been estimated using simulators [4], [19] as well as static and dynamic program analyses [20]–[22]. Our approach differs by not relying on simulation, but instead estimating performance from the source code via a DNN. In addition, STARGAZER estimates not only energy consumption, but also additional performance metrics (e.g., % CPU usage, % memory utilization, etc.). To ascertain the complexity of ML tasks under evaluation, STARGAZER uses Singularity in a black-box fashion [8]. To find the correlations in the collected data sets to train STARGAZER, this work uses linear and polynomial regression models. A linear regression model applies linear predictor functions to estimate model output parameters from the data [23]. A polynomial model regression fits a nonlinear relationship between the data and corresponding output parameters via an n^{th} degree polynomial [24].

B. Clustering

Clustering refers to techniques that group a set of items with similar attributes [25]. Clustering has been the subject of numerous research efforts: a Google Scholar search for the word "Clustering" returns over 770,000 publications only since 2015. With its primary application to exploratory data mining, clustering is used in domains including pattern recognition, image analysis, and information retrieval [26].

Despite a wide variety of clustering algorithms, our exploration focuses on the ones most widely used and offering dissimilar performance characteristics. Specifically, we study *K-Means*, an unsupervised learning technique that assigns *n* elements to *k* clusters, thus allocating each element to the cluster with the nearest mean value [27]; *Farthest First Clustering*, a variant of K-Means, based on heuristics, that chooses centroids and assigns the elements in the cluster at the farthest point from the existing cluster center within the data area [28]; *Expectation-Maximization* (EM), a probability-based technique, that assigns elements to clusters based on a probability model rather than a distance metric, like in K-Means. EM partially assigns elements to different clusters [29]; *Density-Based Spatial Clustering of Application with Noise*, a density-based technique, that identifies clusters by analyzing the concentration (higher point density) of data points, often resulting in arbitrarily shaped clusters, as compared to K-Means [30]. This work is not concerned with distributing, scaling, or optimizing clustering algorithms. Our goal is to understand the performance

characteristics of existing clustering algorithms deployed in the edge.

III. Empirical Study

In this section, we present our empirical study's motivation, system overview, implementation, methodology, and results. The data collected in this Section is analyzed in Section IV and used to train StarGazer in Section V.

A. Study Overview, Objectives, and Methodology

We deploy and benchmark the KMeans, EM, FF, and DBSC clustering algorithms in different variants of an edge-based application. We have chosen KMeans, EM, FF, and DBSC as previous works in the literature compare the performance of these algorithms [31]–[34]. *Our study's objective is to determine how the evaluated variants differ in their respective performance & resource utilization at the edge.* In particular, we obtain the performance metrics of % CPU usage, % memory utilization, block read per second, transactions per second, energy consumption, and task duration. The privacy and security issues are considered as out of scope in this study.

Our ultimate goal is to be able to understand the impact of choosing a particular ML algorithm on the overall system performance. The measurements collected herein can inform developers about the expected behavior of future edge-based ML applications under different requirements (see Section V).

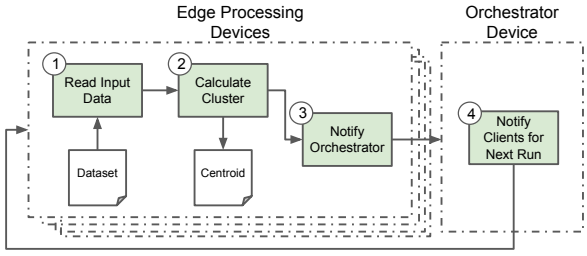

Figure 1. Experimental workflow on the edge.

The workflow diagram in Figure 1 shows our experimental edge-based clustering system that can be parameterized with any clustering algorithm. Its variants take random (x, y) coordinates as input. (**1**) Edge processing devices read their respective datasets. (**2**) Each edge processing device applies the clustering algorithm under evaluation to compute the coordinates of its unique centroid. (**3**) Each edge processing device notifies the orchestrator device that it has finished its execution. (**4**) Lastly, the orchestrator device requests that all edge processing devices perform the next run.

The study's methodology is to execute the four variants under the same load sets and collect the observed performance and resource utilization metrics. The system is composed of nine (eight edge processing devices and an orchestrator) Raspberry Pis 3 Model B, with Quad-Core 1.2 GHz and 1 GB RAM running Raspbian OS version 3.0.1. To measure energy consumption, we use a "watts up? Pro" [35]. This setup is representative of edge computing setups performed on commodity devices[2][3][4].

B. System Implementation

We implemented our system in Java 1.8, the Weka library version 3.7.17 [36], and the LipeRMI [37] library for device-to-device communication. Due to our empirical study's scope, we implemented four edge processing device variants. Each variant uses a different clustering algorithm in step 2 of Figure 1. For our evaluation, we selected the K-Means, Farthest First, EM, and Density-Based Scan (DBSC) Weka clustering algorithms due their popularity, as a Google Scholar search shows 3,180,000 results for K-Means, 40,300 results for Farthest First, 1,150,000 results for EM, and 157,000 results for Density-Based Scan. We profiled each variant with Singularity [8] and decided to use the reported **average asymptotic complexity**, as, according to Singularity, more than one of the clustering algorithms shared the same performance values (e.g., FF and EM). The reproducibility package (Section VII) includes all the source code, performance measurements, and generated data.

C. Results

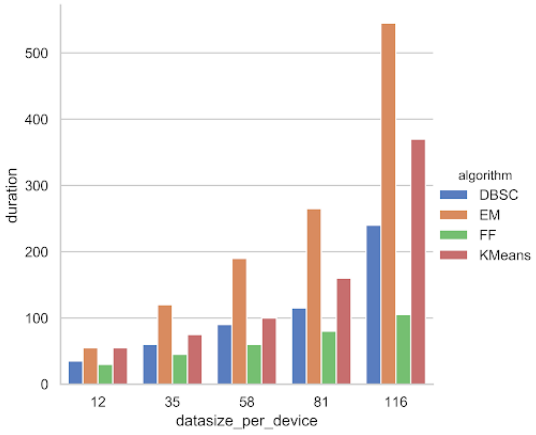

Figure 2. Average duration task per variant in seconds. X axis is the dataset size in MB.

Figure 3 and 4 show the average % CPU usage and % memory utilization of edge processing devices respectively. We observed the highest % CPU usage value, around 23%, with EM algorithm while operating on input data set size 81 MB per device. We observed the highest % memory

[2]https://datacenterfrontier.com/facets-of-the-edge-the-raspberry-pi-as-an-edge-ai-device/

[3]https://www.techrepublic.com/article/25-raspberry-pi-add-on-gets-you-started-with-edge-computing-ai/

[4]https://www.ibm.com/blogs/internet-of-things/edge-iot-analytics/

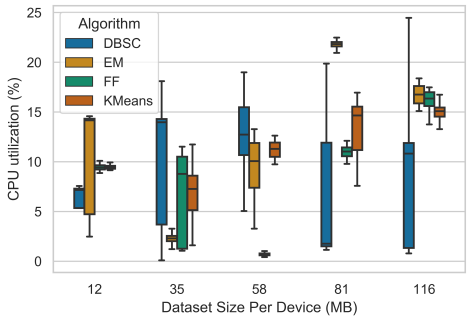

Figure 3. Average % CPU usage per variant. X axis is the dataset size in MB.

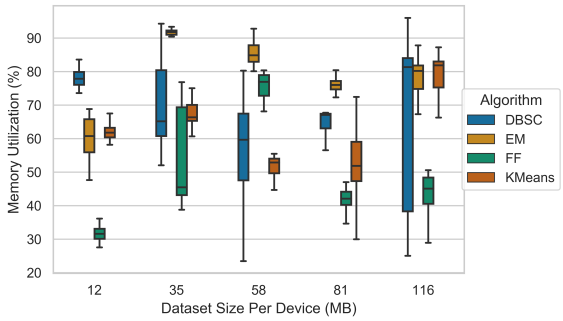

Figure 4. Average % memory utilization. X axis is the dataset size in MB.

utilization, around 93%, with `EM` algorithm while operating data set size 35 MB per device. In terms of the number of transactions per second, we observed that variant `FF` the values were was considerably lower than compared to the other variants. `FF` had at most 25 transactions per second (tps) for the dataset of size 116MB, while `EM` had the highest with at more than 250 tps for the dataset of size 116MB. For energy consumption, variant `EM` had consumed on average the most joules, while variant `FF` consumed the least. We observed that the variant `FF` had the shortest execution time and that `KMeans` had the longest (see Figure 2), additionally, as expected, we observed that a close relationship between duration and energy consumption (see Section IV for more details). Regarding block reads per second (breadps), we observed that variant DBSC had the highest values with more than 6,000 breadps while operating on data set 116 MB, while the other variants had values under 3,500 for the same data set. The reproducibility package includes all graphs generated in this study.

D. Measurements Highlights

The `EM` variant consumes more energy than the other variants on average because it makes heavier usage of the CPU and memory resources of the edge processing devices than the other variants. The `FF` variant consumes less energy and requires less time to complete its operations than the other variants, which is due to its lower asymptotic complexity.

E. Threats to Validity

Internal Threats: We implemented all the systems used in our experiments. It is possible that some other implementations would have yielded different performance characteristics. Nevertheless, our implementation strategies followed widely acceptable principles by using known open-source solutions for distributed interactions.

External Threats: In terms of time for completion, processes running in parallel may have influenced the results. To mitigate this threat, we terminated all background processes before running our experiments; then, we executed multiple runs, averaging the results, so as to effectively approximate the amount of energy consumed by each test. Weka and Raspbian OS[5] are updated continuously, so newer versions may be better optimized than the one used in our experiments. In terms of time, different hardware configurations may offer dissimilar transmission latencies and response times. In terms of the accuracy of our power measurement procedures, our readings are directly related to the accuracy of the power monitoring devices used in our experiments. Finally, our findings may not be directly applicable to other clustering algorithms not covered in our evaluations. Nevertheless, if the performance of the actual clustering algorithms improves, the observed measurements are likely to change as well.

IV. Data Analysis

A. Motivation and Research Questions

With the increasing popularity and scale of edge-based applications, developers have to take into account a growing number of complex factors and requirements in their designs. One of the hardest system design issues is estimating how an algorithmic choice would translate into the actual performance metrics in an implemented and deployed system. To make inroads in tackling this issue, we analyse the data collected in Section III for the presence of correlations between the performance metrics, code properties, and input data size, thus seeking answers to the following questions:

- **MQ1**: Are code complexity and input data size correlated with the collected metrics?
- **MQ2**: Which collected metrics are cross-correlated?

B. Methodology

To answer **MQ1**, we developed six multilinear regression models, one for each performance metric that use as input dataset size and algorithmic complexity. To answer **MQ2**, we apply polynomial regression models to identify whether the collected performance metrics are correlated with each other. Each model targets a different performance estimator

[5]https://www.raspberrypi.org/downloads/noobs/

from a given input data. We used the t-statistic data obtained from regression analysis to determine if the dependent variables were influential in describing the variation of the dependent variable. We used coefficient determination (R^2), F-test, and root mean squared error (RMSE) to determine the accuracy of the regression model. R^2 is a statistical measure of how close the data are to the regression line. F-test checks if the variances of two populations are the same, by comparing the ratio of the variances. An F-test of 1 indicates that two populations are the same. The RMSE represents the differences between estimated and observed values. Additionally, it measures the model's accuracy. We used the measurements gathered from the empirical study (Section III) for the regression model. Additionally, we used Singularity [8] to analyse the asymptotic complexity the clustering algorithms. In total we used 7201 data points from the empirical study, please refer to the Availability section III-B for the data. For the regression model, we divided the data in two sets, 80% (5761) of the data set was used for training model and the remaining 20% (1440) for testing. We adhere to a higher model accuracy than prior research on modeling performance in distributed applications [38]. A regression model is considered successful only if its accuracy exceeds 60%.

C. MQ1: On Complexity, Data Set Size, and Performance

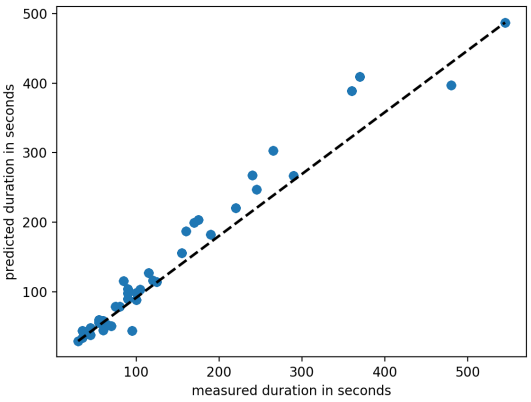

Figure 5. Regression line for duration.

We observed that **task duration**, and **energy consumption** are directly correlated to **data set size per device**, and **asymptotic complexity**. Figure 5 and Figure 6 show the regression lines for duration and energy consumption, respectively. We observed the following indicators for the multilinear regression models **task duration** R^2 0.95 MSE 708.50; **energy** R^2 0.97 & MSE 1857.16; We observed a small correlation between to the input data for **% CPU usage**, **%memory utilization**, **tps**, and **breadps**. We observed the following values for accuracy and mean square error: **%**

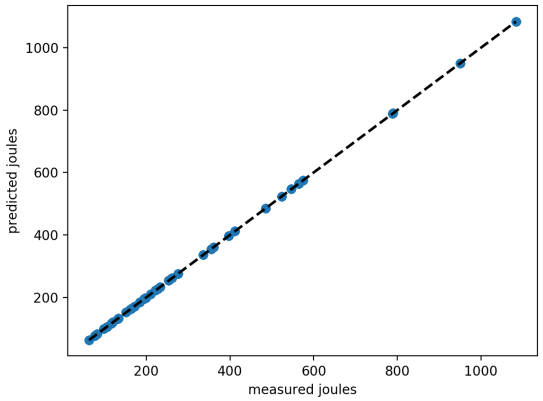

Figure 6. Regression line for energy consumption.

CPU usage – R^2 0.33 & MSE 36.24, **%memory utilization** – R^2 0.60 & MSE 135.57, **breadps** – R^2 0.29 & MSE 1.39 Kb; tps R^2 0.34 & MSE 2384.04;

D. MQ2: Correlations Between Performance Metrics

We observed that the following correlations for performance metrics: **% memory utilization** is correlated to **energy consumption**, **duration**, **asymptotic complexity**, and **data set size**, as a 6th degree polynomial regression model has accuracy of 0.68 (R^2) and mean square error of 111.03. We observed a correlation between **% CPU usage** and **asymptotic complexity**, **duration**, and **% memory utilization**, in 5th degree polynomial regression. We observed a accuracy of 0.84 R^2 and mean square error of 8.75.

Regarding transactions per second (**tps**), we observed the accuracy of 0.90 R^2 and mean square error of 350.69, when using a 4th degree polynomial regression model that receives as input **%memory utilization**, **asymptotic complexity**, **duration**, **%CPU usage**, and **block reads per second**. For block reads per second (**breadps**), we observed the accuracy accuracy of 0.88 R^2 and mean square error of 243.40 Bytes, when using a 4th degree polynomial regression model that receives as input **%memory utilization**, **asymptotic complexity**, **duration**, **%CPU usage**, and **tps**.

E. Summary of the Results

We developed six regression models by means of leveraging the t-statistic data obtained from regression analysis, it was determined if the dependent variables are influential in describing the variation of the dependent variable. The accuracy of the regression models was determined using coefficient determination (R^2), F-test, and root mean squared error (RMSE). The developed regression models for energy consumption, duration, % CPU usage, % memory utilization, transactions per second, and block reads per second are available in the reproducibility package (Section VII).

The following box summarizes the answers to our motivating questions:

MQ1:

- **duration** and **energy consumption** are correlated to asymptotic complexity and data set size.

MQ2:

- **% memory utilization** is correlated to data set size, duration, asymptotic complexity, and energy consumption.
- **% CPU usage** is correlated to % memory utilization, energy consumption, duration, complexity, and data set size.
- **tps** is correlated to % memory utilization, energy consumption, duration, asymptotic complexity, data set size, block reads per second, and % CPU usage.
- **breapds** is correlated to % memory utilization, energy consumption, duration, asymptotic complexity, data set size, tps, and % CPU usage.

V. STARGAZER

STARGAZER is a DNN that estimates the performance of four clustering ML algorithms. While traditional performance estimation approaches require that the code be deployed and executed, STARGAZER estimates the expected performance and resource utilization of edge-based ML applications from the code properties of the given ML clustering algorithm and data input size. Developers provide the input according to how their intended system will operate, and STARGAZER estimates total and per second performance of the system. That is, values for block reads per second (**breadps**), transactions per second (**tps**), percentage of memory utilization (**memory**), duration in seconds (**duration**), CPU percentage usage (**CPU**), and system energy consumption in joules.

A. Training

Table I
LAYERS OF THE NEURAL NETWORK

layer	Layer Size	Activation Function
input layer	12 neurons	Rectified Linear Unit
hidden layer #1	24 neurons	Scaled Exponential Linear Unit
hidden layer #2	36 neurons	Rectified Linear Unit
hidden layer #3	96 neurons	Exponential Linear Unit
hidden layer #4	96 neurons	Scaled Exponential Linear Unit
hidden layer #5	32 neurons	Rectified Linear Unit
hidden layer #6	24 neurons	Scaled Exponential Linear Unit
hidden layer #7	1 neurons	Linear

We trained the Keras Sequential model [39] by minimizing the Mean-Squared-Error between the predicted and ground truth on CPUs. The stochastic gradient descent variant Adam was applied as the optimizer [40]. We use 11,955,031 data vectors, totaling 5.4GB of data to train the DNN for the estimation of a per-second performance and 7,201 data vectors for estimation of average system performance. Both training sets were collected by our empirical study (Section III). We used the training/validation split of 80%/20%. When updating global models, we selected the following hyper-parameters by optimizing them throughout the empirical analysis: 128 for batch size and 200 for epochs. We supplemented these vectors with the complexity of clustering algorithms, obtained by running Singularity on the original versions of these algorithms. Table I shows the architecture of the DNN. We optimized the activation methods through trial and error, systematically experimenting with different configurations, activation methods, and numbers of neurons per layer to maximize the prediction accuracy for all metrics.

We used the insights we gained from Section IV to train the DNN. To predict total task duration, we use as input data set size and algorithmic complexity. For total **energy consumption**, we use as input data set size, algorithmic complexity, and duration. For **% memory utilization** prediction, we use as input data set size, complexity, and duration. For **% CPU usage** prediction, we use data set size, complexity, energy, duration, and memory. For **tps** we use as input data set size, complexity, memory, duration, energy, and breadps. For **breadps**, we use as input data set size, complexity, memory, duration, energy, and tps.

B. Results

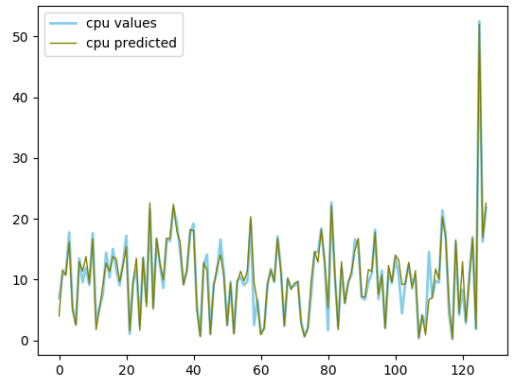

Figure 7. Estimated and observed system's average %CPU usage. Y axis is the % CPU usage, X is the index of the test in the validation dataset.

To estimate performance metrics in a per second arrangement, we use a training/validation split of 80% (9,564,025 data points) / 20% (2,391,006 data points), resulting in: **energy** accuracy values are 0.9999 (R^2), mean square error of 0.0340, and mean absolute error 0.28. For **total duration**

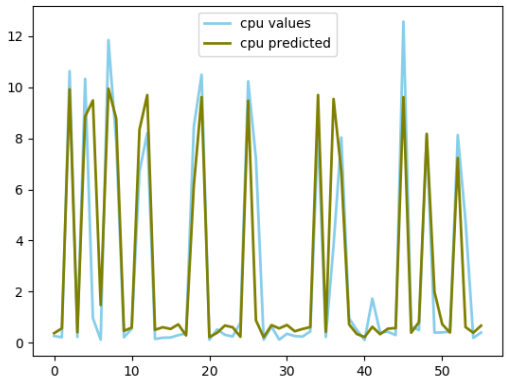

Figure 8. Estimated and observed system %CPU usage at a given second. Y axis is the % CPU usage, X is the index of the test in the validation dataset.

task accuracy values are 0.98 (R^2), mean square error of 214.92, and mean absolute error 0.47. For **memory** accuracy values are 0.67 (R^2), mean square error of 106.87, and mean absolute error 0.00. For **CPU** accuracy values are 0.89 (R^2), mean square error of 5.33, and mean absolute error 0.01. For **tps** accuracy values are 0.79 (R^2), mean square error of 326.50, and mean absolute error 0.00. For **breadps** accuracy values are 0.83 (R^2), mean square error of 197.85 Bytes, and mean absolute error 4.18. Figure 8 shows predicted and observed values estimating the average % CPU usage of the system at a given second.

To estimate average performance metrics, we also use a training/validation split of 80% (5,761 data points) / 20% (1,440 data points), resulting in: for **energy** estimation the accuracy values are 0.99 (R^2), mean square error of 7.98, and mean absolute error 0.14. For **total task duration** estimation the accuracy values are 0.98 (R^2), mean square error of 221.47, and mean absolute error 0.24. For **% memory utilization** estimation the accuracy values are 0.79 (R^2), mean square error of 68.34, and mean absolute error 0.00. For **% CPU usage** estimation the accuracy values are 0.98 (R^2), mean square error of 0.99, and mean absolute error 0.01. For **tps** estimation the accuracy values are 0.88 (R^2), mean square error of 410.50, and mean absolute error 0.00. For **breadps** estimation the accuracy values are 0.98 (R^2), mean square error of 49.23 bytes, and mean absolute error 0.0. Figure 7 shows the estimated and the observed average % CPU usage of the system. Refer to the reproducibility package (Section VII) for the DNN source code and datasets.

C. Results Discussion

From operating STARGAZER, we derived the following insights. First, the STARGAZER is more accurate at estimating the application's average performance than at estimating the same metrics at a given timestamp. Second, as we identified in Section IV, it is impossible to accurately estimate some metrics in isolation; interconnected metrics should be evaluated in concert. For example, % CPU usage is dependent on % memory utilization.

D. Use Case Revisited

Recall the volcanic data use case in Section I. Energy harvesting devices collect and locally process volcanic data. The hardware capabilities of these devices (e.g., low-power, energy harvesting, and heat dissipation) are known to application developers. In addition, based on the frequency of sensor readings, one can approximate the size of the volcanic data to be collected. However, for budgetary reasons, it would be infeasible to set up a test bed for the application under design. Assume that due to environmental temperatures, developers need to optimize the CPU usage to be under 15%, as heat dissipation is not optimal. Finally, developers know which clustering algorithm is best suited for the task. How can developers verify that once the application is deployed, its % CPU usage would be under 15%?

By using STARGAZER, developers can not only estimate the designed application's performance and resource utilization but also identify early on whether or not it would meet the requirements. For example, developers would parameterize STARGAZER with data as follows for an average system energy consumption and % CPU usage:

> `Input` – **Data set size per device** → 12MB; **ML algorithm complexity**→ Singularity's Complexity Report for KMeans; **duration** → 0.27 seconds; `Output` – **average energy consumption** → 0.04 joules

> `Input` – **Data set size per device** → 12MB; **ML algorithm complexity**→ Singularity's Complexity Report for KMeans; **% memory utilization**→ 50%; **duration** → 0.27 seconds; **average energy consumption** → 0.04 joules; `Output` – **% CPU usage** → 11%;

In the example above, developers learn that one device running the application would, on average, consume 0.04 joules, utilize 11% of its CPU capacity, and take 0.27 seconds to process 12MB of volcanic data. Assume that having obtained these values, developers become concerned with the possibility of the application taking too long to execute the task, as it would imply additional energy costs and CPU usage, hence more heat dissipation.

Developers would use STARGAZER to estimate the performance at a timestamp, different than 0.27 seconds (e.g., 3 seconds). To that end, they parameterize STARGAZER with data as follows to estimate the % CPU usage at the 3 seconds

timestamp to verify if the % CPU usage will be under the 15% requirement:

> `Input` – **Data set size per device** → 12MB; **ML algorithm complexity**→ Singularity's Complexity Report for KMeans; **Estimate % memory utilization**→ 50%; **time stamp** → 3 seconds; **average energy consumption** → 0.04 joules; `Output` – **% CPU usage** → 7%;

As reported by STARGAZER, developers would become aware that if the execution lasts for more than 3 seconds, the CPU usage is likely to reduce to 7%. This estimate shows that the designed application would meet the design constraints, thus avoiding costly deployments on a test bed.

Improving Guidelines Utility

To come up with a widely applicable and comprehensive estimation model for understanding the performance of different clustering algorithms at the edge, one would have to study numerous combinations of setups and clustering algorithms. Our study is only the first step in this process. Our hope that our findings can bootstrap many additional studies in this important area. By collating the results of such studies, one can define actionable information that would become a valuable asset in the decision-making process of system designers.

VI. Conclusions and Future Work

In this work, we put forward STARGAZER, a DNN that estimates the performance of edge-based clustering applications. To arrive at the prediction model, we performed an empirical study, in which we benchmarked four variants of an edge-based clustering application. Additionally, we used regression techniques to identify correlations between the collected metrics. We have identified several possible future work directions. First, we would like to extend our prediction models to support settings other than edge-based architectures. Second, we would like to identify if our findings are also applicable to cloud architectures. Finally, we would like to deploy STARGAZER to measure its usefulness in real-world software development settings.

VII. Availability

The project's source code, datasets, and collected metrics are available in the following online repository: https://github.com/brenodan/stargazer.

Acknowledgements

This research is supported by the National Science Foundation through the Grant CCF-1717065.

References

[1] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 20–26, IEEE, 2016.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] J. M. Rodriguez, C. Mateos, and A. Zunino, "Energy-efficient job stealing for cpu-intensive processing in mobile devices," *Computing*, vol. 96, no. 2, pp. 87–117, 2014.

[4] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] A. Toor, S. ul Islam, G. Ahmed, S. Jabbar, S. Khalid, and A. M. Sharif, "Energy efficient edge-of-things," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 82, 2019.

[7] J. Parra, O. Fuentes, E. Anthony, and V. Kreinovich, "Use of machine learning to analyze and–hopefully–predict volcano activity," *Acta Polytechnica Hungarica*, vol. 14, no. 3, 2017.

[8] J. Wei, J. Chen, Y. Feng, K. Ferles, and I. Dillig, "Singularity: Pattern fuzzing for worst case complexity," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 213–223, ACM, 2018.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] G. Liu, Y. Xu, Z. He, Y. Rao, J. Xia, and L. Fan, "Deep learning-based channel prediction for edge computing networks toward intelligent connected vehicles," *IEEE Access*, vol. 7, pp. 114487–114495, 2019.

[12] N. Jain, A. Bhatele, M. P. Robson, T. Gamblin, and L. V. Kale, "Predicting application performance using supervised learning on communication features," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 95, ACM, 2013.

[13] A. Grebhahn, N. Siegmund, H. Köstler, and S. Apel, "Performance prediction of multigrid-solver configurations," in *Software for Exascale Computing-SPPEXA 2013-2015*, pp. 69–88, Springer, 2016.

[14] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.

[15] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *2013 IEEE International Conf. on Acoustics, Speech and Signal Processing*.

[16] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 2014.

[17] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes, "Hdltex: Hierarchical deep learning for text classification," in *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pp. 364–371, IEEE, 2017.

[18] M. Z. Nezhad, D. Zhu, N. Sadati, and K. Yang, "A predictive approach using deep feature learning for electronic medical records: A comparative study," *arXiv preprint arXiv:1801.02961*, 2018.

[19] D. N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R. K. Naha, S. K. Battula, S. Garg, D. Puthal, P. James, A. Y. Zomaya, *et al.*, "Iotsim-edge: A simulation framework for modeling the behaviour of iot and edge computing environments," *arXiv preprint arXiv:1910.03026*, 2019.

[20] Y.-k. Choi, P. Zhang, P. Li, and J. Cong, "Hlscope+: Fast and accurate performance estimation for fpga hls," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 691–698, IEEE, 2017.

[21] R. W. Ahmad, A. Naveed, J. J. Rodrigues, A. Gani, S. A. Madani, J. Shuja, T. Maqsood, and S. Saeed, "Enhancement and assessment of a code-analysis-based energy estimation framework," *IEEE Systems Journal*, vol. 13, no. 1, pp. 1052–1059, 2018.

[22] R. W. Ahmad, S. H. Ab Hamid, A. Gani, M. S. Obaidat, J. Shuja, F. Rehman, and A. U. R. Khan, "Performance assessment of dynamic analysis based energy estimation tools," in *2018 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp. 1–12, IEEE, 2018.

[23] F. Khademi and K. Behfarnia, "Evaluation of concrete compressive strength using artificial neural network and multiple linear regression models," *Iran University of Science & Technology*, vol. 6, no. 3, pp. 423–432, 2016.

[24] J. R. Edwards and M. E. Parry, "On the use of polynomial regression equations as an alternative to difference scores in organizational research," *Academy of Management journal*, vol. 36, no. 6, pp. 1577–1613, 1993.

[25] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[26] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[27] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[28] M. Bilenko, S. Basu, and R. J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering," in *Proceedings of the twenty-first international conference on Machine learning*, p. 11, ACM, 2004.

[29] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[30] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[31] A. Karahoca and A. Kara, "Comparing clustering techniques for telecom churn management," in *Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics*, pp. 27–29, 2006.

[32] S. Revathi and D. T. Nalini, "Performance comparison of various clustering algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 2, 2013.

[33] G. Sehgal and D. K. Garg, "Comparison of various clustering algorithms," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 3074–3076, 2014.

[34] I. Rozhnov, V. Orlov, and L. Kazakovtsev, "Ensembles of clustering algorithms for problem of detection of homogeneous production batches of semiconductor devices," 2018.

[35] J. M. Hirst, J. R. Miller, B. A. Kaplan, and D. D. Reed, "Watts up? pro ac power meter for automated energy recording," 2013.

[36] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[37] F. S. Andrade, "LipeRMI a light weight internet approach for remote method invocation," 2006.

[38] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pp. 73–80, IEEE, 2012.

[39] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.