Machine Learning-Based Real-time Task Scheduling for Apache Storm

Cheng-Ying Wu[b], Qi Zhao[a], Cheng-Yu Cheng[b], Yuchen Yang[a], Muhammad A. Qureshi[c], Hang Liu[b], and Genshe Chen[a]

[a]Intelligent Fusion Technology, Inc, 20410 Century Blvd, Suite 230, Germantown, USA
[b]The Catholic University of America, 620 Michigan Ave NE, Washington D.C., USA
[c]US Army C5ISR Center, 6662 Gunner Circle, Aberdeen Proving Ground, USA

ABSTRACT

Apache Storm is a popular open-source distributed computing platform for real-time big-data processing. However, the existing task scheduling algorithms for Apache Storm do not adequately take into account the heterogeneity and dynamics of node computing resources and task demands, leading to high processing latency and suboptimal performance. In this thesis, we propose an innovative machine learning-based task scheduling scheme tailored for Apache Storm. The scheme leverages machine learning models to predict task performance and assigns a task to the computation node with the lowest predicted processing latency. In our design, each node operates a machine learning-based monitoring mechanism. When the master node schedules a new task, it queries the computation nodes obtains their available resources, and processes latency predictions to make the optimal assignment decision. We explored three machine learning models, including Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN), and Deep Belief Networks (DBN). Our experiments showed that LSTM achieved the most accurate latency predictions. The evaluation results demonstrate that Apache Storm with the proposed LSTM-based scheduling scheme significantly improves the task processing delay and resource utilization, compared to the existing algorithms.

Keywords: Machine Learning, Task Scheduling, Apache Storm, Long Short-Term Memory, Convolutional Neural Networks, Deep Belief Networks

1. INTRODUCTION

The recent years have seen a remarkable rise in the adoption of distributed systems,[1] primarily due to their ability to enable efficient and flexible collaboration among computers, Internet of Things (IoT) devices, and mobile devices. As the number of devices in these distributed systems continues to surge,[2] the volume of data generated has reached unprecedented heights. Big data analysis, for instance, frequently involves processing millions of data points or continuous data streams. Such applications demand rapid processing within constrained resource environments. Distributed systems offer vast computing capabilities to meet the soaring demand for computing resources, rendering them invaluable for modern data-driven applications.

To address the challenges posed by these large-scale data processing tasks, we employ the open-source Apache Storm framework[3] as our platform. Apache Storm is an open-source framework designed for distributed systems, offering real-time data streaming processing and reliable handling of unbounded data streams. With Apache Storm, data processing occurs in real-time as it arrives, ensuring timely and efficient processing of streaming data. Additionally, Apache Storm remains on standby when no data is being processed, optimizing resource utilization and ensuring readiness to handle incoming data streams.

By leveraging Apache Storm, our platform can effectively process large volumes of data streams in real-time, enabling timely and accurate analysis for various applications. This capability is particularly crucial for scenarios where fast decision-making is essential, such as real-time monitoring, anomaly detection, and predictive analytics.

Further author information: (Send correspondence to Qi Zhao or Genshe Chen)
Qi Zhao: E-mail: qi.zhao@intfusiontech.com
Genshe Chen: E-mail: gchen@intfusiontech.com

Through our utilization of Apache Storm, we aim to harness the power of distributed computing to address the growing demand for efficient data processing in modern distributed systems.

The heterogeneity of devices in distributed systems[4] poses significant challenges for effective system design, particularly in task scheduling and resource allocation areas. Devices exhibit diverse operational characteristics and capabilities, leading to varying resource requirements across applications, with some prioritizing memory usage while others demanding more CPU resources. Apache Storm, a prominent platform employed in such distributed environments, offers scheduling options like the default round-robin scheduler and a resource-aware scheduler. However, these schedulers primarily operate under the assumption of a homogeneous computing environment, diminishing their effectiveness in heterogeneous systems where the disparity in device capabilities can result in sub-optimal resource utilization and inefficient task distribution.

The mismatch between the homogeneous assumptions of existing schedulers and the heterogeneous nature of modern distributed systems necessitates the development of more sophisticated scheduling algorithms. These algorithms must intelligently consider the unique characteristics and resource profiles of each device in the system, ensuring that tasks are allocated in a manner that maximizes overall system performance and efficiency. Addressing this challenge is crucial for leveraging the full potential of heterogeneous distributed systems and enabling efficient resource utilization across diverse device capabilities.

In this paper, we introduce a novel machine learning-based task scheduling scheme tailored for Apache Storm, designed to address the limitations of traditional schedulers that fail to account for the heterogeneity of resource availability and task demands. Our approach leverages advanced machine learning algorithms to predict the latency performance, defined as the total processing time of a task, based on the current resource availability of each worker node. This innovative scheduler continuously monitors the resource status within the Apache Storm cluster and strategically assigns tasks to the nodes that are best suited to handle them, thereby optimizing latency performance. Crucially, our scheme takes into account the influence of various resources, such as CPU and memory, on the computational delay of tasks, to ensure that tasks are allocated to nodes with sufficient resources to minimize processing times.

A notable advantage of our proposed method is its ability to operate effectively without requiring prior knowledge of the available resources or the specific demands of each task. This dynamic adaptation is particularly valuable in real-time applications, where data processing and resource needs can fluctuate rapidly. By continuously monitoring and adjusting to changes in the system, our scheduler can respond promptly to varying workloads and resource availability, ensuring optimal task allocation at all times.

To evaluate the efficacy of our proposed scheduler, we implemented a real-time object detection[5] application on the testbed comprising heterogeneous worker nodes. Comparative results showed that our machine learning-based scheduler significantly outperformed the default schedulers provided by Apache Storm, achieving substantial reductions in latency of the scheduled tasks. This empirical evidence validates our hypothesis that tasks scheduled based on predictive analytics can lead to markedly improved performance, particularly in heterogeneous distributed environments.

Furthermore, our approach to task scheduling is not limited to Apache Storm but can be extended to other distributed data processing frameworks. By integrating our machine learning-based scheduler into these frameworks, we can unlock the potential for optimized resource utilization and improved performance across a wide range of applications and domains. Moreover, our proposed scheduling scheme exhibits remarkable flexibility, allowing for seamless enhancements by incorporating additional features into its predictive model. For example, it can be extended to consider network link bandwidth, device power constraints, and other contextual factors that may influence task performance. This extensibility enables our scheduler to become increasingly robust and adaptable, tailored to diverse computing environments, and ensuring its continued effectiveness across varying operational conditions. By integrating these supplementary parameters, our scheduler can deliver even more precise task allocation decisions, optimizing not only for speed and efficiency but also for broader operational dynamics inherent to modern distributed systems. The adaptability of our approach empowers it to evolve and accommodate emerging requirements, offering a future-proof solution for efficient resource allocation in dynamic and heterogeneous computing landscapes.

Our proposed machine learning-based task scheduling scheme addresses the critical challenge of efficient resource allocation in heterogeneous distributed systems. By leveraging predictive analytics and dynamic adaptation, our scheduler ensures that tasks are assigned to the most suitable nodes, minimizing latency and maximizing overall system performance. The promising results obtained from our evaluation experiments demonstrate the significant benefits of this approach and pave the way for further research and development in this area.

The rest of the paper is organized as follows. Section 2 introduces the existing related work on this topic. Then, Section 3 briefly describes the background knowledge of the open-source Apache Storm framework. Section 4 presents the details of our task scheduler design and Section 5 presents the details of the testbed implementation and evaluation experiments. Lastly, Section 6 draws the conclusion of our work.

2. RELATED WORK

In the realm of distributed systems, efficient task scheduling is a critical challenge that demands effective partitioning of resources among tasks, particularly in heterogeneous environments. Task scheduling in heterogeneous distributed systems has been extensively studied, with researchers proposing various solutions[6][7] to address the unique challenges posed by diverse computing environments. One notable work, "Hetero-Edge: Orchestration of Real-time Vision Applications on Heterogeneous Edge Clouds",[8] introduces a latency-aware edge resource orchestration platform built upon Apache Storm. The authors first estimate the performance and resource requirements of each task through a process called latency estimation. Subsequently, a scheduler leverages these latency estimations to assign tasks to the nodes that offer the best performance and resource match. Their experiments demonstrate that this latency-aware task scheduling algorithm outperforms both the default scheduler and the resource-aware scheduler in Apache Storm by a margin of 25%. The default scheduler in Apache Storm follows a round-robin approach,[9] assigning tasks equally to machines within the cluster. This scheduler operates under the assumption that all machines possess identical resources, making an equal distribution of tasks an efficient strategy. However, as highlighted by Boyang,[10] the default scheduler fails to consider the resource availability in the underlying cluster or the resource requirements of Storm Topology during task scheduling. To address this limitation, Boyang proposed a Resource-Aware scheduler,[10] which allows users to specify memory usage, heap size, and CPU usage requirements for their Topology. This scheduler acknowledges the resource heterogeneity present in the cluster and assigns tasks to machines that match the specified resource requirements. By considering the diversity of available resources, the Resource-Aware scheduler aims to improve resource utilization and overall system performance. Another group proposed a better scheduler,[11] which first optimizes the Topology structure and then schedules executors on worker nodes by taking into account the communication traffic between components. Zhang and Jing's scheduler[12] operates in two steps, first assigning executors to slots based on traffic patterns, and then allocating slots to worker nodes considering load balancing. Aniello et al.'s adaptive online scheduler[13] continuously monitors system performance and dynamically reschedules task deployment at runtime to improve overall efficiency. The T-Storm scheduler[14] by Xu and Jielong is traffic-aware, using runtime data to allocate tasks in a way that reduces both inter-node and inter-process traffic while ensuring worker node load does not exceed capacity. The P-Scheduler[15] by Eskandari et al. employs a hierarchical graph partitioning approach to schedule tasks, first determining the required number of nodes based on estimated load, and then assigning heavy traffic task pairs to the same worker node to minimize inter-node and inter-process communication. These schedulers offer various strategies to address the limitations of the default scheduler, generally aiming to improve resource utilization, reduce communication overhead, balance load across the cluster, and ultimately enhance the overall performance and efficiency of distributed stream processing in Apache Storm.

While these schedulers offer various strategies to improve upon the limitations of the default Round-Robin scheduler in Apache Storm, they still fall short of providing a truly dynamic and accurate approach for choosing the optimal set of nodes to handle tasks. Existing schedulers, though considering factors like communication patterns, resource availability, and load balancing, may not fully capture the complex interplay of variables that influence task performance in a heterogeneous distributed environment. To address this gap, there is a pressing need for a more intelligent and adaptive scheduling mechanism that can dynamically assess the state of the system, resource profiles of individual nodes, and the characteristics of incoming tasks, and then make informed decisions on task placement. Such a scheduler should be able to continuously learn and evolve, leveraging techniques like machine learning to accurately model the performance implications of different scheduling decisions. It is

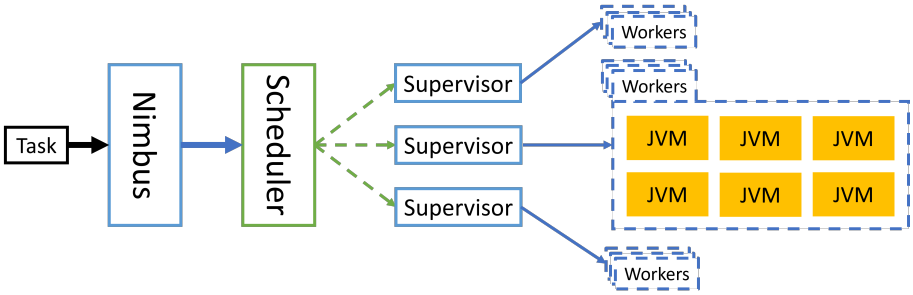

Figure 1. The structure of Apache Storm framework.

this gap that motivated the design of our proposed machine learning-based scheduler for Apache Storm. By harnessing the power of predictive analytics and dynamic adaptation, our scheduler aims to intelligently match tasks with the most suitable nodes, ensuring optimal resource utilization, minimized latency, and overall system efficiency. Apache Storm provides users with the flexibility to implement custom schedulers, enabling them to develop tailored scheduling strategies that align with their specific application needs and operational constraints.

3. BACKGROUND

Apache Storm is a popular distributed computing framework designed for real-time processing of data streams. The architecture of Apache Storm is composed of two main types of nodes: the master node, known as Nimbus, and the worker nodes, referred to as Supervisors. Nimbus is responsible for distributing tasks across the cluster, monitoring the health of the system, and managing fault tolerance. Each Supervisor node manages one or more worker processes that execute the actual computation. These workers are capable of running multiple tasks in parallel, each in its own Java Virtual Machine (JVM),[16] allowing for efficient handling of diverse operations concurrently. The tasks assigned to workers are based on a Topology defined by the user, which dictates the data flow and processing logic across the cluster. Workers within a Supervisor share resources such as CPU and memory, and they execute tasks either in parallel or cooperatively, depending on the configuration of the Topology. Figure 1 illustrates the typical structure of an Apache Storm deployment: Nimbus receives a task and assigns it to an appropriate Supervisor via its scheduling mechanism, ensuring optimal distribution of work across the available resources.

The user configures the behavior of Apache Storm through the task Topology, which consists of two main components: Spout and Bolt. Spouts typically serve as sources of data streams, while Bolts handle intermediate data processing tasks. The Topology defines the functionality and execution method for each Spout and Bolt, specifying whether they operate in parallel or cooperatively, and how data streams flow between them. Figure 2 shows two types of Topology: serial and parallel. In a serial Topology, Spouts and Bolts perform different functions and operate sequentially, with each Spout/Bolt waiting for the previous one to finish processing data before it begins. In contrast, Bolts in a parallel Topology can execute the same function concurrently, enabling faster computation by leveraging parallel processing.

The default task scheduler[9] in Apache Storm operates in a round-robin fashion, assigning tasks equally to workers within the cluster without considering the diversity of available resources on each worker node. While this approach ensures an even distribution of tasks, it fails to account for the heterogeneity of resource availability and task demands. Consequently, a machine could be assigned multiple tasks, potentially leading to resource overload and increased latency if the scheduling process disregards the current resource utilization levels. To address this limitation, we propose a novel task scheduling scheme that leverages Machine Learning techniques to predict application latency based on resource availability. By strategically assigning tasks to the nodes with the lowest predicted latency, our approach aims to optimize resource utilization and enhance overall application performance. Through intelligent task allocation, our scheduler mitigates the risk of resource overload and ensures that tasks are executed on nodes with sufficient computational capacity, thereby minimizing processing delays and improving system efficiency. In the subsequent sections, we will provide a detailed description of our system model and the underlying algorithm employed in our Machine Learning-based task scheduling approach.

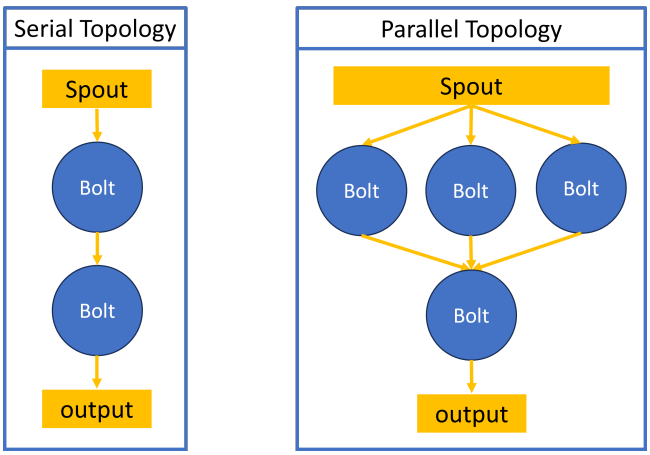

Figure 2. Serial Topology and Parallel Topology.

This comprehensive explanation will elucidate the fundamental principles and methodologies underpinning our proposed solution, enabling a thorough understanding of its operational mechanics and potential benefits.

4. TASK SCHEDULER DESIGN

Apache Storm is a real-time distributed system. It keeps opening and waiting for the task to be submitted. Furthermore, Apache Storm can work on multiple tasks at the same time. The existing scheduler doesn't know the node's current resource availability. If a task occupies all the resources on a node, the node cannot work on another task, but the scheduler still assigns new tasks to the nodes. This problem also brings about the scheduler not knowing the task resource demand and causes the scheduler to assign the task to the node that doesn't satisfy the task resource demand.

There are three components for our machine learning-based scheduler: (1) the machine learning model training, (2) the API that communicates between Nimbus (the master node) and Supervisor (the leaf node), and (3) the machine learning-based scheduler algorithm.

4.1 Machine learning model training and data collection

To understand the task performance under different resource utilization. The features for the machine learning model to predict performance are CPU utilization and memory usage. These two features are representative of the resources on a computer. The machine learning model predicts the performance according to these features. First, we need to estimate the performance of the task. We implement an example real-time object detection application as the task. The latency can be defined as the average frame processing time in a minute. We executed the application and collected the latency under different CPU utilization and memory usage. Next, we used the Deep Neural Network (DNN)[17] as a machine learning model. We explored three types of DNN models: Long Short-Term Memory (LSTM),[18] Convolutional Neural Networks (CNN),[19] and Deep Belief Networks (DBN).[20] We compared three models' Means Square Error (MSE),[21] as shown in Figure 3. The LSTM model is the most accurate among these three DNN models.

4.2 The RESTful API

We used RESTful API[22] to be the communication between Nimbus and Supervisor. The RESTful interface could allow two systems to exchange their message through the internet. The RESTful API securely transmits the message through HTTP. We use the API when Nimbus needs to ask for the predicted latency from the Supervisor. Figure 4. shows the implementation of the API in our testbed. The number is the expected latency that the Supervisor's LSTM model predicted based on the current latency.

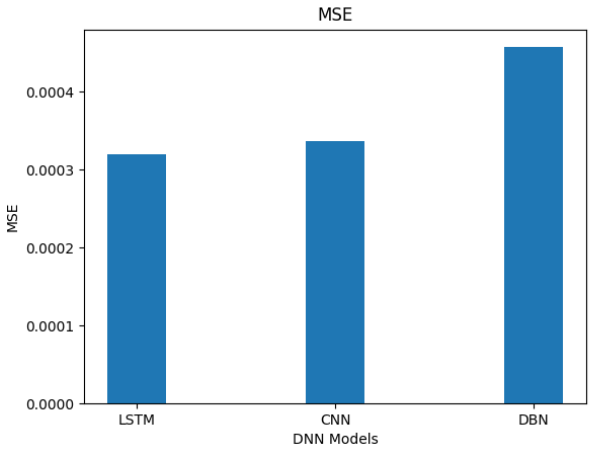

Figure 3. Three models' mean square error (MSE) comparison.

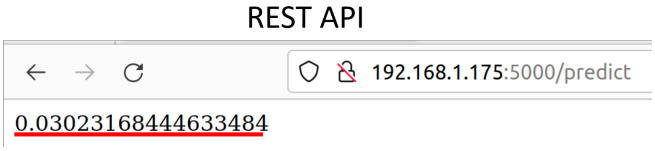

Figure 4. The RESTful API implementation.

4.3 Machine learning-based Scheduler System Model

In our scheme, the Apache Storm cluster is a heterogeneous distribution system. The nodes have different computation capacities and resources. Therefore, the nodes have their own machine-learning model and API. The machine learning model is trained on the node independently. The API relates to the machine-learning model. Nimbus sends requests to the Supervisor's API; the predicted latency will be returned to Nimbus via API. Apache Storm is a real-time distributed system. Nimbus will keep opening and waiting for the task. When Nimbus receives the task, it asks the Supervisor to predict the latency based on the current CPU utilization and memory usage through the RESTful API. The predicted latency by a node represents the performance of the task if the Nimbus schedules the task to the node. After Nimbus collects all available Supervisors' predicted latency in the cluster, the scheduler assigns the task to the node with the lowest predicted latency.

5. EVALUATION

5.1 Testbed Design and Implementation

We experimented to compare our machine learning-based scheduler and Apache Storm default scheduler. The default scheduler is in round-robin style and assumes all the nodes in the cluster have the same computation capacity, resources, and environment. This means that equally scheduling the task to nodes is the most efficient method. However, we propose to implement the algorithm on the heterogeneous distributed system. We integrated heterogeneous nodes with different computing power into the testbed in the experiment. We ran the testbed on Linux Ubuntu. There are three computers, all of which are Supervisors, and one is Nimbus.

Second, we executed different numbers of tasks simultaneously, ranging from 3 to 30. This helps us determine how the scheduler assigned the task to the node and the latency variety when all the resources are occupied in the cluster. However, our nodes behave great when executing only one application simultaneously. If multiple applications are executed, the node might run out of all its resources and cause the application process to slow or shut down. We implemented an example application, real-time video object detection by YOLOv5.[23] The video was inputted to the Supervisor, who then processed the frames individually. Therefore, for our scheduler, we used the LSTM model to learn this application latency under different CPU utilization and memory usage.

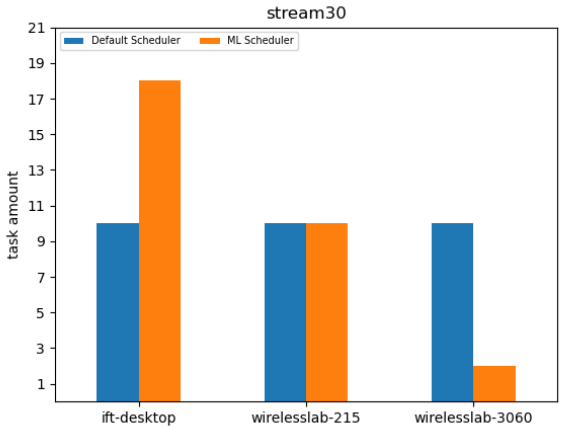

Figure 5. The task amount for each Supervisor.

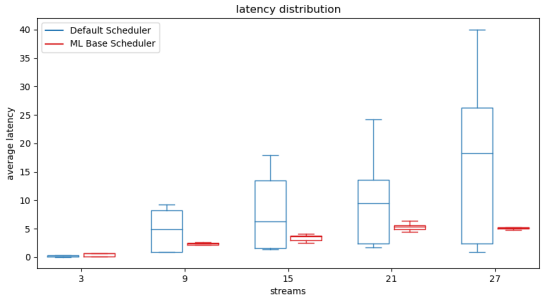

Figure 6. Quartiles express the latency distribution.

After generating the machine-learning model, we connected the model and the API that is used to communicate with Nimbus.

We've designed a serial Topology featuring a single Spout and a single Bolt. The Spout's role is crucial as it handles frame processing, specifically object detection. On the other hand, the Bolt is responsible for displaying the results. Our latency measurement is from the frame's input to the system until the object detection process is complete. In this Topology, the Spout and Bolt are assigned to separate nodes. Every Topology is independent. If we want to execute the application more than once in the cluster, we must submit multiple Topology to Nimbus. After Nimbus gets the Topology, the scheduler starts assigning the task to the node according to the algorithm. Figure 5 shows the task amount for each Supervisor when we submitted 30 Topology into the cluster. According to Figure 5, the blue bar results from the default scheduler, assigning the task to three Supervisors equally. The orange bar is our machine learning-based scheduler; it schedules the tasks for the Supervisor with the best performance.

5.2 Evaluation Experiments and Results

We compared the Apache Storm default scheduler and our machine learning-based scheduler. First, we observed the latency distribution when multiple Topology were executed, as shown in Figure 6. Figure 6 shows that our scheduler latency distribution is more stable and gathered. The default scheduler's latency is more separated, and the maximum latency is higher than that of a machine learning-based scheduler's maximum latency because nodes perform the task differently. When the Topology increases, the variation between the two schedulers is more prominent.

Additionally, we compared the average latency of the cluster when two schedulers work. Figure 7 shows the line graph of the average latency for two schedulers. We can see the blue line, which represents the default

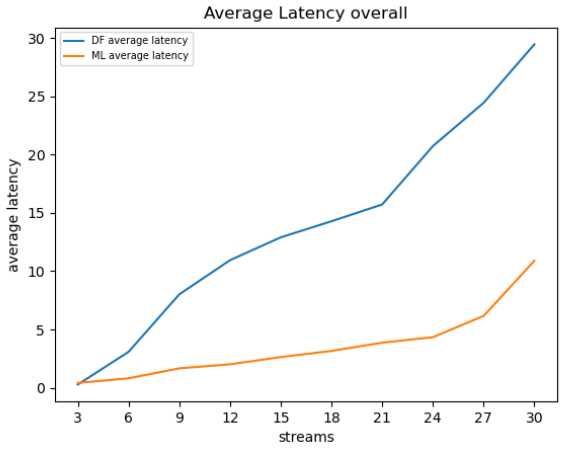

Figure 7. The average latency for two schedulers.

scheduler, is higher than the orange line, which is the machine-learning scheduler. As a result of Figure 7, we figure out that our machine learning-based scheduler increases the efficiency of the cluster overall. Our scheduler decreases latency by 60% compared to the default scheduler.

6. CONCLUSION

We proposed a machine learning-based task scheduling scheme for Apache Storm. This scheme solved the problem of existing schedulers assigning tasks without considering resource availability and task demand. We combined machine learning to predict the task's performance, and it benefited the master node by scheduling the task on the best-performing node. In summary, we successfully implemented our machine learning-based scheduler in Apache Storm. As a result of the experiment, our machine learning-based scheduler's latency performance is 60% lower than the default scheduler in Apache Storm. In the future, we will continue investigating how to utilize resources efficiently and minimize the application's latency. For the next step, we will add bandwidth as a machine learning training feature and investigate the impact on the distributed system. Additionally, the machine learning model could be changed to reinforcement learning. The reinforcement learning model could learn from the difference between performance prediction and actual performance. Instantly updating the model could make the performance prediction more accurate. In conclusion, this machine learning-based scheduler is an excellent beginning for us to keep researching task scheduling in distributed systems.

Acknowledgment

The work was supported under the Army contract W15P7T-21-C-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Army.

REFERENCES

[1] Van Steen, M. and Tanenbaum, A. S., [*Distributed systems*], Maarten van Steen Leiden, The Netherlands (2017).

[2] Donta, P. K., Murturi, I., Casamayor Pujol, V., Sedlak, B., and Dustdar, S., "Exploring the potential of distributed computing continuum systems," *Computers* **12**(10), 198 (2023).

[3] Iqbal, M. H., Soomro, T. R., et al., "Big data analysis: Apache storm perspective," *International journal of computer trends and technology* **19**(1), 9–14 (2015).

[4] Maheswaran, M., Braun, T. D., and Siegel, H. J., "Heterogeneous distributed computing," *Encyclopedia of electrical and electronics engineering* **8**, 679–690 (1999).

[5] Zou, Z., Chen, K., Shi, Z., Guo, Y., and Ye, J., "Object detection in 20 years: A survey," *Proceedings of the IEEE* **111**(3), 257–276 (2023).

[6] Zhao, Q., Feng, M., Li, L., Li, Y., Liu, H., and Chen, G., "Deep reinforcement learning based task scheduling scheme in mobile edge computing network," in [*Sensors and Systems for Space Applications XIV*], **11755**, 65–73, SPIE (2021).

[7] Feng, M., Zhao, Q., Sullivan, N., Chen, G., Pham, K., and Blasch, E., "Task assignment in mobile edge computing networks: a deep reinforcement learning approach," in [*Sensors and Systems for Space Applications XIV*], **11755**, 74–82, SPIE (2021).

[8] Zhang, W., Li, S., Liu, L., Jia, Z., Zhang, Y., and Raychaudhuri, D., "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in [*IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*], 1270–1278 (2019).

[9] Hahne, E. L., "Round-robin scheduling for max-min fairness in data networks," *IEEE Journal on Selected Areas in communications* **9**(7), 1024–1039 (1991).

[10] Peng, B., Hosseini, M., Hong, Z., Farivar, R., and Campbell, R., "R-storm: Resource-aware scheduling in storm," in [*Proceedings of the 16th Annual Middleware Conference*], *Middleware '15*, 149–161, Association for Computing Machinery, New York, NY, USA (2015).

[11] Li, C., Zhang, J., and Luo, Y., "Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm," *Journal of Network and Computer Applications* **87**, 100–115 (2017).

[12] Zhang, J., Li, C., Zhu, L., and Liu, Y., "The real-time scheduling strategy based on traffic and load balancing in storm," in [*2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*], 372–379, IEEE (2016).

[13] Aniello, L., Baldoni, R., and Querzoni, L., "Adaptive online scheduling in storm," in [*Proceedings of the 7th ACM international conference on Distributed event-based systems*], 207–218 (2013).

[14] Xu, J., Chen, Z., Tang, J., and Su, S., "T-storm: Traffic-aware online scheduling in storm," in [*2014 IEEE 34th International Conference on Distributed Computing Systems*], 535–544, IEEE (2014).

[15] Eskandari, L., Huang, Z., and Eyers, D., "P-scheduler: adaptive hierarchical scheduling in apache storm," in [*Proceedings of the Australasian Computer Science Week Multiconference*], 1–10 (2016).

[16] Java, S., "Java virtual machine," *Meyer et T. Downing* (2002).

[17] Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S., "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE* **105**(12), 2295–2329 (2017).

[18] Graves, A. and Graves, A., "Long short-term memory," *Supervised sequence labelling with recurrent neural networks* , 37–45 (2012).

[19] O'shea, K. and Nash, R., "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458* (2015).

[20] Hua, Y., Guo, J., and Zhao, H., "Deep belief networks and deep learning," in [*Proceedings of 2015 international conference on intelligent computing and internet of things*], 1–4, IEEE (2015).

[21] Hodson, T. O., "Root mean square error (rmse) or mean absolute error (mae): When to use them or not," *Geoscientific Model Development Discussions* **2022**, 1–10 (2022).

[22] Biehl, M., [*RESTful Api Design*], vol. 3, API-University Press (2016).

[23] Jocher, G., "Ultralytics yolov5," (2020).