Group 5: Load balancing after profiling straggler tasks

Khushi Mahesh
RIT
km1139@rit.edu

Ramprasad Kokkula
RIT
rk1668@rit.edu

Mona Anil Udasi
RIT
mu9326@rit.edu

Snehith Reddy Baddam
RIT
sb3994@rit.edu

ABSTRACT

In contemporary data centers, the optimization of job performance is hindered by the presence of heterogeneous hardware, leading to the occurrence of straggler tasks on slower resources. This project explores load balancing after profiling straggler tasks, focusing on the Negative-Unlabeled learning approach with Reweighting and Distribution-compensation (NURD) [6] algorithm. NURD [6], designed for non-straggler bias, will be simulated on the Alibaba 2017 trace [2]. Our objective is to seek inspiration from NURD [6] in identifying stragglers within distributed environments that execute iterative workloads. The proposed load-balancing algorithm adapts dynamically to changing conditions in iterative patterns, aiming to enhance system performance. Integrating the Weighted Least Outstanding Requests (WLOR) [15] algorithm, the approach dynamically adjusts task distribution based on identified stragglers, further optimizing resource utilization. The challenges encountered in this endeavor include evaluating NURD [6] on real-world workloads like the Alibaba 2017 trace [2] , addressing complexities associated with distributed setups, and managing induced heterogeneity.

KEYWORDS

Load balancing, Straggler, Machine Learning

ACM Reference Format:
Khushi Mahesh, Ramprasad Kokkula, Mona Anil Udasi, and Snehith Reddy Baddam. Year. Group 5: Load balancing after profiling straggler tasks. In *Proceedings of Conference Name*. ACM, New York, NY, USA, 8 pages. https://doi.org/DOI

1 INTRODUCTION

In the contemporary landscape of data centers, the pursuit of optimal job performance encounters a significant hurdle: the presence of heterogeneous hardware often leads to the emergence of straggler tasks, which impede efficiency and degrade overall system performance. These stragglers, characterized by their slower execution on specific resources, pose a huge challenge to system administrators and engineers striving for seamless operation within distributed environments.

Conference Name, Month, Year, Location

ACM ISBN ISBN
https://doi.org/DOI

Recognizing this challenge, our project embarks on an exploration of load balancing strategies tailored specifically to address the issue of straggler tasks. At the heart of our investigation lies the Negative-Unlabeled learning approach with Reweighting and Distribution-compensation (NURD) [6] algorithm, meticulously designed to confront non-straggler biases and excel in identifying and handling straggler instances within distributed computing frameworks. Our endeavor is motivated by the imperative to devise solutions that can navigate the complexities inherent in contemporary data center architectures. The crux of our approach lies in devising strategies that dynamically adapt to changing conditions, thereby optimizing resource utilization and enhancing overall system efficiency.

Our research is rooted in the observation that real-world workloads, exemplified by the Alibaba 2017 trace [2], present nuances that demand careful innovative solutions. By embracing the challenges posed by these real-world datasets, we aim to bridge the gap between theoretical algorithms and practical implementation. Throughout this endeavor, we confronted a myriad of challenges, ranging from the intricacies of evaluating algorithms on real-world workloads to the complexities of managing distributed setups and heterogeneous hardware configurations.

In this report, we delve into our methodology, dissect the challenges encountered along the way, and present the results of our analysis and simulations. By shedding light on the effectiveness of the NURD [6] algorithm in identifying and mitigating straggler instances, and proposing a dynamic load-balancing approach bolstered by the Weighted Least Outstanding Requests (WLOR) [15] algorithm, we contribute to a growing body of knowledge aimed at enhancing the efficiency and resilience of modern data center infrastructures.

In this work, we plan to study the following: (1) Simulate the NURD [6] algorithm on the Alibaba trace [2] from 2017 (2) Leverage the NURD [6] algorithm for identifying the straggler tasks in our distributed setup environment running reference workloads of iterative nature (e.g. PyTorch, TensorFlow, Spark, etc.); and (3) Design and develop techniques to remap the workload tasks to non-straggler resources. The focus of this work is on applications characterized by iterative patterns. In these scenarios, workloads are divided into epochs, and the proposed load-balancing algorithm is designed to reevaluate and adjust resource allocations at the end of each iteration. This iterative approach enables the system to adapt dynamically to changing conditions and evolving straggler

profiles.

We have shared our code at https://github.com/Sin317/652-ds-project

2 CHALLENGES

The primary challenge stemmed from evaluating the NURD [6] algorithm on real-world workloads, particularly the Alibaba 2017 trace [2]. While the trace provides valuable insights into cluster dynamics and task behavior, its application in simulating and testing load-balancing algorithms introduces complexities. Real-world workloads often exhibit diverse characteristics and complexities, necessitating careful consideration and adaptation of algorithms to ensure robust performance. Resource limitations, including computational resources such as GPU, CPU, and memory, impose constraints on the simulation and analysis process. Preprocessing large-scale datasets, such as the Alibaba trace [2], and conducting extensive simulations require significant computational resources and may necessitate trade-offs in terms of dataset size, simulation duration, and analysis depth. Balancing the need for comprehensive analysis with computational constraints presents a practical challenge in research endeavors of this nature.

3 RELATED WORK

The following papers have helped in understanding the straggler problem and the different load-balancing strategies:

- Factors Causing Stragglers [7]
- Efficient fair queuing using deficit round-robin [12]
- Strategies for dynamic load balancing on highly parallel computers [16]
- Straggler mitigation at scale [1]
- Addressing the straggler problem for iterative convergent parallel ML [8]
- Execution Analysis of Load Balancing Algorithms in cloud computing Environments [14]
- Semi-Dynamic Load Balancing: Efficient Distributed Learning in Non-Dedicated Environments [4]
- START: Straggler Prediction and Mitigation for Cloud Computing Environments using Encoder LSTM Networks [13]
- Load balancing with memory [10]
- Load balancing algorithms in cloud computing: A survey of modern techniques [3]
- Load balancing algorithms in cloud computing: A survey [9]
- Observations on using genetic algorithms for dynamic load-balancing [17]
- DBSCAN Clustering Algorithm Based on Density [5]
- A bagging SVM to learn from positive and unlabeled examples [11]

3.1 Factors Causing Stragglers

An in-depth analysis categorizes dominant factors leading to straggler occurrences, including high CPU utilization, disk overloading, unbalanced workload aggregation, poor user code, server faults, network conditions, and task request surges. Insights from the study provide guidance for handling various straggler scenarios in cloud data centers through historical data analysis and real-time monitoring using offline and online analytics components. [7]

3.2 Efficient fair queueing using deficit round-robin

Fair queuing is a technique that allows each flow passing through a network device to have a fair share of network resources. This paper talks about a new approximation of fair queuing, called as deficit round-robin. This achieves nearly perfect fairness in terms of throughput, requires only O(1) work to process a packet, and is simple enough to implement in hardware. [12]

3.3 Strategies for dynamic load balancing on highly parallel computers

The analysis compares five dynamic load balancing strategies tailored for highly parallel systems to minimize total job completion time. The Dimension Exchange Method (DEM) stands out, with variable efficiency based on system topology. Strategies include Sender (Receiver) Initiated Diffusion (SID-IRID), Hierarchical Balancing Method (HBM), Gradient Model (GM), DEM, and Load Balancing Profitability Determination. They address aspects like global load balancing, hierarchical organization, proximity-based migration, dimension-based synchronization, and profitability assessment, collectively enhancing load balancing efficiency in highly parallel systems. [16]

3.4 Straggler Mitigation at scale

This paper presents a cost (pain) vs. latency (gain) analysis of executing jobs of many tasks by employing replicated or erasure coded redundancy. The tail heaviness of service time variability is decisive on the pain and gain of redundancy and we quantify its effect by deriving expressions for cost and latency. [1]

3.5 Mitigation of the straggler problem for iterative convergent parallel ML

FlexRR [8] addresses the straggler problem in distributed machine learning (ML) environments with an adaptable synchronization model that dynamically reallocates work among worker threads, ensuring scalability and efficiency. Unlike traditional Bulk Synchronous Parallel (BSP) methods, FlexRR mitigates the impact of stragglers on overall performance through peer-to-peer work reassignment. Experimentation on Amazon EC2 and Microsoft Azure platforms, along with stress tests simulating straggler behavior, validate FlexRR's efficacy. It consistently achieves near-optimal runtimes across diverse straggler scenarios, demonstrating significant reductions ranging from 15% to 56% compared to existing approaches. FlexRR's integration of flexible consistency bounds and temporary work reassignment strategies outperforms BSP and SSP methods, making it a promising solution for optimizing distributed ML workflows and enhancing performance and productivity. [8]

3.6 Execution Analysis of Load Balancing Algorithms

The analysis compares various load-balancing algorithms in cloud computing, considering factors like stability, resource utilization,

scalability, and cost-effectiveness. Token Routing minimizes costs but struggles with scalability, while Round Robin offers simplicity but may lead to uneven loads. Randomized algorithms allocate processes based on static probabilities, posing challenges in handling workload complexities. Central Queuing buffers requests dynamically, and the Connection Mechanism uses least connection algorithms for load balancing. Despite diversity, challenges persist in managing non-uniform load arrivals, emphasizing the need for comprehensive evaluations. Additionally, algorithms are classified as static or dynamic based on workload assignment timing, each with its benefits depending on system requirements and environmental conditions. [14]

3.7 Semi-Dynamic Load Balancing: Efficient Distributed Learning in Non-Dedicated Environments

The paper introduces LB-BSP, a novel load balancing strategy for distributed machine learning (ML) workloads. LB-BSP aims to eliminate stragglers—slow workers—in clusters with heterogeneous resources. It balances ML workers at iteration boundaries without disrupting intra-iteration execution, adapting load based on workers' processing capabilities. LB-BSP is implemented as a Python module for TensorFlow and PyTorch and accelerates distributed training by up to 54% on Amazon EC2. Overall, LB-BSP offers a practical and effective solution for improving model convergence in non-dedicated clusters.[4]

3.8 START: Straggler Prediction and Mitigation for Cloud Computing Environments using Encoder LSTM Networks

The paper introduces START, a pioneering Straggler Prediction and Mitigation Technique engineered to tackle performance bottlenecks stemming from straggler tasks within large-scale computing systems. START leverages an innovative approach by employing an Encoder Long-Short-Term-Memory (LSTM) network. This neural network architecture not only predicts potential stragglers but also actively mitigates their impact by dynamically adjusting scheduling strategies to optimize response times. Through extensive simulation conducted in a cloud computing environment, START undergoes rigorous comparison with contemporary techniques, showcasing its superior performance. Results demonstrate significant reductions in execution time, alleviated resource contention, lowered energy consumption, and mitigated SLA violations. These advancements ultimately contribute to an enhanced overall system performance and improved quality of service, solidifying START as a pioneering solution for addressing straggler-related challenges in large-scale computing environments.[13]

3.9 Load balancing with memory

The paper explores a load balancing model where balls are sequentially placed into bins, with each ball having a choice of d possible locations. Introducing memory into the model, where the least loaded choice of the previous ball is remembered and considered for the next ball's placement, significantly improves performance. For instance, with just one random choice and one choice from memory, the maximum number of balls in a bin is log log n/2 log /spl phi/ + O(1) with high probability, outperforming models with only random choices. This suggests that incorporating memory, akin to providing a small amount of choice, greatly enhances load balancing performance. The study also extends its findings to continuous-time variations resembling queueing systems, yielding similar improvements. [10]

3.10 Load balancing algorithms in cloud computing: A survey of modern techniques

The abstract highlights the importance of load balancing in cloud computing and outlines the research conducted on load balancing algorithms from 2004 to 2015. Various algorithms such as Round Robin, Min-Min, Max-Min, Ant colony, Carton, and Honey bee are compared based on characteristics like fairness, throughput, fault tolerance, overhead, performance, response time, and resource utilization. The paper aims to provide a structured overview of these algorithms, categorizing them based on their underlying models. The conclusion emphasizes the need for addressing issues like fairness and high throughput in load balancing algorithms and suggests exploring hybrid approaches for better performance and system security in the future. [3]

3.11 Load-balancing algorithms in cloud computing: A survey

This paper highlights the literature on the task scheduling and load-balancing algorithms and presents a new classification of such algorithms, for example, Hadoop MapReduce load balancing category, Natural Phenomena-based load balancing category, Agent-based load balancing category, General load balancing category, application-oriented category, network-aware category, and workflow specific category. [9]

3.12 Observations on using genetic algorithms for dynamic load-balancing

Genetic algorithms have gained immense popularity over the last few years as a robust and easily adaptable search technique. The work proposed here investigates how a genetic algorithm can be employed to solve the dynamic load-balancing problem. A dynamic load-balancing algorithm is developed whereby optimal or near-optimal task allocations can "evolve" during the operation of the parallel computing system. [17]

3.13 DBSCAN Clustering Algorithm Based on Density

Density-based clustering algorithms, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), excel in clustering data sets with irregular shapes and varying densities, making them particularly adept for scenarios where the data distribution is not well-defined. DBSCAN stands out as a classical method in this domain, renowned for its simplicity and efficiency in data clustering analysis. Unlike traditional methods like K-means, which assume spherical clusters of uniform density, DBSCAN identifies clusters based on the density of data points within their vicinity. It operates

by defining two parameters: epsilon ϵ, the maximum distance between two points to be considered neighbors, and minPts, the minimum number of points required to form a dense region. Through these parameters, DBSCAN efficiently identifies core points, which have a sufficient number of neighbors within ϵ, border points, which lie within the neighborhood of a core point but do not have enough neighbors themselves, and noise points, which are neither core nor border points. By iteratively expanding clusters from core points and incorporating border points, DBSCAN effectively partitions the data set into clusters of varying shapes and sizes, while gracefully handling outliers as noise points. This adaptability to diverse data distributions, along with its straightforward implementation, solidifies DBSCAN's status as a cornerstone in density-based clustering analysis.[5]

3.14 A bagging SVM to learn from positive and unlabeled examples

The proposed algorithm in the paper introduces a Bagging Support Vector Machine (SVM) approach tailored to handle scenarios where only positive examples are labeled, while the rest remain unlabeled. This innovative method iteratively trains multiple binary classifiers by leveraging subsets randomly sampled from the unlabeled data alongside the known positive examples. These classifiers focus on distinguishing between the labeled positive instances and the random subsamples of unlabeled data. By averaging the predictions of these classifiers, the algorithm effectively combines their insights, enhancing its overall predictive performance. Notably, the method demonstrates remarkable efficiency, particularly when confronted with extensive sets of unlabeled examples, outperforming existing techniques in terms of computational speed. This approach offers a practical solution for learning from positive and unlabeled data, showcasing its potential to expedite the analysis process in real-world applications. [11]

3.15 Summary

In the literature review, we explore dynamic load balancing and straggler mitigation in parallel computing environments, focusing on cloud computing and distributed machine learning (ML). It identifies factors contributing to performance issues and emphasizes the importance of real-time monitoring. Various load balancing strategies are analyzed, with the Dimension Exchange Method (DEM) highlighted for its efficiency. Innovative solutions like FlexRR and LB-BSP show promise in mitigating stragglers in distributed ML environments. Load-balancing algorithms in cloud computing are compared based on stability, resource utilization, and scalability. Challenges in managing non-uniform load arrivals are noted, emphasizing the importance of selecting appropriate algorithms. Different Machine Learning Algorithms to detect straggler tasks are also identified.

4 SYSTEM DESIGN

The load-balancing algorithm introduced in this extension operates in conjunction with the NURD [6] algorithm. It utilizes the output generated by NURD [6], which includes information about the resources exhibiting straggler behavior. This information forms the basis for determining the weight for the load balancing.

The weighted least outstanding requests (WLOR) [15] algorithm dynamically adjusts the distribution of tasks among computing resources based on the identified stragglers, optimizing resource utilization and minimizing the impact of performance variations.
A crucial aspect of the experiment involves exploring different configurations to identify the ideal update frequency for reallocating workloads to each worker process to ensure overall optimal utilization of resources.
The Alibaba trace data [2] "ClusterData201708" contains cluster information of a production cluster in 12 hours period: about 1.3k machines that run both online service and batch jobs. We use this dataset after modelling it as a time-series to predict straggler tasks based on the technique defined by [6].

About the dataset:
There were many jobs that were collected. Each job was split into different tasks. Every task could be further split into parallel computed batch instances (having the same type of computation, but ran on different input) that ran on different machines/containers. The tasks are going to be characterized as "stragglers" if any of the instance would use significantly more resources than the other instances.

4.1 Architecture

Figure 1 shows the order of events.
Figure 2 depicts the system design. The data is pre-processed to be in a live data format to simulate an online training-inference system similar to the NURD [6] paper. The pre-processed data is then sent to the Machine Learning (ML) Model to predict the straggler task and update the load balancer task. The ML algorithm chosen is DBSCAN and the straggler tasks will be relaunched by the load-balancing task. This ensures that the straggler nodes are not overloaded with tasks and slow down the entire computation. The simulation assumes that all resources are homogenous.

Algorithm 1 WLOR Algorithm

Input: List of servers S, List of tasks T
Output: Assignment of tasks to servers
Initialize an empty list L for the load of each server
Initialize an empty list A for the assignment of tasks to servers
for each server $s \in S$ **do**
    Calculate the load L_s of server s
    Append L_s to L
end for
Sort the servers in S based on their loads in L in ascending order
for each task $t \in T$ **do**
    Find the server s with the least load in S
    Assign task t to server s
    Update the load of server s in L
    Remove server s from S
    Sort the remaining servers in S based on their loads in L in ascending order
end for

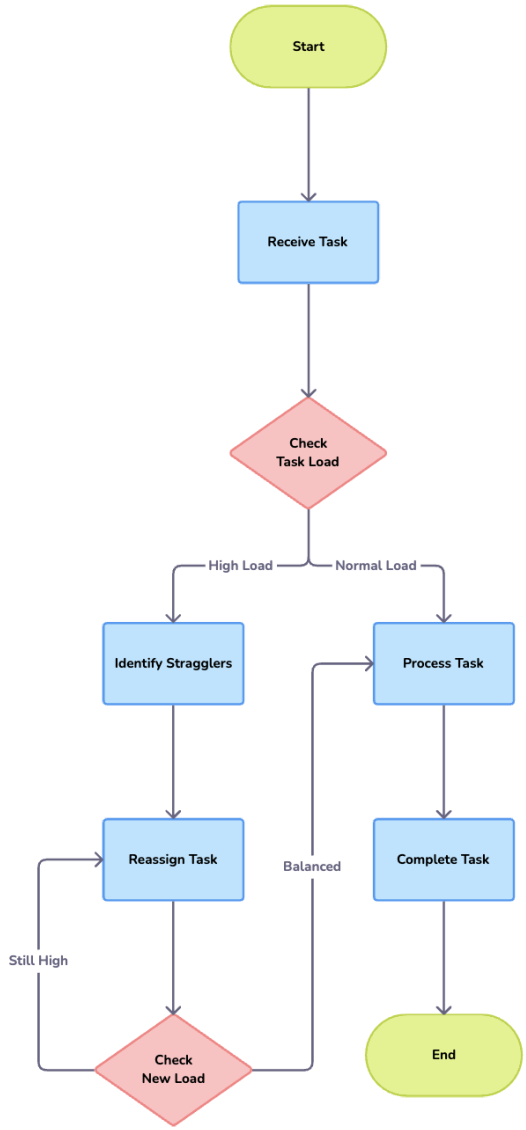


Figure 1: Flowchart

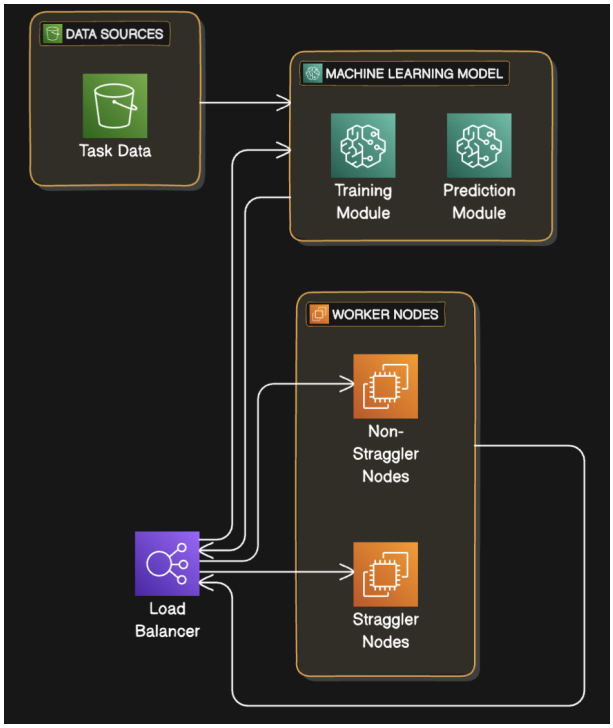


Figure 2: System design

5 PERFORMANCE EVALUATION

In our analysis, we have used the Alibaba cluster trace [2] dataset, specifically focusing on the 'batch_task.csv' and 'batch_instance.csv' files. We chose the Alibaba cluster trace [2] dataset due to its manageable size compared to the vast Google dataset, as preprocessing the latter was proving to be time-intensive. Additionally, considering our limited computational resources such as GPU, CPU, and RAM, we opted to work with a subset of the Alibaba cluster trace [2] dataset to ensure efficient processing and analysis.

Prior to commencing the analysis, we pre-processed the Alibaba cluster trace [2] dataset to ensure alignment with the subsequent phases of the research methodology. This preprocessing involved the development of a custom script to handle Unix timestamp conversion for the 'batch_instance.csv' file. This file contains records of instances within a batch processing system, including timestamps for instance start and end times, identifiers for jobs and tasks, machine IDs, instance status, trial sequence information, and metrics related to CPU and memory utilization.

Leveraging the Pandas library, the script facilitated the transformation of Unix timestamps into human-readable datetime objects for enhanced interpretability of temporal data. Additionally, the script assigned appropriate data types to each column, ensuring consistency and compatibility with subsequent analytical procedures. This pre-processing step laid the groundwork for subsequent transformation of the data into a time-series format conducive to analysis. Tasks were segmented into instances, each characterized by attributes such as start and end timestamps, machine IDs, status, and resource utilization metrics.

Subsequently, the dataset was partitioned into training and testing sets, with a specific focus on tasks and instances labeled as 'terminated' to identify potential straggler instances. Furthermore, we characterized selected tasks from jobs to identify their CPU usage, memory usage, and latency.

Figure 3 depicts this analysis. After predicting the straggler task from outlier detection (DBSCAN) [5], we used the Weighted Least Outstanding Requests (WLOR) [15] algorithm to determine which

machine could take on the straggler task.
In addition to the algorithmic analysis, we developed a simulation tool using Python and the Tkinter library for the graphical user interface. This tool allows users to input a time series dataset containing task information and resource usage metrics. It visualizes workload distribution among resources and simulates load balancing in real-time, providing insights into workload management strategies.
Overall, our analysis and simulation results contribute to a comprehensive understanding of the NURD [6]algorithm's capabilities in identifying and mitigating straggler instances within the selected trace.

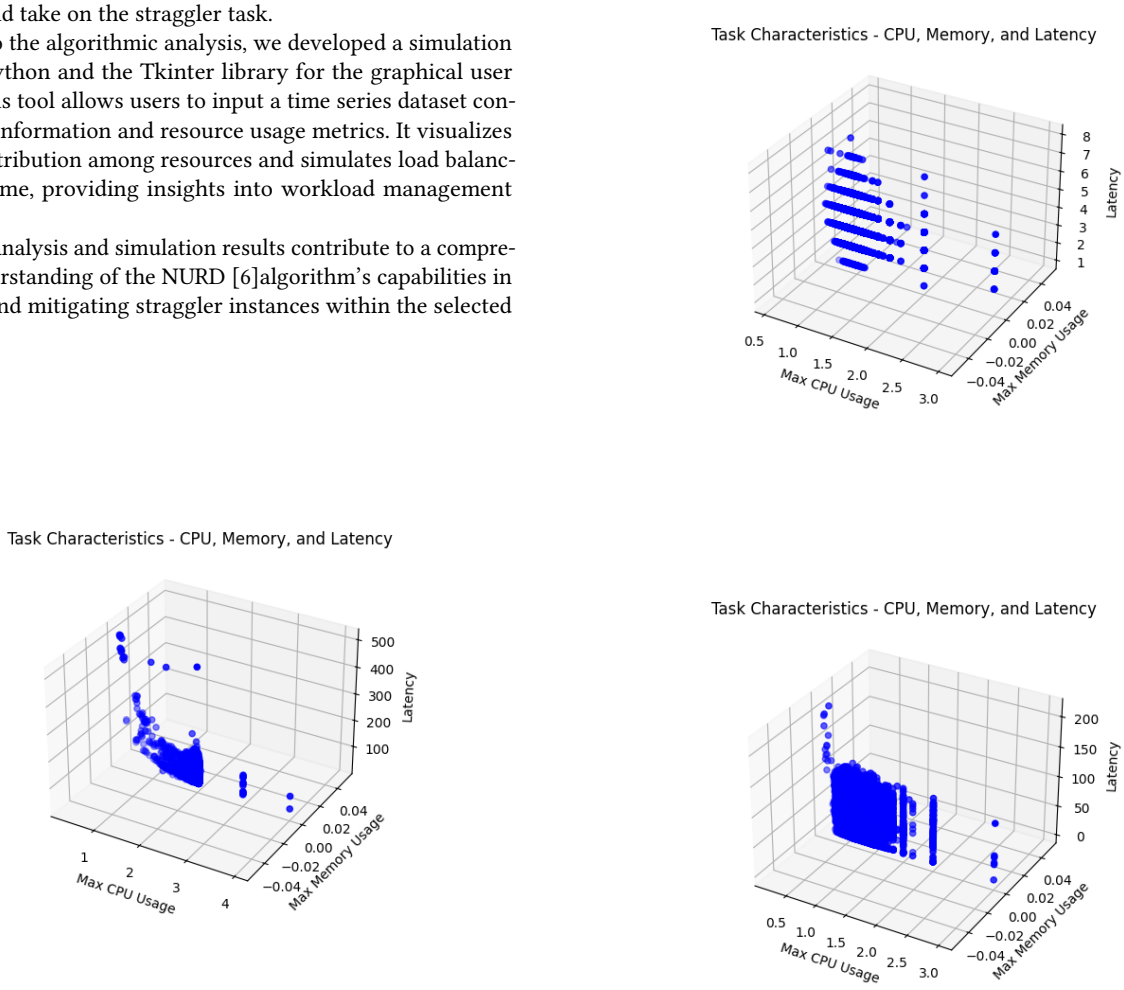
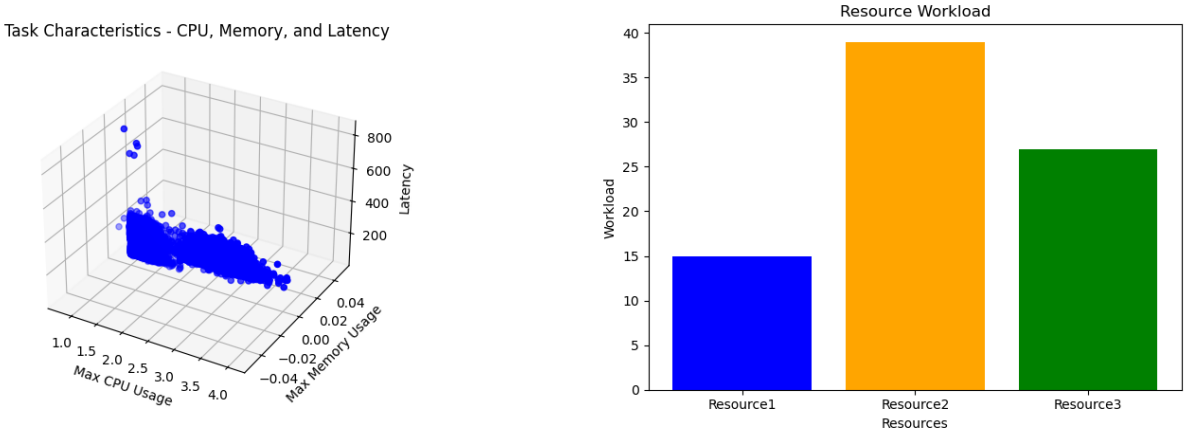


Figure 3: Task characteristics



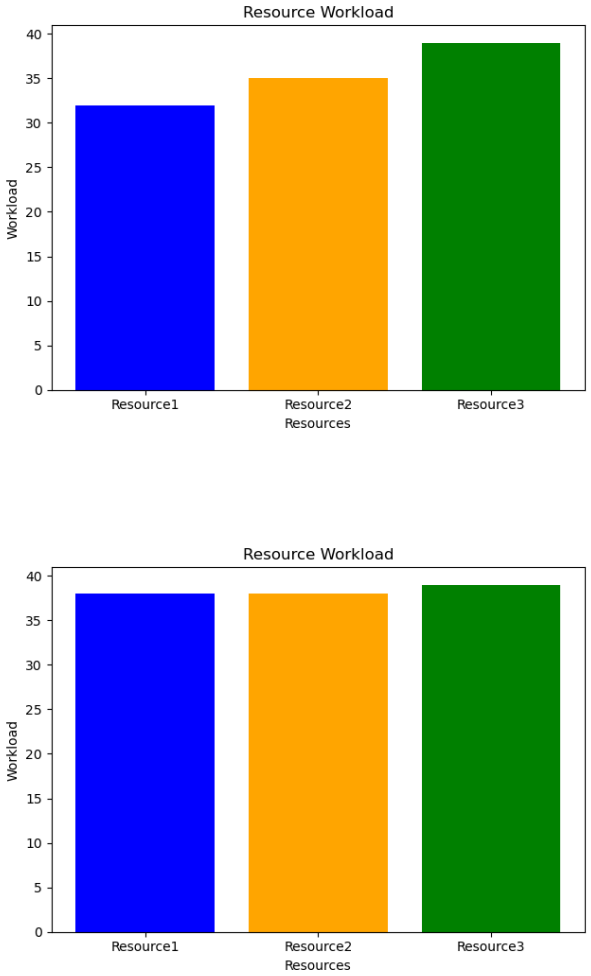


Figure 4: Resource utilization
shows sample result of load-balancing the straggler tasks identified in the Alibaba [2] trace dataset.

6 CONCLUSION

This project has delved into the realm of load balancing within heterogeneous data center environments, with a particular focus on mitigating the impact of straggler tasks on job performance. Leveraging the Negative-Unlabeled learning approach with Reweighting and Distribution-compensation (NURD) algorithm, originally designed to address non-straggler bias, we adapted its principles to identify stragglers within distributed setups executing iterative workloads. Through simulations on the Alibaba 2017 trace [2] and subsequent algorithmic analysis, we have demonstrated the efficacy of NURD in identifying straggler instances and proposed a load-balancing approach that dynamically reallocates tasks based on identified stragglers.

By preprocessing and analyzing the Alibaba cluster trace [2] dataset, partitioning it into training and testing sets, and developing custom scripts and simulation tools, we have paved the way for insightful findings. Our results showcase the potential of integrating the Weighted Least Outstanding Requests (WLOR) [15] algorithm with NURD to optimize resource utilization and minimize performance variations caused by straggler tasks. By dynamically adjusting task distribution based on identified stragglers, our approach aims to enhance system performance in iterative workload scenarios.

Furthermore, our simulation tool provides a valuable means of visualizing workload distribution and simulating load balancing strategies in real-time, offering insights into effective workload management techniques. Moving forward, further research and experimentation can build upon these findings to develop more robust and scalable solutions for load balancing in heterogeneous data center environments.

REFERENCES

[1] Mehmet Fatih Aktaş and Emina Soljanin. 2019. Straggler Mitigation at Scale. *IEEE/ACM Transactions on Networking* 27, 6 (2019), 2266–2279. https://doi.org/10.1109/TNET.2019.2946464

[2] Alibaba. [n. d.]. Alibaba trace. https://github.com/alibaba/clusterdata

[3] Sidra Aslam and Munam Ali Shah. 2015. Load balancing algorithms in cloud computing: A survey of modern techniques. In *2015 National Software Engineering Conference (NSEC)*. 30–35. https://doi.org/10.1109/NSEC.2015.7396341

[4] Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li. 2020. Semi-dynamic load balancing: efficient distributed learning in non-dedicated environments. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 431–446. https://doi.org/10.1145/3419111.3421299

[5] Dingsheng Deng. 2020. DBSCAN Clustering Algorithm Based on Density. In *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)*. 949–953. https://doi.org/10.1109/IFEEA51475.2020.00199

[6] Yi Ding, Avinash Rao, Hyebin Song, Rebecca Willett, and Henry (Hank) Hoffmann. 2022. NURD: Negative-Unlabeled Learning for Online Datacenter Straggler Prediction. In *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu (Eds.), Vol. 4. 190–203. https://proceedings.mlsys.org/paper_files/paper/2022/file/21d45762b38dbbd05c0ddbd6601d8f6c-Paper.pdf

[7] Peter Garraghan, Xue Ouyang, Renyu Yang, David McKee, and Jie Xu. 2019. Straggler Root-Cause and Impact Analysis for Massive-scale Virtualized Cloud Datacenters. *IEEE Transactions on Services Computing* 12, 1 (2019), 91–104. https://doi.org/10.1109/TSC.2016.2611578

[8] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. 2016. Addressing the straggler problem for iterative convergent parallel ML. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '16)*. Association for Computing Machinery, New York, NY, USA, 98–111. https://doi.org/10.1145/2987550.2987554

[9] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. 2017. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications* 88 (2017), 50–71. https://doi.org/10.1016/j.jnca.2017.04.007

[10] M. Mitzenmacher, B. Prabhakar, and D. Shah. 2002. Load balancing with memory. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* 799–808. https://doi.org/10.1109/SFCS.2002.1182005

[11] F. Mordelet and J. P. Vert. 2014. A bagging SVM to learn from positive and unlabeled examples. *Pattern Recogn. Lett.* 37 (feb 2014), 201–209. https://doi.org/10.1016/j.patrec.2013.06.010

[12] M. Shreedhar and G. Varghese. 1996. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking* 4, 3 (1996), 375–385. https://doi.org/10.1109/90.502236

[13] Peter Garraghan Rajkumar Buyya Giuliano Casale1 Nicholas R. Jennings Shreshth Tuli, Sukhpal S. Gill. 2021. START: Straggler Prediction and Mitigation for Cloud Computing Environments using Encoder LSTM Networks. IEEE Transactions on Services Computing. https://doi.org/10.1109/TSC.2021.3129897

[14] Ajanta De Sarkar Soumya Ray. 2012. EXECUTION ANALYSIS OF LOAD BALANCING ALGORITHMS IN CLOUD COMPUTING ENVIRONMENT. *International Journal on Cloud Computing: Services and Architecture* 2, 5 (2012). https://doi.org/10.5121/ijccsa.2012.2501

[15] Weikun Wang and Giuliano Casale. 2014. Evaluating Weighted Round Robin Load Balancing for Cloud Web Services. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 393–400. https://doi.org/10.1109/SYNASC.2014.59

[16] M.H. Willebeek-LeMair and A.P. Reeves. 1993. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems* 4, 9 (1993), 979–993. https://doi.org/10.1109/71.243526

[17] A.Y. Zomaya and Yee-Hwei Teh. 2001. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 9 (2001), 899–911. https://doi.org/10.1109/71.954620