Cluster Computing
https://doi.org/10.1007/s10586-024-04377-8

Analyzing the impact of various parameters on job scheduling in the Google cluster dataset

Danyal Shahmirzadi[1] · Navid Khaledian[2] · Amir Masoud Rahmani[3]

Received: 28 November 2023 / Revised: 20 February 2024 / Accepted: 22 February 2024

Abstract
Cloud architecture and its operations interest both general consumers and researchers. Google, as a technology giant, offers cloud services globally. This paper analyzes the Google cluster usage trace, focusing on three key aspects: task execution times, rescheduling frequency, and the relationship between task priority and rescheduling. Firstly, we examine how memory and processor performance impact task execution times across different machines. Next, we investigate how the number of task constraints influences rescheduling frequency and overall environmental efficiency. Furthermore, we analyze how task priority affects rescheduling and explore its correlation with task constraints. The results reveal that doubling the memory size can accelerate tasks by a factor of nine and that 90% of rescheduling is associated with tasks having less than seven constraints. We aim to enhance data center performance by identifying bottlenecks in the Google Cluster Dataset and providing recommendations for all cloud service providers. Our key findings indicate that memory plays a more significant role than the processor, and tasks with higher constraints have a less pronounced impact on rescheduling than anticipated.

Keywords Cluster computing · Cloud computing · Google cluster · Load balancing · Rescheduling · 80/20 rule

1 Introduction

The popularity of cloud services has increased in recent years, offering numerous advantages over personal or physical services, such as cost efficiency, scalability, flexibility, accessibility, collaboration, backup, and security [1]. However, as the number of users grows, the architecture of these services becomes more intricate. Cloud customers are becoming increasingly concerned about the availability and reliability of the services they purchase, driven by recent failures experienced by various software and infrastructure services. Reliability and availability concerns have emerged as significant challenges for traditional systems and new cloud services. The complexity of cloud architectures can amplify failure probabilities and decrease performance, particularly with the escalation of resources, including power consumption [2, 3].

In cloud systems, jobs are limited to a certain number of resources. If a job exceeds its limit, it will be killed or postponed, which can cause overhead for the system. Google's *autopilot* dynamic configuration helps to overcome this problem, but it still leaves 23% of the resources needed [4]. Most studies have concentrated on failure analysis, characterization, and prediction, with limited research conducted on bottleneck detection. This paper aims to develop a bottleneck detection framework to enhance cluster performance.

Failed jobs create bottlenecks by consuming substantial amounts of memory, CPU, disk, and power. The occurrence of failed jobs is directly linked to rescheduling, wherein a job is frequently placed in a cluster queue until completion. Historically, several companies, including

✉ Amir Masoud Rahmani
rahmania@yuntech.edu.tw

[1] Graduate School of Engineering Science and Technology, National Yunlin University of Science and Technology, 123 University Road, Douliou 64002, Yunlin, Taiwan

[2] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Esch-sur-Alzette, Luxembourg

[3] Future Technology Research Center, National Yunlin University of Science and Technology, Douliou, Yunlin, Taiwan

Published online: 29 March 2024

Twitter, LinkedIn, Amazon's Elastic service, and Microsoft's Azure HDInsight, should have capitalized on existing cloud systems to augment their capabilities with optimal flexibility [5]. Outdated technologies have since been replaced by a new data processing system called Cloud Dataflow. Over the past decade, Distributed Dataflow Systems (DDS) have emerged as a standard technology. Users employ constrained dataflow programming models, such as Cloud Dataflow, to scale program execution across a cluster of machines with a shared-nothing architecture. Upon closer examination, it becomes evident that Cloud Dataflow operates atop the Google Borg system.

Borg, a foundational system, introduces a crucial abstraction layer between the underlying physical hardware and its applications [6]. This strategic implementation empowers Google with efficient resource management capabilities and the ability to scale its applications to accommodate varying demands seamlessly. In contrast, Cloud Dataflow functions as a high-level system designed to facilitate distributed data processing. While it leverages Borg and other Google technologies, it maintains its autonomy as a separate system [7]. During our research, our primary focus was to delve into the logs generated by the Borg clustering system. All the Google services are running on this platform.

Recent resource management and models that can schedule workloads try to facilitate the sharing of computing resources in data centers. Efficiency and resource optimization are crucial for Cloud Service Providers; thus, many researchers have recently focused on this area [9–12]. Many researchers are trying to develop a state-of-the-art algorithm and middleware to make the most of these data center infrastructures. The realm of data centers and their resource management has garnered increasing attention recently, spurred by the rising prominence of novel concepts such as IoT (Internet of Things), edge computing, fog computing, and dew computing [17]. These emerging paradigms are gaining popularity for bringing computation and data processing closer to the source of data generation and consumption, thereby enhancing efficiency and reducing latency. It is worth noting that data centers serve as the cornerstone of cloud computing, providing the infrastructure and computational power essential for delivering cloud-based services and applications. In this context, the efficient operation and resource allocation within data centers play a pivotal role in ensuring the reliability and performance of cloud computing services.

Our study delves into the intricate domain of data center resource management, focusing on the Google Borg system—a fundamental component within Google's infrastructure. Borg is a linchpin in Google's ability to efficiently oversee and allocate resources, ensuring its applications' scalability and responsiveness to meet users' demands worldwide. Our exploration of the logs generated by the Borg clustering system offers valuable insights into the inner workings of this critical resource management system. Given the high cost of large-scale computing clusters, optimizing their usage is imperative. Optimizing needs deep knowledge of system behavior, and one of the best ways is to investigate current data centers' logs, like Google cluster traces produced by the Borg system and operators' experience in such environments [18]. In 2011, Google introduced a cluster trace dataset [19]. Subsequently, in 2019, another cluster dataset trace was released [20]. We investigated the Google cluster 2011, which contains the log for 12.5 k virtual machines, to find the bottlenecks and propose ideas to improve efficiency in big data centers. We chose this data set because more papers have investigated it in more detail [21–24]. The first Google dataset contains a trace of one Google cluster, while the second version includes workloads of eight different Google clusters. To the best of our knowledge, we are the first to investigate the impact of different resource types on data center scalability. We also identify the root cause of rescheduling, which can waste many resources.

These days, data centers, which need large capital investments, are utilized for many different things, including video processing, machine learning, search engines, and third-party cloud services. Most of the technologies we use are hosted by these data centers. Modern cluster management systems have evolved to manage these data centers effectively. Several businesses have made job-scheduling traces of their cluster management systems public so that outside academics can investigate how they accomplish this.

Borg is a prominent example among cluster management systems, having been meticulously developed and extensively employed by Google. Google's commitment to advancing external research in large-scale compute clusters was evident in 2011 when it released a 1-month trace from its Borg cluster management system. This initiative facilitated exploration by hundreds of researchers into scheduling intricacies [25]. A lingering question pertains to the evolution of workloads and its impact on scheduling decisions. To address this, Google has published a new "2019" trace, offering detailed Borg job scheduling information from eight distinct compute clusters throughout May 2019. This updated dataset extends the scope of research opportunities initiated in 2011 and provides a contemporary perspective on the dynamic interplay between evolving workloads and cluster management decisions within Google's infrastructure. Table 1 demonstrates the detailed comparison of these to traces.

The contemporary challenge lies in enhancing the overall efficiency of data centers by strategically reducing the time, energy consumption, and resource allocation

Table 1 Comparision of Google Borg traces

Date	May 2011	May 2019
Days	30	31
Machines	12,600	96,400
Priority	0–11	0–450
Format	CSV	BigQuery

required to execute a comparable volume of tasks. While most studies have traditionally focused on failure analysis, characterization, and prediction, this paper endeavors to pioneer a novel approach by introducing a bottleneck detection framework. The primary objective is to augment cluster performance, facilitating optimal distribution of resources such as RAM and CPU within the Borg system. This research aims to enhance system efficiency and resource utilization by emphasizing proactive bottleneck identification.

The research structure contains four different sections. Section 2 presents the scheme of Google cluster traces, provides basic statistics and behavior of failed jobs, and includes a comprehensive literature review on cluster research. Section 3 demonstrates our data analysis within four different phases. In the first phase, we investigated the effects of resources on task execution time and tried to find out that CPU has more impact on task execution time or memory. Rescheduling is an essential parameter in studying cluster efficiency; hence, in the next phase, we investigated the relationship between how the number of task constraints are related to rescheduling. In the third phase, we analyzed how task rescheduling is connected to tasks' priority. In the last step, we delve into task categories to determine how they relate to the number of tasks' constraints. Section 4 contains the study's conclusion and outlines future work.

2 Research questions (RQs)

In this study, our primary objective is to identify and address key considerations in optimizing data center performance and reducing energy consumption. The research questions outlined below guide our investigation into the factors influencing the efficiency of Borg systems.

RQ1 Which resource has a more significant impact on the Borg system—CPU or RAM?

The first data analysis aims to discern the relative impact of CPU and RAM on the performance of the Borg system.

RQ2 Do tasks with higher or fewer constraints predominantly influence rescheduling activities?

The second data analysis endeavors to uncover the primary factors affecting rescheduling, specifically examining whether tasks with higher or fewer constraints play a more significant role.

RQ3 Does the rescheduling of tasks correlate with their priority? Which type of task—higher or lower priority—exerts a more substantial impact?

The third data analysis addresses the interplay between task rescheduling and priority, investigating whether higher or lower-priority tasks have a greater influence.

RQ4 Is there a discernible relationship or correlation between task constraints and their respective categories?

The fourth data analysis explores potential connections or correlations between task constraints and their assigned categories. By answering these research questions, we aim to contribute insights that can inform strategies for optimizing data center performance, striking a balance between increased efficiency and reduced energy consumption.

3 Literature review

Fernández-Cerero et al. [8] attempted to extract workflows from raw log files in the Google cluster. The presence of numerous logs in the dataset resulted in a convoluted workflow resembling spaghetti, making it incomprehensible for humans. Process mining tools such as ProM and Disco were employed to investigate each process independently, creating more structured workflows. Additionally, the XES generator was utilized to modify the dataset format, enabling its use as input for the tools above. The researchers concluded that process mining tools faced challenges processing such extensive log data. To overcome this, it was necessary to split the data or use a subset of the top 5000 records from each log file as samples, as demonstrated in their study. The resulting workflows offer a more precise understanding for humans, allowing interpretation of information concealed in the logs, such as identifying users who submitted more jobs, determining the number of jobs assigned to each machine, and assessing the number of machines running a specific task.

Umer et al. [19] studied the transition of physical and virtual machines between different states. By understanding these data from logs, a data center operator can better decide to extend its infrastructure or push it into sleep mode to preserve energy. Their study discovered a 13% probability of another machine failure occurring on the same network switch within one minute of a previous failure. They suggested a Markov model for machine state prediction, enabling accurate forecasting of machine states over an extended period by leveraging estimated

probabilities. Their proposed model revealed the active machines' trend and found a 1.76% bias when matching it with the data set.

In [20], authors investigated the failures in Google cluster logs that usually happen because of the data center's large scope. The failures should be avoided because they will reduce performance and waste our resources. They proposed a failure prediction platform to detect a job failure before it happens. Using machine learning technics, they could successfully submit a method that predicts failed jobs even before Borg's operating system scheduling. They first developed a job status prediction model with 98% accuracy, but they could increase the accuracy to 99% for job failure prediction by selecting specific manual parameters.

Wang et al. [21] employed an enhanced mixture of Gaussian (GMM) algorithms to predict missing tasks. The Google cluster dataset includes a field for missing information with various values. They identified scheduling class, priority, and resource requests as three attributes significantly impacting missing information fields in the task log file. Both tasks and jobs share a common attribute: scheduling, providing a general indication of the task's sensitivity to latency; the priority attribute indicates a task's priority level. Additionally, each task specifies its own 'CPU' and 'Memory' requirements, detailing the resource limitations within the data center infrastructure.

They proposed an algorithm capable of predicting the value of a missing type with an accuracy of 99.73%. Subsequently, they recommended transferring that task to a real-time monitoring section to mitigate potential side effects on other tasks. While they demonstrated commendable performance in predicting missing information types for tasks, their analysis focused solely on individual tasks and did not delve into their potential impacts on other tasks.

Njuguna Ngang'a et al. [22] created an improved cloud failure prediction model using Adaboost ensemble Machine Learning algorithms capable of predicting hardware and software failures. The model was developed using Google Cluster 2019, Azure Clouds, and Alibaba Clouds datasets. Their model utilizes an ensemble classification approach incorporating random forest, decision tree classifiers, and regression. The result shows a slightly higher accuracy rate than previous studies in this field. The decision Tree Classifier produced the most favorable average model performance results by 91.7% compared to previous work done by Soualhia et al. [23], which reached 85.6% accuracy. They also noticed that Adaptive Boosting improved the overall model accuracy across all classifiers and datasets, except for Logistic Regression in the Azure dataset, which did not enhance the overall model performance. Regarding convergence time, the decision tree classifier had the shortest average time, followed by logistic regression and the random forest classifier.

In [24], Chen et al. believe that most of the previous works in this field focused on how they can allocate resources fairly while neglecting the efficiency of data centers. However, unequal usage of data center infrastructure lowers resource utilization but significantly impacts cloud application service quality. They defined data center resource utilization as a problem of balancing efficiency and balance, considering cluster systems' dynamic, discrete, and heterogeneous nature. Simulations conducted on the Google cluster dataset indicate that their proposed algorithm can enhance data center infrastructure usage while ensuring all users have the right to access resources.

Gupta et al. [26] analyzed the long-range dependence nature of cloud resource workloads using autocorrelation and rescaled range analysis methods. They investigated the presence of long-range dependence in cloud workloads and provided experimental evidence for its origins. They also used the Google cluster trace for analysis as a standard real dataset and used metrics such as arrival, service distributions of jobs, and resource usage. They showed that these metrics demonstrate heavy-tailed behavior, and a mathematical formulation proves that aggregate workload exhibits long-range dependence. Finally, they suggested that their analysis provides crucial information that can facilitate designing optimal resource management policies for cloud workloads. Loo et al. [15] proposed a scalable infrastructure to analyze the operation logs of cloud environments. They analyzed the dataset for workload characterization, considering the decision tree classifier had the shortest average time, followed by logistic regression and then the random forest classifier, their proposed infrastructure.

In their study [27], Subramanian et al. introduced a novel algorithm named CDB-LSTM, which is designed to predict resource usage in data centers. The authors sought to identify informative samples by leveraging the hypergraph concept and applying specific filters while disregarding outlier data. The proposed model provides a more accurate estimation of future resource usage and minimizes virtual machine relocations. Additionally, it selects an appropriate destination server using a correlation coefficient measure. The authors validated their model using the Google cluster dataset within a simulation environment. The results demonstrated the model's effectiveness in reducing the number of virtual machine migrations, thereby enhancing performance while conserving energy.

4 Data analyzing

The "Google Cluster Trace Usage 2011" is a dataset containing a comprehensive record of the resource request and usage data of Google's data centers over one month in May 2011. The dataset includes information on the usage

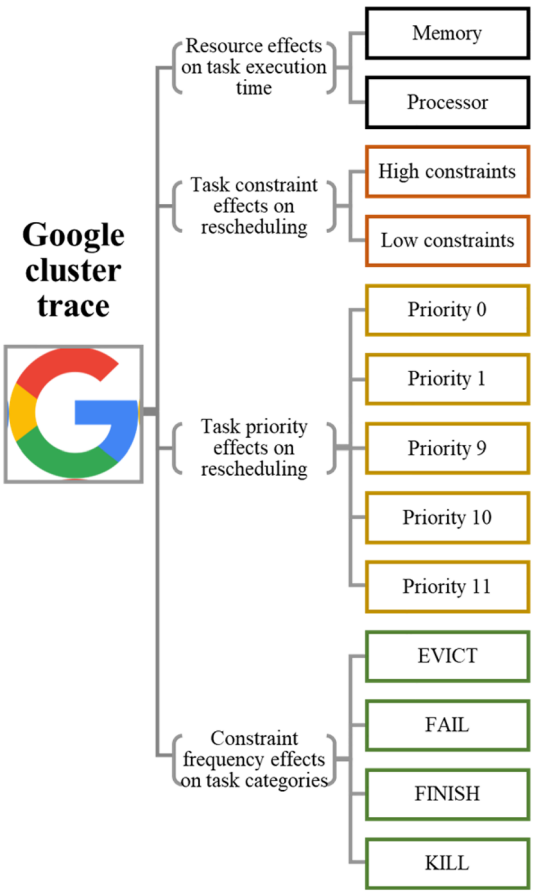


Fig. 1 The research framework

of CPUs, memory, and disk I/O for tens of thousands of machines and network usage data for millions of flows. At the same time, it does not have information about end users, their data, or even their access patterns to systems and services. The dataset is available on Google storage and is 41 GB in size. Researchers can download it using the gsutil command line tool provided by Google. It covers many workloads, including batch jobs and web-serving jobs. This dataset is intended to help researchers better understand the characteristics of large-scale distributed systems and to facilitate the development of more efficient and reliable computing infrastructures.

Four different analyses of the tables included in the "Google cluster trace usage 2011" were investigated in this study, as illustrated in Fig. 1. The first analysis reveals the effects of resources on task execution time and shows how we can increase the cluster performance with the least expenses. The second analysis tries to figure out the role of task constraints on rescheduling; by reducing the number of rescheduling, we can reduce resource waste and increase cluster performance. Which one makes the bottleneck: tasks with higher priority or lower ones? The third phase tries to find the best answer to this question. Finally, different categories of tasks and their relation with task constraints are investigated in the last phase. However, in some phases, the data need preprocessing to eliminate outliers. The Pareto rule also demonstrates the relationship between factors impacting scheduling and rescheduling. Some revealed that relations between task type and rescheduling highly contrast expectations.

4.1 First data analysis

This analysis presents average task execution times on various machines based on memory and processor performance. The goal is to determine how memory and processor affect task execution time. First, using the columns of event type and time stamp in the table of machine events, the service duration of the machines in the whole 29 days of data has been calculated. Therefore, first, we have considered all the machines whose event type column shows zero, which means they were added to the environment as the machines in the environment, and we have placed them in a new table called machines-service-time. Then, using the time of the two events of adding (0) and removing (1) the machine from the environment, the desired service time is calculated as the service period of a machine with a certain amount of memory and processor.

Due to update events, machines' memory and processor sizes may change multiple times during the test period. Consequently, after each update event, we treated each machine as new with different specifications and service duration. The time between addition and update events reflects the service duration of such machines. Our new table, "machines-task-time," has eight combinations of machines for different amounts of memory and processors. We count the total service time of each type of machine and the overall number of tasks performed by a distinct virtual machine. The average time to run each task by a specific machine can be calculated by (total machine service time)/(all unique tasks performed by a machine type). Table 2 shows the final results.

Here, we analyzed the first four groups from the table to provide evidence supporting the statement that main memory has a more significant impact on a machine's task capacity and performance compared to the processor. Upon examining the data for the first four groups, which all share the same CPU capacity of 0.5, we can discern substantial variations in their average times per task.

In Group 1, where the memory capacity is 0.03085, the average time per task is remarkably high at 1,417,500. This suggests that machines with limited memory capacity experience significant performance bottlenecks, even when the CPU capacity remains constant. Moving to Group 2, with a slightly higher memory capacity of 0.06185, we see a noticeable reduction in the average time per task, which

Table 2 Tasks average time per machine type

Group number	CPU capacity	Memory capacity	Total machines	Average time/per task
1	0.5	0.03085	6	1,417,500
2	0.5	0.06185	3	154,797
3	0.5	0.1241	97	10,872.9589
4	0.5	0.2493	10,188	5276.779295
5	0.25	0.2498	510	3975.903614
6	0.5	0.749	2983	2502.837684
7	1	1	2218	2178.148921
8	0.5	0.4995	21,731	1856.602755

stands at 154,797. This performance improvement suggests that a modest increase in memory capacity can lead to substantial gains in task efficiency, reinforcing the importance of memory. Group 3, with a memory capacity of 0.1241, exhibits further improvement in task performance, with an average time per task of 10,872.9589. The data underscores that as memory capacity increases, task execution becomes increasingly efficient, even with the same CPU capacity.

Lastly, in Group 4, where the memory capacity jumps to 0.2493, we observe a significant drop in the average time per task, down to 5,276.779295. This compelling evidence reinforces the argument that memory capacity plays a pivotal role in influencing task execution time, as the gains achieved by upgrading memory capacity are pretty substantial, even when the CPU capacity remains constant. In summary, the analysis of the first four groups in the table strongly supports the notion that main memory has a more profound impact on a machine's task capacity and performance than the processor, as evident in the substantial variations in task execution times corresponding to different memory capacities while maintaining the same CPU capacity.

Table 2 shows that increasing machine memory reduces task execution time. Main memory impacts a machine's task capacity and performance more than the processor. Upgrading the main memory capacity has a greater impact on overall machine performance and efficiency. Table 2 shows that doubling the memory size results in at least nine times reduction in average task time, as seen by comparing the first two rows; even comparing rows 2 and 3 indicates a much higher (14.2 times) reduction in average task time by doubling the memory. Comparing rows 4 and 5 reveals a hidden truth about CPU in the dataset: Doubling the CPU capacity while slightly decreasing memory can't reduce the average task execution time but also increase it by 30%. Expanding the CPU while we don't have enough memory can negatively impact the performance. Expanding the CPU can speed up just CPU-bound tasks but has little impact on the overall cluster's performance.

In the cluster dataset, we observed that Google upgraded the server memory to its maximum capacity exclusively for servers with the highest CPU capacity, totaling 2218 virtual machines. However, by reallocating 25 percent of the memory from these 2218 machines—each with full CPU and memory capacity—and applying the additional memory to machines with a CPU capacity of 0.5 and memory of 0.2493 (comprising 10,188 machines in this dataset), we can significantly enhance the overall performance of Borg.

We have investigated new clusters, focusing on exploring the latest Google Cluster Dataset from 2019 (Fig. 2). This dataset commands a substantial size of approximately 2.4 terabytes, marking a significant increase in volume compared to its predecessor, which measured a mere 44 gigabytes. Managing data of such magnitude presents notable challenges regarding time and resource utilization.

To address these challenges, we harnessed the power of BigQuery, a service provided by Google Cloud. This allowed us to execute intricate queries on the Google Cluster Dataset without necessitating the download of the entire cluster trace. Utilizing the query illustrated in Fig. 3, we could extract comprehensive insights regarding the total number of machines and their corresponding execution times, all through the capabilities of Google BigQuery.

BigQuery uses a distributed architecture and algorithm, where the data is distributed across multiple servers, enabling parallel processing for faster query performance. We also utilized BigQuery ML, which allows users to build and execute machine learning models directly within BigQuery. Without the power of the cloud and its parallel algorithm, exploring big data like Table 2 is almost impossible. In contrast to conventional relational databases, BigQuery employs diverse parallel schemas to enhance the speed of query execution [28]. BigQuery is a serverless data warehouse supporting scalable analysis over large datasets (even over petabytes), which helped us calculate the average time and number of constraints in a reasonable time. Figure 2 demonstrates the pseudo code we

Fig. 2 Pseudo code for proposed BigQuery algorithm

```
# Step 1: Select all machines added to the cluster
SELECT *
INTO machines_added
FROM event_table
WHERE type = 0;  -- Machines added to the cluster
# Step 2: Select all machines removed or updated
SELECT *
INTO machines_removed_updated
FROM event_table
WHERE type IN (1, 2);  -- Machines removed or updated
# Step 3: Calculate machine service time
SELECT   machine_id,
  (machines_removed_updated.event_time - machines_added.event_time) AS service_time
INTO machine_service_time
FROM machines_added
JOIN machines_removed_updated
  ON machines_added.machine_id = machines_removed_updated.machine_id;
# Step 4: Group the machine_service_time table by CPU and RAM
SELECT   capacity.cpus,   capacity.memory,   SUM(service_time) AS total_service_time
INTO grouped_machine_service_time
FROM machine_service_time
GROUP BY
  capacity.cpus,   capacity.memory;
# Step 5: Calculate the total tasks performed by each machine
SELECT   machine_id,   COUNT(*) AS total_tasks
INTO total_tasks
FROM task_table
GROUP BY   machine_id;
# Step 6: Calculate the average time per task
SELECT   machine_id,   total_service_time / total_tasks AS avg_time_per_task
INTO avg_time_per_task
FROM grouped_machine_service_time
JOIN total_tasks
  ON grouped_machine_service_time.machine_id = total_tasks.machine_id;
```

proposed to reveal the average time per task for each group of virtual machines.

In our analysis of task constraint usage within the Google Cluster Trace 2011 dataset, as shown in Fig. 4 we employed another BigQuery to categorize individual constraint values into eight distinct ranges, from 0 to 180 or greater. Examining the distribution of tasks across these ranges revealed a predominant preference for lower constraints (0–59). However, we observed notably low frequencies in the higher constraint ranges, underscoring the significance of tasks with lower constraints. This categorization approach not only offers valuable insights into the characteristics of the workload but also sheds light on potential bottlenecks, particularly concerning tasks with constraints below 20.

It is worth noting that while this invaluable Google Trace Dataset is accessible free of charge, employing BigQuery for analysis does consume computational resources. Researchers should be mindful of potential associated costs when opting for this approach. All queries used in this paper are accessible by demand for further studies. The conclusive findings derived from our analysis of the Cluster 2019 dataset, as presented in Table 3, shed light on the notable influence of memory capacity on task execution time. This observation underscores the significance of memory resources in the context of computational tasks. A particularly striking insight emerges when we compare the last two rows of the table. In this comparison, we discern that, with the CPU already operating at its peak performance level, doubling the memory size leads to a noteworthy consequence—a task execution time increase of at least two-fold. This revelation underscores the pivotal role that memory capacity plays in shaping the efficiency and performance of computational tasks within the dataset.

The comparison between group B (machines with maximum CPU and half memory) and group A (machines with maximum CPU and maximum memory) reveals intriguing insights. Specifically, we observe that the number of machines falling into group B (as denoted by row 12) exceeds three times the count of machines in group A (row 13), indicating a significant disparity in distribution. Despite this numerical advantage, the performance of group B machines is markedly inferior, registering at less than fifty percent (48%) of the performance of group A machines.

This stark contrast prompts a compelling proposition: by consolidating every two machines from group B into machines with maximum memory and CPU, we could

Fig. 3 BigQuery command to calculate the number of machines with maximum CPU and Memory capacity in Google cluster 2019 trace

```
WITH MachineEventsWithDiff AS (
  SELECT
    machine_id,
    type,
    TIMESTAMP_MICROS(time) AS event_time,
    LAG(TIMESTAMP_MICROS (time)) OVER (PARTITION BY machine_id ORDER BY time) AS
prev_event_time,
    capacity.cpus AS cpu_capacity,
    capacity.memory AS memory_capacity
  FROM
    `google.com:google-cluster-data.clusterdata_2019_a.machine_events`
  WHERE
    type IN (1, 2,3)
    AND capacity.cpus = 1
    AND capacity.memory = 1
)
SELECT
  machine_id,
  SUM(event_duration) AS total_running_time
FROM (
  SELECT
    machine_id,
    type,
    event_time,
    prev_event_time,
    TIMESTAMP_DIFF(event_time, prev_event_time, MICROSECOND) AS event_duration
  FROM
    MachineEventsWithDiff
  WHERE
    type IN (1, 3)  -- Filter for 'add machine' and 'update machine' events
)
GROUP BY
  machine_id
ORDER BY
  total_running_time DESC;
```

Fig. 4 Categorization of task constraints using BigQuery

```
# Select relevant columns and create range buckets
SELECT
  CASE
    WHEN constraint BETWEEN 0 AND 9 THEN "0-9"
    WHEN constraint BETWEEN 10 AND 19 THEN "10-19"
    WHEN constraint BETWEEN 20 AND 29 THEN "20-29"
    WHEN constraint BETWEEN 30 AND 59 THEN "30-59"
    WHEN constraint BETWEEN 60 AND 89 THEN "60-89"
    WHEN constraint BETWEEN 90 AND 119 THEN "90-119"
    WHEN constraint BETWEEN 120 AND 149 THEN "120-149"
    WHEN constraint BETWEEN 150 AND 179 THEN "150-179"
  END AS constraint_range,
  COUNT(*) AS frequency

# Filter entries with constraints
FROM `google.com:google-cluster-data.clusterdata_2011`
WHERE constraint IS NOT NULL

# Group by constraint range and sort by frequency descending
GROUP BY constraint_range
ORDER BY frequency DESC;
```

effectively create 3642 machines optimized for performance. This consolidation would not only enhance overall system performance but also potentially lead to a reduction in power consumption.

The anticipated impact of this consolidation is profound. With an average time per task projected to be nearly 7271, the consolidated machines would boast significantly improved efficiency. Furthermore, this enhanced performance capacity, coupled with the surplus of 3,642 CPUs that are now idle, positions the system to handle the workload effectively with optimized resource utilization. Consequently, the consolidation presents a compelling

Table 3 Machine task-time—Google cluster 2019

Group number	CPU capacity	Memory capacity	Total machines	Average time/per task
1	0.386	0.166	2844	33,547
2	0.386	0.333	4948	31,029
3	0.479	0.25	4	17,515
4	0.591	0.166	1293	24,137
5	0.591	0.333	16,360	22,522
6	0.708	0.25	3	349
7	0.708	0.333	6973	21,799
8	0.708	0.666	2592	16,225
9	0.958	0.5	583	13,861
10	0.958	1	365	13,596
11	1	0.25	193	16,581
12	1	0.5	7285	14,852
13	1	1	2317	7271

strategy for maximizing system efficiency and minimizing resource waste.

Comparing rows 9 and 12 in Table 3 reveals an intriguing observation: despite solely increasing CPU capacity (as evident in the transition from 0.958 to full capacity) while maintaining memory at 0.5, the average time per task unexpectedly increases instead of decreasing. However, when we examine instances where memory is augmented, such as the comparisons between rows 9 and 10, 7 and 8, 11 and 12, and also row 13, a consistent pattern emerges: augmenting memory while keeping CPU capacity fixed consistently leads to a decrease in the average time per task. This finding underscores the potential for enhanced virtual machine (VM) performance within a cloud environment through memory upgrades.

Furthermore, it's noteworthy to consider the feasibility of capacity upgrades. Due to hardware limitations, adding CPU capacity is inherently more constrained than memory updates. Typically, servers can accommodate up to 4 CPUs, whereas they can support up to 24 memory slots, enabling configurations of up to 3 TB. This practical constraint suggests that expanding memory capacity is a more viable and scalable option for improving performance within cloud environments in real-world scenarios.

4.2 Second data analysis

This section investigates how the number of task constraints affects the rescheduling frequency and overall environment efficiency. Analyzing Google's tables and files poses a challenge due to their high volume of records and files. Processing all files in a table would be time-consuming and require substantial processing and main memory capacity. Therefore, we only examined approximately 20,000 tasks from the first day of a 29-day experiment. We identified tasks in the environment by filtering the task with event type zero (ADD) and created a separate table called "aggregated-tasks-constraints".

We matched the Job and Task ID columns from the Task table with the corresponding columns in the table of Task Constraints to determine all constraints for each task. In the first step, we obtained a table with 19,641 rows and two columns: the number of constraints and rescheduling for each task. Tasks with a value of one in the rescheduling column were registered only once and not rescheduled, so they were removed from the table. After using the Boxplot Adjusted method to eliminate outlier data from the restriction column, we obtained a table with 8,271 rows. The final result was derived after excluding tasks with only one scheduling (without rescheduling). Table 4 illustrates the distribution of tasks within the range of restrictions.

Figure 5 gives a better idea of task frequency within constraint ranges. The figure shows that 92.36 percent of tasks in the first three rows have less than 30 constraints. Hence, there are outlier data. Outliers are data points that deviate substantially from the rest of the observations, suggesting they may have been generated differently. They can be categorized into two groups: those resulting from data errors and those resulting from inherent data diversity [29]. The Boxplot method (Tukey) is suitable for identifying and eliminating outliers since it does not rely on assumptions about data characteristics [30]. The data in Table 3 is highly skewed, and using the Boxplot method would classify a significant portion (around 30%) as outliers and remove them. Thus, the *Adjusted* Boxplot method is preferred.

In this method, a powerful technique called MedCouple is usually used to measure skewness [28].

$$\text{MedCouple}(x_1, \ldots, x_n) = med \frac{(x_1 - med_k) - (med_k - x_i)}{x_j - x_i} \quad (1)$$

Table 4 Tasks frequency within constraint ranges

Constraint range	Task frequency	Cumulative frequency of tasks (%)
0–9	5674	68.60
10–19	1578	87.68
20–29	387	92.36
30–59	272	95.65
60–89	31	96.02
90–119	9	96.13
120–149	3	96.17
150–179	3	96.20
180–209	5	96.26
210–239	3	96.30
240–269	1	96.31
270–299	3	96.35
Higher	302	100

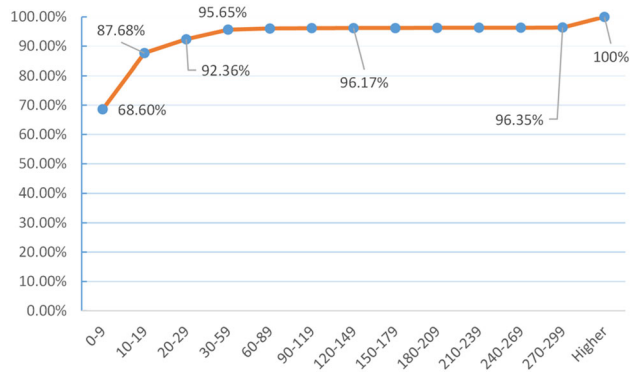


Fig. 5 Distribution of task constraints

$Subscriptmed_k$ represents the median of x_n while the value of i and j need to fulfill: $x_i \leq med_k \leq x_j$ and, $x_i \neq x_j$.

Given the large dataset size (8271) and the high time complexity of the calculation for the Boxplot method ($O(n^2)$), an alternative approach called the octile skewness method is used instead [31]. The octile skewness can be calculated by the formula (2) using quartile skewness described in [16]. Data skewness is a measure of the asymmetry of a distribution. A distribution is asymmetrical when its left and right sides are not mirrored. Skewness can be positive, negative, or zero. Octile skewness indicates whether the distribution is skewed to the left or right, while quartile skewness indicates whether the distribution is skewed upwards or downward.

$$\begin{aligned} &octileskewness \\ &= \frac{(Quartile_{0.875} - Quartile_{0.50}) - (Quartile_{0.5} - Quartile_{0.125})}{Quartile_{0.875} - Quartile_{0.125}} \\ &= \frac{(18-8)-(8-0)}{18-0} = 0.1111 \end{aligned} \tag{2}$$

With replacing octile skewness in the following equation with 0.1111, which we calculated above, the upper and lower bounds will be revealed to are 37 and − 11, respectively:

$$\begin{aligned} [L,U] = &\Big[Quartile_1 - 1.5 * e^{(-3.5octileskewness)} * IQR,\ Quartile_3 \\ &+1.5 * e^{(4octileskewness)} * IQR\Big]\ if\ octileskewness \geq 0 \\ =&[Quartile_1 - 1.5 * e((-4 * octileskewness)) * IQR,\ Quartile_3 \\ &+1.5 * e((3.5octileskewness)) * IQR]\ if\ octileskewness < 0 \end{aligned} \tag{3}$$

After removing outlier data located outside [− 11,37], the distribution of tasks within different constraints would be like Fig. 6, in which data is less skewed.

This figure illustrates that over 90 percent of tasks have less than 14 constraints. Using the Pareto rule, we can better understand these numbers.

In the context of Google Cluster and resource allocation, tasks with fewer constraints are prioritized more than their counterparts. These constraints, which encompass various user-defined properties influencing task placement, including machine specifications and task relationships, introduce a competitive dynamic within the scheduler. The scheduler's decisions are guided by ensuring that resources on the machine are optimally utilized while

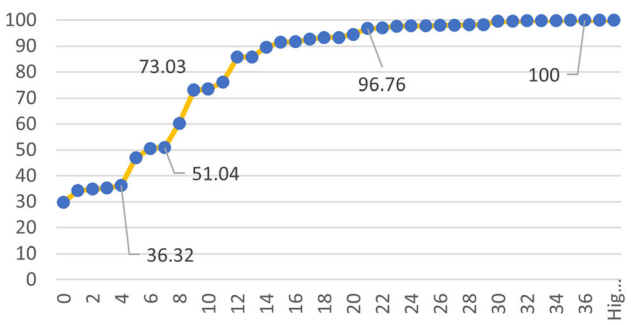


Fig. 6 Distribution of task constraints after removing outliers

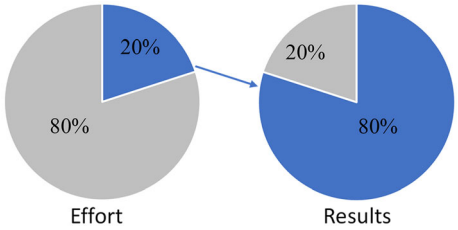


Fig. 7 Pareto rule

accommodating less-constrained tasks, which may need to be relocated, completed, or terminated to meet this objective [13].

4.2.1 Pareto Rule (20/80 concept)

Applying the 80/20 rule, we find that 20% of inputs, causes, or efforts often yield 80% of outputs, results, or rewards. This principle, as seen in Fig. 7, highlights that 80% of work achievements stem from just 20% of time spent defying common expectations [14].

The 20/80 concept refers to the likelihood of imbalance and balance in cause-and-effect data analysis, with imbalances occurring between forms such as 65/35, 75/25, 20.80, 95.5, and even 85/10. The sum of two numbers may not equal 100 sometimes. Figure 8 shows that 87% of environment restrictions are associated with tasks having over 7 and under 38 constraints, while only 13% are linked to tasks with 0–7 constraints.

But in Fig. 9, you can see that 90% of all rescheduling is related to the range of tasks with limits between 0 and 7 numbers.

Table 5 displays the number of restrictions and reschedules per range and their cumulative total.

Surprisingly, tasks with more restrictions could be more effective in rescheduling than expected. We discovered that tasks with fewer constraints can cause more rescheduling, and Google, in the future, can focus more on these tasks to investigate them, reduce the rescheduling, and increase the performance of its Clusters. Tasks with a higher number of restrictions contribute less to rescheduling because their priority is also higher.

4.3 Third data analysis

In the third step, we analyze how task priority affects rescheduling. The relationship between rescheduling and task priority will be revealed. We match the job ID and task index in the task events table, then count reschedules for each task to create a table with 19,641 rows. The priority will be in the first column, and the number of schedules in the next. Tasks with a value of one in the rescheduling column are removed from the table, as they have only been registered once and have yet to be rescheduled. Then, using the "Adjusted Boxplot", rows with values higher than 25 will be removed as outliers, giving a table with 8272 rows. Table 6 has 12 rows for priorities 0 to 11, showing the number of rescheduling for them. The number of reschedules is divided by the task frequency to determine

Fig. 8 Task restrictions frequency in two ranges

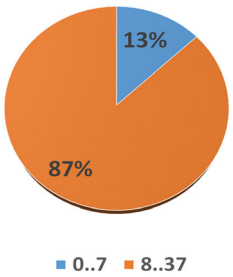


Fig. 9 Task reschedule frequency in two ranges

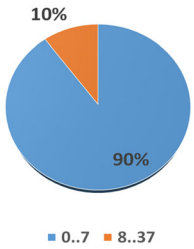


Table 5 Rescheduling in task constraint ranges

Task restrictions ranges	Constraint's count	Rescheduling count
8..37	47,551	7954
0..7	6927	72,140
Total	54,478	80,094

Table 6 Reschedules versus priority

Priority	Rescheduling per task
0	15.37662
1	5.153199
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	1.87736
10	3.852792
11	2.276276

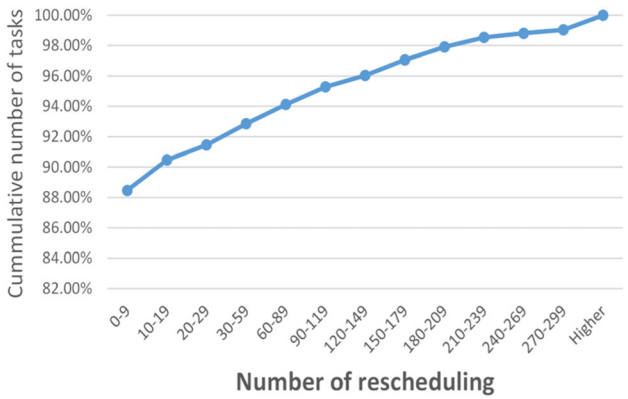


Fig. 10 Number of rescheduling

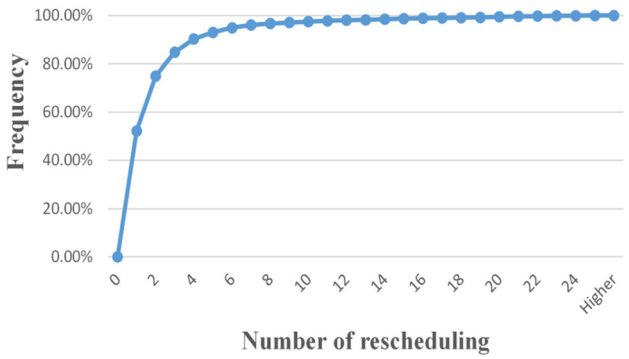


Fig. 11 Number of rescheduling after removing outliers

the average per task for priorities 0–11. No tasks were recorded for priorities 2 to 8 during the analysis period.

The new Table has 8271 rows that are right-skewed. The distribution of data in this table is demonstrated in Fig. 10.

We removed outlier data from the table using the "Adjusted Boxplot" method by setting upper and lower limits of 25 and zero. Data outside this range are considered outliers and deleted. This result is demonstrated in Fig. 11.

After outlier removal and calculating task rescheduling averages based on priority, Table 7 is obtained. It can be concluded that lower priority tasks that use resources and resources are withdrawn from them for the use of higher priority tasks are among the main factors of rescheduling in our study environment. The authors suggest that setting a minimum time access to the resources by tasks (especially those with the lowest priority) can significantly improve the cluster performance by reducing the number of rescheduling. The number of rescheduling for tasks with the lowest priority (0) is three times more than rescheduling tasks with priority (1). It means tasks with priority (0) can be easily postponed to run later because of their low importance, but this rescheduling itself can bring a significant overhead to the whole cloud. Considering a minimum resource access time for each task in the scheduling and rescheduling algorithm can increase cloud performance. This time can be adjusted during specific time intervals regarding rescheduling per task ratio within different priorities.

This principle holds within the Google Cluster Trace context, reflecting the intricacies of resource allocation and job scheduling. In this environment, jobs categorized in lower priority tiers must reside in the queue, patiently biding their time until higher-priority jobs release the compute resources essential for their execution. This orchestration ensures that resource utilization aligns with the priority hierarchy, optimizing the allocation of computational assets by the specific demands of each job and the overall system efficiency [15].

Table 7 Constraint and event type relations

Event type	Frequency	Sum of constraints	Constraint per event
EVICT	60,717	775,050	12.76
FAIL	7993	16,485	2.06
FINISH	263,497	5136	0.01
KILL	168,753	20,379,811	120.76

4.4 Fourth data analysis

This analysis investigated the relationship between the number of task constraints and their categories. First, we introduced the job and task life cycles and their corresponding event types.

Figure 12 demonstrates the state transition of a job or task in its entire life cycle. Each arrow indicates the transition between different states. There are two types of events; they either do the scheduling or change the task's state. Each job and task in the dataset has a number that reveals its event type. This number reveals what happened to a task after an event. In case of death, the event type will contain metadata about its reason. Event types can vary between 0 and 8. Submitted (0) means a task or job is ready for scheduling in the Borg system. Then, tasks will be scheduled (1) to run on a specific machine, but sometimes, they may get de-scheduled because of a higher priority task, which will go to the Evict (2) state. A task can be failed (3) or finished (4) successfully. Sometimes, a task will kill (5) by either user cancellation or because its dependent task has failed before. If a task is terminated, but we don't have its log in our database, we call it Lost (6). A task or job requirements or constraint may change while unsubmitted (Update-pending (7)) or even when it is scheduled: Update-running (8).

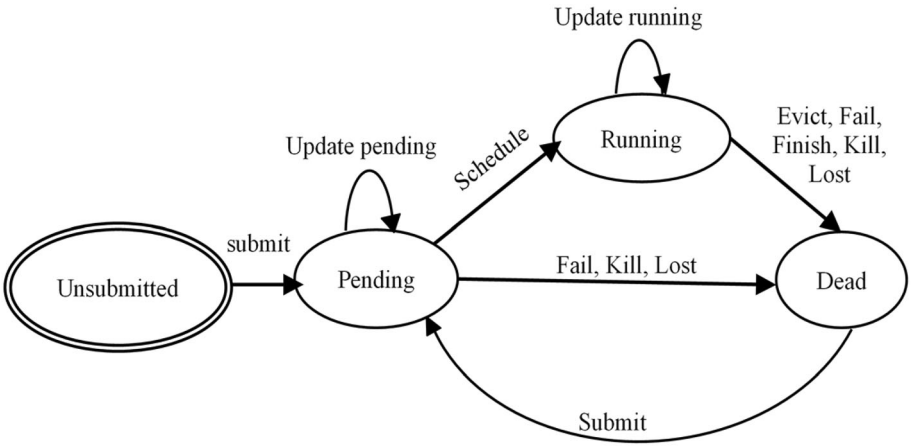


Fig. 12 State transition for jobs and tasks

In this section, we move the entries from the task event table whose type column shows the digits 2, 3, 4, or 5, representing EVICT, FAIL, FINISH, or KILL, respectively, into a new table. We only look at the roughly 500,000 tasks from the first ten days of the experiment to compensate for the massive amount of data in the task event table. We next count the number of execution constraints for each line of this table (per task) and record it in a column of this table using the table of task restrictions. In addition, the event type is counted. Lastly, we can determine the average limit for each task by dividing these two columns. Table 7 demonstrates the final result.

The result reveals 120.76 constraints per KILLed task, which is exceptionally high compared to other task types. Tasks with higher constraints are prone to be killed once before they finish. Killing a task or entire job causes significant overhead to the cluster. To bypass this overhead, cloud service providers can break the tasks with too many constraints into subtasks with fewer constraints, reducing rescheduling and increasing the cluster performance. If we decrease the number of killed tasks, breaking them into subtasks, tasks with evicted event type will also decrease because they are positively correlated.

We utilized the Google cluster jobs dataset, comprising 3,535,029 rows and six columns (Time, ParentID, TaskID, JobType, NrmlTaskCores, NrmlTaskMem), to develop a predictive model using XGBoost regression. We aimed to predict the normalized task memory based on other columns. Employing the parameters (objective = 'reg:squaredlogerror', n_estimators = 1000, learning_rate = 0.05, max_depth = 7), our analysis revealed that the Job type emerged as the most significant feature, contributing to 65% importance in predicting memory requirements. Following closely, NrmlTaskCores accounted for 15% importance, albeit considerably less influential than the Job title. This underscores the strong correlation between job type and memory allocation for each task, as shown in Fig. 13a.

In our second prediction model, we aimed to forecast normalized task cores using the information from the other five columns. Interestingly, our analysis revealed a significant correlation between normalized task cores and the ParentID column, followed by memory allocation, with job type ranking third. These findings suggest that prioritizing job types over CPU capacity could yield substantial benefits when considering memory augmentation within the Google cluster. This underscores the importance of implementing an optimized scheduler capable of allocating virtual machines with varying memory capacities to the corresponding job types, thereby potentially enhancing the overall performance of the cluster task, as shown in Fig. 13b.

5 Conclusion

In this study, we investigated the Google Trace dataset, which is priceless because, undoubtedly, many researchers are working on it, and the result can shape the next generation of cloud and data centers. We investigated the Google Trace from four perspectives to determine how to increase the cluster performance. In the first data analysis, we discovered that memory has a higher impact than CPU on performing tasks. We found that Google just upgraded the servers with the full CPU to have full memory, while the bottleneck is mostly memory, not CPU. Reducing 25% memory from 2218 machines with full CPU and memory capacity and applying the extra memory to 10,188 machines with 0.5 CPU capacity and 0.2493 memory can significantly improve the performance of Borg. In the second analysis, contrary to our expectation, we found that tasks with higher restrictions have less impact on rescheduling them; however, in the third section, we discovered that tasks with lower priority are the main reason

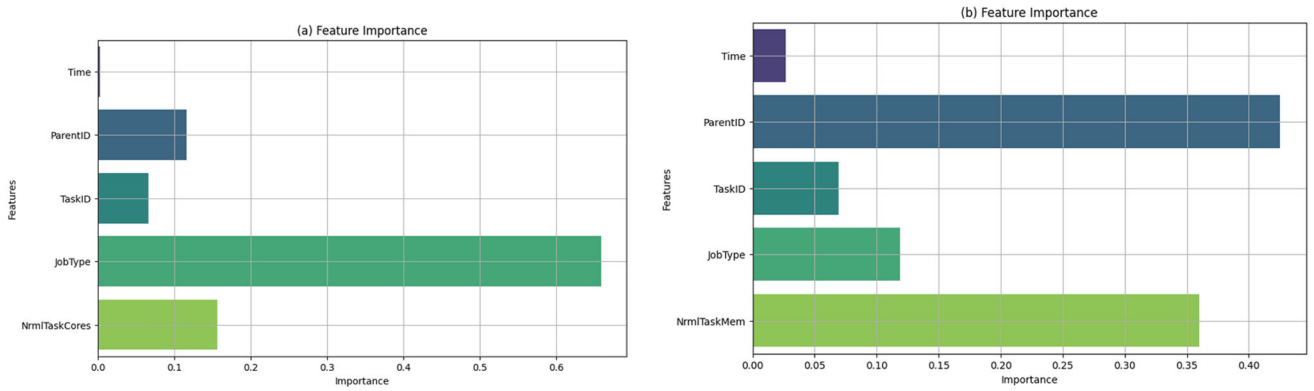


Fig. 13 Predictive model insights on task memory allocation (**a**) and CPU utilization (**b**) in Google Cluster Jobs Dataset

for rescheduling, and they increase the cloud overhead. We suggest using minimum time to use resources for each task, which can let tasks with lower priority have the opportunity to run entirely. Finally, we showed that tasks that are KILLed by the cloud management system have the highest number of constraints compared to those that are FINISHed, EVICTed, or FAILed. We suggest breaking the task into subtasks, decreasing the number of killed tasks, and increasing performance.

While the Google cluster dataset 2019 has been explored, it holds untapped insights. To unlock these, we propose leveraging AI and GPU power to predict task constraints, a major challenge in cloud environments. Accurately forecasting constraint numbers can lead to significant time and energy savings, as demonstrated by studies suggesting the effectiveness of combining machine learning algorithms like XGBoost with hyperparameter tuning techniques like swarm-based artificial bee colony optimization. Our work aims to push the boundaries of this dataset's potential, revealing valuable knowledge with real-world impact [32]. In addition, we suggest optimizing Borg system performance by exploring the possibility of disregarding virtual machines with exceptionally low resources, be it CPU or RAM. By addressing the potential bottleneck effect caused by such under-provisioned VMs, our future research aims to provide nuanced recommendations for resource allocation strategies within Borg. Furthermore, we recommend including datasets from other major cloud service providers like AWS, Microsoft Azure, and Alibaba in future research endeavors. Comparative analyses across these platforms could yield valuable insights for the broader research community.

Author contributions "D.S: Writing original draft, Methodology, Software, N.K: preparation, visualization; A.M.R.: conceptualization, validation, supervision, and investigation. All authors reviewed the manuscript."

Declarations

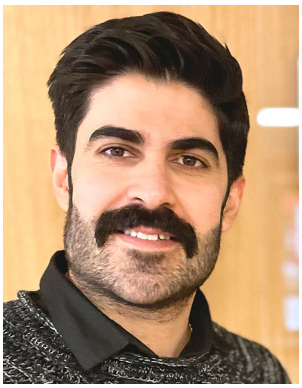
Competing interests The authors declare no competing interests.

References

1. Khaledian, N., Khamforoosh, K., Azizi, S., Maihami, V.: IKH-EFT: an improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment. Sustain. Comput.: Inform. Syst. **37**, 100834 (2023)
2. Rosà, A., Chen, L.Y., Birke, R., Binder, W.: Demystifying casualties of evictions in big data priority scheduling. ACM SIGMETRICS Perform. Eval. Rev. **42**(4), 12–21 (2015)
3. Chen, X., Lu, C. D., Pattabiraman, K.: Failure analysis of jobs in compute clouds: a Google cluster case study. In 2014 IEEE 25th International Symposium on Software Reliability Engineering (pp. 167–177). IEEE. (2014)
4. Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Wilkes, J.: Autopilot: workload autoscaling at Google. In proceedings of the fifteenth european conference on computer systems (pp. 1–16), (2020)
5. Anil, R., Capan, G., Drost-Fromm, I., Dunning, T., Friedman, E., Grant, T., Yılmazel, Ö.: Apache mahout: machine learning on distributed dataflow systems. J. Mach. Learn. Res. **21**(127), 1–6 (2020)
6. Gévay, G.E., Soto, J., Markl, V.: Handling iterations in distributed dataflow systems. ACM Comput. Surv. (CSUR) **54**(9), 1–38 (2021)
7. Tirmazi, M., Barker, A., Deng, N., Haque, M.E., Qin, Z.G., Hand, S., Wilkes, J.: Borg: the next generation. In proceedings of the fifteenth european conference on computer systems (pp. 1–14), (2020)
8. Fernández-Cerero, D., Varela-Vaca, Á.J., Fernández-Montes, A., Gómez-López, M.T., Alvárez-Bermejo, J.A.: Measuring data-centre workflows complexity through process mining: the Google cluster case. J. Supercomput. **76**, 2449–2478 (2020)
9. Gog, I., Schwarzkopf, M., Gleave, A., Watson, R.N., Hand, S.: Firmament: Fast, centralized cluster scheduling at scale. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 99–115), (2016)
10. Fernández Cerero, D., Fernández Montes González, A., Jakóbik, A., Kolodziej, J.: Stackelberg game-based models in energy-aware cloud scheduling. In ECMS 2018: 32nd European Conference on Modelling and Simulation (2018). European Council for Modelling and Simulation, (2018)
11. Khaledian, N., Khamforoosh, K., Akraminejad, R., Abualigah, L., Javaheri, D.: An energy-efficient and deadline-aware

workflow scheduling algorithm in the fog and cloud environment. Computing **106**(1), 109–137 (2024)
12. Fernández-Cerero, D., Jakóbik, A., Grzonka, D., Kołodziej, J., Fernández-Montes, A.: Security supportive energy-aware scheduling and energy policies for cloud environments. J. Parallel Distrib. Comput. **119**, 191–202 (2018)
13. Maala, H.H., Yousif, S.A.: Cluster trace analysis for performance enhancement in cloud computing environments. J. Theor. Appl. Inf. Technol. **97**(7), 2019 (2019)
14. R. Koch, "The 20/80Principle: the secret of achieving more with less.," *Doubleday,* (1999)
15. Van Loo, T., Jindal, A., Benedict, S., Chadha, M., Gerndt, M.: Scalable infrastructure for workload characterization of cluster traces. (2022), *arXiv preprint*
16. Adil, I.H., Wahid, A., Mantell, E.H.: Split sample skewness. Commun. Stat. Theory Methods **50**(22), 5171–5188 (2021)
17. Olabisi, D., Abubakar, S.K., Abdullahi, A.T.: demystifying dew computing: concept, architecture and research opportunities. Int. J. Comput. Trends Technol. **70**, 39–43 (2022)
18. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In Proceedings of the third ACM symposium on cloud computing (pp. 1–13), (2012)
19. Umer, A., Mian, A.N., Rana, O.: Predicting machine behavior from Google cluster workload traces. Concurr. Comput.: Pract. Exp. **35**(5), e7559 (2023)
20. Jassas, M. S., Mahmoud, Q. H.: Failure characterization and prediction of scheduling jobs in Google cluster traces. In 2019 IEEE 10th GCC Conference & Exhibition (GCC) (pp. 1–7). IEEE. (2019)
21. Wang, H., Jiang, C., Xie, B.: Missing data analysis and prediction: a Google cluster case study. (2022)
22. Ngang'a, D.N., Cheruiyot, W.K., Njagi, D. A Machine Learning Framework for Predicting Failures in Cloud Data Centers-A Case of Google Cluster-Azure Clouds and Alibaba Clouds. Available at SSRN 4404569
23. Soualhia, M., Khomh, F., Tahar, S.: Predicting scheduling failures in the cloud: A case study with Google clusters and Hadoop on Amazon EMR. In 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems (pp. 58–65). IEEE. (2015)
24. Chen, S., Yang, C., Huang, W., Liang, W., Ke, N., Souri, A., Li, K.C.: Fairness constraint efficiency optimization for multiresource allocation in a cluster system serving internet of things. Int. J. Commun. Syst. **36**(3), e5395 (2023)
25. Wilkes, J.: More Google cluster data. *Google research blog, Nov*, (2011)
26. Gupta, S., Dileep, A.D.: Long range dependence in cloud servers: a statistical analysis based on Google workload trace. Computing **102**(4), 1031–1049 (2020)
27. Subramanian, N.V., Sriram, V.S.: Load-aware VM migration using hypergraph based CDB-LSTM. Intell. Autom. Soft Comput. **35**(3), 3279–3294 (2023)
28. Berisha, B., Mëziu, E., Shabani, I.: Big data analytics in Cloud computing: an overview. J. Cloud Comput. **11**(1), 24 (2022)
29. Osborne, J.W., Overbay, A.: The power of outliers (and why researchers should always check for them). Pract. Assess. Res. Eval. **9**(1), 6 (2019)
30. Seo, S.: A review and comparison of methods for detecting outliers in univariate data sets (Doctoral dissertation, University of Pittsburgh), (2006)
31. Brys, G., Hubert, M., Struyf, A.: A robust measure of skewness. J. Comput. Graph. Stat. **13**(4), 996–1017 (2004)
32. Tawhid, A., Teotia, T., Elmiligi, H.: Machine Learning for Optimizing Healthcare Resources Machine Learning, Big Data, and IoT for Medical Informatics, pp. 215–239. Academic Press, Cambridge (2021)

Danyal Shahmirzadi is a computer scientist with academic foundations extending over two decades. He earned his Bachelor of Science in Computer Science in 2008 and subsequently completed a Master's degree in Software Engineering in 2010. Demonstrating his ongoing commitment to his field, he is presently a Ph.D. candidate in Computer Science and Information Engineering at the National Yunlin University of Science and Technology (Yuntech) in Taiwan.



Navid Khaledian received his Ph.D. in Artificial Intelligence from the Islamic Azad University, Sanandaj branch, Iran, in 2023. He is currently a postdoctoral researcher at the University of Luxembourg. His research interests include Artificial Intelligence (AI), Distributed Systems, and Internet of Things (IoT), focusing on task scheduling, recommender systems, and data mining.

Amir Masoud Rahmani received his BS in computer engineering from Amir Kabir University, Tehran, in 1996, his MSc in computer engineering from Sharif University of Technology, Tehran, in 1998, and his PhD in computer engineering in 2005. Currently, he is a Professor in the Department of Computer Engineering. His research interests are the Internet of Things, cloud/fog computing, and machine learning.