Baghdad Science Journal

Volume 23 | Issue 4 Article 17

4-23-2026

Optimizing Quality of Service in Cloud Computing: A Performance-Aware Task Classifier for Heterogeneous Big Data Workloads

Pawan Bhaker
Department of Computer Applications, Lovely Professional University, Phagwara, Punjab, India, pawanbhaker88@gmail.com

Sophiya Sheikh
Department of Computer Applications, Lovely Professional University, Phagwara, Punjab, India, sophiya.sheikh@gmail.com

Rintu Nath
Department of Science and Technology, New Delhi, India, rnath07@gmail.com

Ajay Nain
Department of Computer Applications, Lovely Professional University, Phagwara, Punjab, India, mr.ajaynain@gmail.com

How to Cite this Article

Bhaker, Pawan; Sheikh, Sophiya; Nath, Rintu; and Nain, Ajay (2026) "Optimizing Quality of Service in Cloud Computing: A Performance-Aware Task Classifier for Heterogeneous Big Data Workloads," *Baghdad Science Journal*: Vol. 23: Iss. 4, Article 17.
DOI: https://doi.org/10.21123/2411-7986.5273

BAGHDAD SCIENCE JOURNAL 2026;23(4):1363–1377

Scan the QR to view the full-text article on the journal website

RESEARCH ARTICLE

Optimizing Quality of Service in Cloud Computing: A Performance-Aware Task Classifier for Heterogeneous Big Data Workloads

Pawan Bhaker [1], **Sophiya Sheikh** [1,*], **Rintu Nath** [2], **Ajay Nain** [1]

[1] Department of Computer Applications, Lovely Professional University, Phagwara, Punjab, India
[2] Department of Science and Technology, New Delhi, India

ABSTRACT

The Big data originates from various heterogeneous sources worldwide. Due to the diverse origins of data generation, big data involves a variety of tasks. Some require storage resources, while others need computational resources. Additionally, cloud computing provides centralized storage and computational resources for executing these tasks. However, resource optimization and efficient task scheduling remain challenging. Moreover, accurately categorizing tasks and effectively allocating resources based on their nature pose challenges for cloud computing, especially when assigning tasks to heterogeneous resources to minimize task completion time. To address these issues, this paper proposes a Performance-Aware Task Classifier (PATC) algorithm that classifies tasks and allocates resources accordingly while optimizing QoS parameters. The task classification algorithm divides tasks into two categories: data-intensive and compute-intensive. Threshold coefficients for transmission time and CPU usage are used for task classification. Furthermore, resources are allocated to compute-intensive tasks to minimize their execution time as much as possible. To evaluate system performance, various QoS parameters such as Average Response Time, Total Completion Time, Makespan, and Average Resource Utilization are analyzed. Simulation results show that our approach effectively improves these parameters compared to existing methods such as First Come First Serve (FCFS), Shortest Job First (SJF), Modified SJF, Task Scheduling Strategy (TSS), and IMFO (Improved Moth Flame Optimization).

Keywords: Big data, Cloud computing, Intensive tasks, QoS, Resource optimization, Task categorization

Introduction

Big data can manage enormous and complex datasets that are typically difficult to process, manage, and analyze using traditional data processing tools and methods, such as RDBMS and SQL.[1] It is a valuable resource for improving businesses and organizations. Many companies and organizations utilize big data to enhance their performance. Big data is applied in various sectors, including healthcare, education, aerospace, military, e-commerce, and other fields. Social media applications like Facebook, Instagram, Twitter, and LinkedIn generate enormous amounts of data daily, containing a lot of information, and their insights are extracted by tools like Hadoop.[2] In 2024, approximately 147 zettabytes of data were generated, and by 2025, an estimated 181 zettabytes of data are expected to be produced.[3] Big data is heterogeneous because it originates from diverse sources, resulting in different tasks that require various resources for execution. Some tasks demand computational resources, while others need storage or data processing resources. Based on these requirements, tasks are categorized into distinct types: data-intensive tasks, I/O-intensive tasks, compute-intensive tasks, memory-intensive tasks, etc.

Received 6 August 2025; revised 27 October 2025; accepted 31 October 2025.
Available online 23 April 2026

* Corresponding author.
E-mail addresses: pawanbhaker88@gmail.com (P. Bhaker), sophiya.sheikh@gmail.com (S. Sheikh), rnath07@gmail.com (R. Nath), mr.ajaynain@gmail.com (A. Nain).

https://doi.org/10.21123/2411-7986.5273

Data-intensive tasks require significant data transfer and storage capacity, whereas compute-intensive tasks demand high computational power. Many applications and organizations manage both types of tasks, requiring their categorization and the allocation of resources accordingly to enhance system performance.

As data-intensive tasks involve processing large volumes of data efficiently, they focus on managing massive datasets that require high storage capacity, rapid data retrieval, and optimized data movement. These tasks include data mining, large-scale analytics, and real-time data streaming, all of which require robust storage options and high-speed data transfer mechanisms. Effectively storing and managing such extensive datasets necessitates the use of distributed file systems, such as Hadoop Distributed File System (HDFS) and Amazon S3, which enable data to be stored across multiple nodes to ensure redundancy and fault tolerance.[4] However, bandwidth management remains a significant challenge, as transferring large volumes of data between storage locations and processing nodes can cause network congestion and impair overall performance. To tackle this issue, many researchers have employed various optimization techniques, including machine learning-based predictive models, heuristic approaches, and data compression algorithms, to enhance bandwidth efficiency and minimize latency.

Conversely, compute-intensive tasks require substantial processing power, often utilizing High Performance Computing (HPC) resources such as CPUs and GPUs.[5] These tasks include scientific simulations, training deep learning models, cryptographic calculations, and complex mathematical modelling. Compute-intensive tasks need optimized execution strategies to make efficient use of available computational resources. Researchers frequently use machine learning techniques, parallel computing frameworks, and optimized algorithms to achieve faster and more accurate computations. Unlike data-intensive tasks that mainly concentrate on storage and data transfer, compute-intensive tasks rely on the efficiency of processing units and the distribution of workloads across multiple computing nodes.

Many applications and organizations handle data-intensive and compute-intensive tasks, making it essential to classify these tasks and allocate resources appropriately.[6] Adequate task classification enables systems to optimize resource distribution, ensuring that compute-heavy processes receive sufficient processing power while data-heavy operations have access to adequate storage and bandwidth. Through effective task categorization, system performance can be significantly enhanced, resulting in reduced execution times, lower costs, and improved scalability. Despite its significance, big data alone cannot provide the necessary computational and storage resources for these compute- and data-intensive tasks. This is where cloud computing comes into play.

Cloud computing offers scalable and on-demand computational and storage resources through private, public, and hybrid cloud models. Microsoft Azure, Google Cloud, and Amazon Web Services (AWS) are examples of public cloud providers.[7] Cloud computing offers low-cost resources that enhance the performance of the IT sector. Cloud distributes different resources in a cost-effective and flexible way that improves the yield of different organizations and the IT sector.[8] Cloud computing supplies high-performance computing instances and distributed storage solutions that meet the resource requirements of both compute and data-intensive applications. It is essential to provide resources efficiently and optimally to both types to enhance system performance. Therefore, there is a need to develop a technique that manages resources effectively and allocates them in a way that improves QoS (Quality of Service) in a cloud environment. Moreover, there is a need to create a system that can dynamically classify tasks based on their attributes and implement intelligent scheduling approaches that consider CPU demand and bandwidth usage.[9] The primary goal is to ensure that the most suitable resources are allocated to each task, thereby improving system utilization and reducing execution time. For optimizing data-intensive applications, parallelization techniques and high-performance computing methods are used to enhance data and compute efficiency.[10]

Based on the existing literature, we found that there is no adequate task categorization, and resource optimization is conducted based on the task's resource requirements. To overcome this, we introduced a technique called Performance-Aware Task Classifier (PATC). The algorithm is divided into two parts: the first one is task categorization, and the second is resource optimization. After generating the task by the user, the PACT algorithm segregates the tasks based on their resource needs. For this purpose, we use threshold coefficients for transmission time and CPU usage. Initially, we calculate the transmission time and evaluate it with the considered transmission time threshold. After this, we calculate the CPU usage and assess it against the CPU usage threshold. Based on this evaluation, the tasks that require more storage capacity are considered data-intensive tasks, while those needing high computational capacity are categorized as compute-intensive tasks.

Data-intensive tasks are stored, while compute-intensive tasks are executed on computational servers. To execute compute-intensive tasks, we provide servers in a manner that minimizes the task running time as much as possible. Here, we optimize various QoS parameters, such as Average Response Time, Total Completion Time, Makespan, and Average Resource Utilization. The proposed PATC algorithm outperforms several techniques, including FCFS (First Come First Serve),[11] SJF (Shortest Job First),[12] Modified SJF,[12] TSS (Task Scheduling Strategy),[13] and IMFO (Improved Moth Flame Optimization),[14] across these QoS metrics. Our specific contribution to this research work is as follows:

- A Performance-Aware Task Classifier algorithm specifically designed for cloud environments to efficiently handle and manage big data.
- In a Big Data-based cloud environment, tasks are assigned through a strategic approach to make the best use of resources, speed up completion, and ensure quick responses, improving the system's overall performance.
- A four-layer big data-based cloud architecture is developed to support the implementation of the task categorization and resource allocation algorithm.
- Task segregation enables the efficient management of diverse tasks by optimally allocating resources based on their specific needs and requirements. Here, data-intensive and compute-intensive tasks are isolated.
- After task segregation, resources are allocated to these tasks. The compute-intensive tasks are scheduled to ensure they are assigned to the most suitable server, minimizing task completion time as much as possible. For data-intensive tasks, resources are allocated based on their storage requirements.
- We compare our method with existing ones and find that it optimizes task scheduling and various QoS parameters.

A comparison of the proposed PATC technique with existing technology is illustrated in Table 1.

Related work

Different categories of data and tasks are generated from diverse sources in big data. The tasks may be data-intensive, computing-intensive, I/O-intensive, and many others. Several researchers have conducted extensive studies on data-intensive and compute-intensive applications, but these studies have been conducted separately. Some of these researchers have also optimized data-intensive and compute-intensive tasks on various computing platforms. However, very few have optimized them together. A few recent optimizations related to data-intensive and compute-intensive tasks are discussed below:

Optimization of data-intensive applications in big data

Data-intensive applications involve large datasets that require extensive storage resources and a robust data transfer system. Many researchers have conducted studies to optimize storage solutions and improve bandwidth systems for running data-intensive applications, aiming to boost overall system performance. This section reviews several research efforts focused on task scheduling and resource provisioning for such applications. In,[15] the authors introduced a novel Cuckoo Search-based Optimized Resource Allocation (CSO-AR) technique, which achieved optimal response times and effectively allocated resources for data-intensive applications. The CSO-RA technique provided an optimized solution for data-heavy tasks in a cloud computing environment. Large data size in data-intensive applications increases the communication cost and uses more energy. Therefore, authors[16] proposed a mixed-integer linear programming model that considered the data size and the location of the data. Further optimized communication cost and energy consumption in mobile edge cloud computing environments. Additionally, to reduce the completion time of the reducer task in MapReduce, the authors introduced a Balanced Data Clusters Partitioner (BDCP) algorithm.[17] In BDCP, a supplementary sampling phase is integrated into MapReduce to analyze data distribution and offer

Table 1. Comparison of the proposed work with existing technology.

	Proposed (PATC)	TSS	M. SJF	SJF	FCFS	IMFO
Task Categorization	√	√	×	×	×	×
Task Scheduling	√	√	√	√	√	√
Makespan	√	×	√	√	×	√
Response Time	√	√	×	×	√	×
Completion Time	√	×	√	√	×	×
Average Resource Utilization	√	×	√	×	×	√

feedback on processing capacity. Based on these insights, they optimized the completion time. Furthermore, a two-phase Virtual Machine Placement (VMP) strategy is proposed to improve host utilization and decrease network traffic within cloud data centers.[18] This method uses a fuzzy inference system and linear programming for rack selection, followed by a greedy algorithm for VM placement. The study also emphasizes the NP-hard nature of this problem, highlighting its complexity. Additionally, in,[19] the authors proposed a shared, data-aware resource provisioning and task scheduling method designed to optimize data-intensive applications on hybrid clouds. They addressed high data transfer costs and network latency by employing Aneka, a cloud platform, to facilitate efficient resource allocation. Their approach reduced data transfer times by co-locating tasks that shared large files, significantly improving response times. Also, a new approach called Cooperative Job Scheduling and Data Allocation (CSA) for managing data-intensive parallel computing clusters was introduced in.[20] Unlike traditional methods that allocate data before scheduling jobs, CSA prioritizes job scheduling first. This strategy improves data locality, decreases network load, and conserves energy. By integrating heuristic algorithms and recursive optimization, CSA enhanced task assignment and data placement. Finally,[21] examined a deep reinforcement learning-based method for scheduling data-intensive workflows across multi-cloud environments. It addressed issues such as data flow control and transmission scheduling by segmenting workflows and optimizing execution constraints. Using an enhanced Deep Q-Network (DQN) algorithm, this approach significantly reduces execution times and enhances load balancing.

Optimization of compute-intensive applications in big data

Compute-intensive applications require high computational capacity to handle complex calculations. There is a need for a technique or strategy to schedule tasks to resources so that tasks are executed more efficiently, thereby boosting system performance. Many researchers focus on developing effective methods for implementing computationally intensive applications. Key research findings are highlighted here. In,[22] the authors introduced a task offloading technique based on Federated Deep Q-Learning (FedDQL) for latency-sensitive and computationally intensive Vehicular Fog Computing (VFC) applications. This approach optimized task placement by considering network latency, energy consumption, and computational efficiency. By integrating federated learning, the method enhanced privacy preservation and resource utilization while reducing system overhead. In,[23] the authors proposed an Actor-Critic-based Compute-Intensive Workload Allocation Scheme (AC-CIWAS) to improve energy efficiency in data centers. Using Deep Reinforcement Learning (DRL), the model dynamically adjusted workload placement while maintaining Quality of Service (QoS). Conversely,[24] discussed MCFuser, a framework designed to optimize Memory-bound Compute-Intensive (MBCI) operators by improving operator fusion in deep learning models. It utilized high-level tiling expressions, DAG analysis, and heuristic search to lower memory access overhead and tuning time. Additionally,[25] presented a Workflow Scheduler for Hadoop (WSH) to improve task scheduling in heterogeneous computing environments. This approach addressed resource heterogeneity by classifying tasks as compute-intensive or I/O-intensive and using training tasks to enhance job allocation. Moreover, a task scheduling scheme for computation-intensive graph jobs in UAV-assisted hybrid vehicular networks is described in.[26] By integrating traditional vehicles, autonomous vehicles, and UAVs, the authors aimed to minimize average job completion delays. The two-stage scheme optimized job scheduling order and task offloading decisions, effectively reducing average completion times compared to existing methods. Furthermore, in,[27] the authors explored how GPU acceleration could enhance the MapReduce framework using Hadoop and OpenCL, specifically for breast cancer image analysis. By integrating GPUs, the system significantly increased processing speeds for large-scale image data. The implementation achieved nearly 13 times better performance without additional optimizations. Meanwhile,[28] investigated methods to make computationally intensive Simulink models run more efficiently by integrating library functions generated with Halide. It used a templated approach to extract parameters from Simulink models and generate optimized Halide code, resulting in a performance boost of up to 80 times.

Optimization for both data and compute-intensive applications

The paper[29] examines how HPC systems can improve the performance of data- and compute-intensive tasks by leveraging big data tools such as Hadoop and Spark. It encompasses system-level improvements, including enhanced scheduling and data management, to improve efficiency and scalability. To decrease response times, the author of the paper[30] suggests a cloud task scheduling method that categorizes tasks into four types: compute-intensive,

data-intensive, hybrid, and low-intensity. Each task is assigned to the most appropriate resource once resources are appropriately prioritized and allocated. In simulations, this method outperforms traditional scheduling techniques in response time and QoS delivery. Meanwhile, to achieve low latency and high throughput, the author in the paper[31] presents an Improved Scheduling Algorithm (I-PDWA) for stream-based, data-intensive workflows. Breaking down workflows, minimizing inter-partition data movement, and applying data parallelism for compute-heavy tasks help reduce latency and increase throughput. Evaluations on both synthetic and real-world workflows demonstrate significant performance improvements compared to existing methods. To optimize memory usage, a dynamic memory controller is proposed in paper[32] that balances memory allocation between compute-intensive HPC applications and data-intensive frameworks such as Spark. DynIMS achieves a fivefold speed increase over static memory allocation strategies by adaptively managing in-memory storage in real time through feedback control.

Motivation and research gap

Big data handles a variety of tasks, and effective resource allocation is essential for optimizing system performance and improving user experience. After reviewing the literature, we found that most researchers focus on tasks within a single category, such as compute-intensive, data-intensive, or I/O-intensive tasks. Very few studies consider multiple task categories, and some limitations have been identified among them. TSS was proposed to classify data-intensive and compute-intensive tasks by first dividing resources based on capacity. Resources with high processing power and low storage and bandwidth are considered CIP (Compute-Intensive Partition). In contrast, those with high storage and bandwidth but low processing power are classified as DIP (Data-Intensive Partition). Resources that possess both capabilities are called HIP (Hybrid-Intensive Partition), and those with limited processing, storage, and bandwidth are labelled LIP (Low-Intensive Partition). The system contains information about tasks, matches task requirements to the resource pool, and categorizes tasks based on their resource needs. However, it only isolates tasks without accounting for their specific resource requirements. The researchers categorize tasks but do not efficiently allocate resources, leading to minimized resource utilization and longer task completion times. In IMFO, the author categorizes VM using SVM into four categories: unstable VM with high resource utilization, moderately stable with moderate resource utilization, and stable with high and low resource utilization. However, this approach is more resource-focused and does not consider the task properties. Therefore, sometimes the tasks may suffer from starvation. Moreover, this approach concentrates on load balancing rather than executing the task in the least possible way. FCFS was proposed to allocate resources on a first-come, first-served basis. In FCFS, the earliest task is scheduled on the server, regardless of its priority or deadline. While this can reduce response time, it may negatively affect other optimization parameters. SJF was suggested to allocate resources to the task with the shortest execution time, but if short-duration tasks repeatedly arrive, longer tasks may face difficulties. Consequently, it struggles with dynamic tasks, which can lead to starvation. To address this, a modified SJF was proposed to manage the dynamic nature of tasks. It dynamically adjusts task priority, considering task size and resource availability, to allocate resources effectively. However, these approaches do not consider task categorization or allocate resources based on specific needs. Therefore, there is a critical need for an effective scheme that can not only organize tasks according to their nature but also meet their particular resource requirements in an optimized manner. This was the primary motivation for introducing the proposed PATC algorithm.

Proposed methodology

A cloud-based architecture is designed to allocate resources to different types of intensive tasks efficiently. Furthermore, this section covers the system architecture, problem formulation, different QoS metrics, and algorithms.

System architecture

A system architecture is introduced that optimizes and supplies resources to big data tasks based on their requirements. Here, we present a four-layer architecture, as shown in Fig. 1, which categorizes tasks and allocates resources according to their needs. Each layer of the architecture is briefly described below.

- **Application Layer:** At this layer, many users submit tasks through various applications. This layer provides an interface through which users can request tasks and receive responses from the server. These tasks vary, so they require different types of handling. Some tasks need computational power, while others require storage capacity for their execution. Therefore, it is essential to classify them

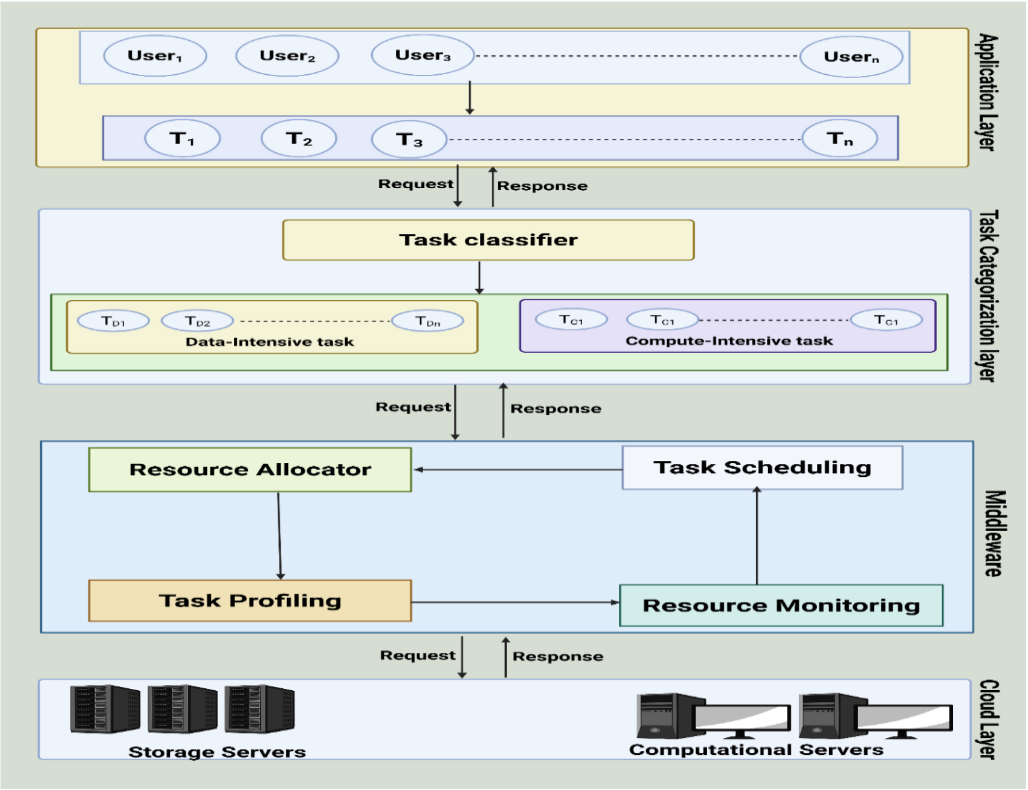

Fig. 1. The proposed system architecture.

into their respective categories to improve system performance.

- **Task Categorization Layer:** This layer uses a task classifier to categorize tasks based on their needs. Tasks can be data-intensive or compute-intensive, requiring resources suited to their characteristics for efficient performance. The task classifier sorts tasks into two groups: data-intensive and compute-intensive. After categorizing these tasks, they are sent to the middleware layer for better and optimized resource allocation.
- **Middleware:** This layer uses scheduling techniques for compute-heavy tasks, optimizing QoS parameters by efficiently assigning resources. At this level, some components include task profiling, resource monitoring, task scheduling, and resource allocation.
  - ➢ **Task profiling:** Task profiling offers details about the task, such as task size, task type, and CPU usage, aiding in effective task scheduling.
  - ➢ **Resource monitoring:** It determines whether a resource is optimal, overloaded, or underloaded. It tracks resources in real-time and ensures none are overused or underused, improving resource management.
  - ➢ **Task scheduling:** This assists in assigning tasks to available resources. It aims to schedule the task to the most suitable resource, thereby improving the system's performance.
  - ➢ **Resource allocator:** The resource allocator assigns resources to tasks based on their needs. It determines where and how to allocate resources to the task.
- **Cloud Layer:** At the cloud layer, resources are assigned to tasks after being properly categorized through a scheduling method. The cloud layer includes storage and computational resources, which are then allocated to functions based on their features to ensure successful completion.

Problem formulation

In this technique, we consider a set of independent tasks, denoted by $T = \{t_i : 1 \leq I \leq N\}$ which are sent for execution on a set of resources $R = \{r_k : 1 \leq k \leq K\}$. Resources having processing power/speed $P[r_k]$ in MIPS (Million Instructions per Second). K resources are considered for executing computational tasks, while we utilised a dedicated resource, represented by r'_k, with minimal processing power, to process data-intensive tasks. A few of the considered and essential resource characteristics are discussed as follows:

- K resources are considered for processing the task with an efficient resource allocation strategy.
- Resources are heterogeneous and can process both data-intensive and compute-intensive tasks.
- A dedicated server r'_k with the lowest processing capacity is considered for executing data-intensive tasks, and the remaining servers are computational.
- The processing speed/capacity $P[r_k]$ of resources, r_k is taken in MIPS.

After the task arrives in the system, we allocate it to the resources for execution. However, we need to map the task to the appropriate resource so that task execution time can be minimized. The mathematical formulation for task-to-resource mapping can be considered as follows:

$$M : T \rightarrow R$$

Some of the specifications for the tasks considered are as follows:

- Every task has a different size $S(t_i)$ is considered in MB.
- Based on task size and bandwidth, we calculate transmission time, TT. If $TT \geq \alpha_t$, we send the task to a dedicated server, r'_k and calculate the CPU usage, CU. For $CU \leq \alpha_c$, we consider them data-intensive.
- α_t and α_c are the threshold coefficients for transmission time and CPU usage, respectively.

Tasks are sent to the cloud servers for execution through the network bandwidth. The tasks are generated from various heterogeneous sources and have different processing demands, such as storage and computation. The tasks need to be categorized according to their nature so that appropriate resources can be allocated to them for further execution. For the task segregation, the transmission time $TT(t_i)$ is calculated by Eq. (1).

$$TT\ (t_i) = \frac{S\ (t_i)}{BW} \tag{1}$$

Here, $S(t_i)$ represents the task size, and BW stands for bandwidth.

If the transmission time is equal to or above the α_t, then the task will be sent to the dedicated server. Afterwards, CPU usage $CU(t_i)$ will be calculated. To find the same, we need to find the execution time from Eq. (2), for each task on the dedicated server as follows:

$$ET\ (t_i) = \frac{S\ (t_i)}{P\left[r'_k\right]} \tag{2}$$

Where $P[r'_k]$ The processing capacity of the dedicated server.

$$CU\ (t_i) = \frac{ET\ (t_i)}{P\left[r'_k\right]} \times 100 \tag{3}$$

After calculating the $CU(t_i)$, from Eq. (3) we compare it to the α_c. If $CU(t_i)$ is less than or equal to α_c, this will be considered as a data-intensive task, i.e., $T(t_{di})$ and send them to the dedicated server for their execution. The remaining tasks from the T are considered as compute-intensive tasks, i.e., $T(t_{ci})$ as calculated using Eq. (4).

$$T\ (t_{ci}) = T - T\ (t_{di}) \tag{4}$$

The execution time of $T(t_{ci})$ is calculated as follows in Eq. (5).

$$ET\left(t_{ci,r_k}\right) = \frac{S\ (t_{ci})}{P\ [r_k]} \tag{5}$$

$S(t_{ci})$ is the size of compute-intensive tasks, and $P[r_k]$ is the processing capacity of computational servers. Afterwards, we calculate the Completion Time (CT) for all compute intensive tasks t_{ci} on each server r_k as mentioned in Eq. (6).

$$CT\left(t_{ci,r_k}\right) = EL\ [r_k] + ET\left(t_{ci,r_k}\right) \tag{6}$$

It is the sum of the Existing Load (EL) also called Ready Time, and the sum of the execution time of the task executed on a particular server r_k. The $T(t_{ci})$ is further sent to the computational server for execution. The $T(t_{ci})$ are scheduled to the best possible server that can execute them in a way that minimizes the server's running time as much as possible, as given in Eq. (6).

Here, we optimise QoS parameters like the makespan, total completion time, average response time, and average resource utilization for the $T(t_{ci})$ as discussed below:

Response Time: It is the time a server takes to respond to a task initially. It also refers to the period a task waits before being assigned to the server. The mathematical expression for response time is computed in Eq. (7) as follows.

$$RT\ (t_{ci}) = ST\ (t_{ci}) - AT\ (t_{ci}) \tag{7}$$

Where $RT(t_{ci})$ represents the response time, $ST(t_{ci})$ stands for the start time of a task or when the users submit the tasks for execution. While $AT(t_{ci})$ represents the arrival time of a task or the time at which tasks arrive in the system for execution. Initially, the arrival time of each task is 0.

Average Response Time: The Average Response Time ($Avg\ RT$) of a task is the sum of the response times of all tasks divided by the total number of tasks N. The same is calculated in Eq. (8).

$$Avg\ RT\ (t_{ci}) = \sum RT\ (t_{ci})\ /N \tag{8}$$

Total Completion Time: It is the sum of the completion times (CT) of all tasks executed on a particular server r_k. The mathematical expression of total completion time (TCT) is computed in Eq. (9) given below.

$$TCT\left[r_k\right] = \sum CT\left[r_k\right] \tag{9}$$

Makespan: It is the completion time of the last executed task or the maximum running time of a server among all servers. The makespan (MS) can be calculated in Eq. (10).

$$MS = \max\left(CT\ [r_k]\right) \tag{10}$$

Resource Utilization: The resource utilization RU_k is calculated in Eq. (11) as mentioned below.

$$RU_k = \frac{CT\ [r_k]}{MS} \tag{11}$$

However, the average resource utilization ARU is calculated in Eq. (12).

$$ARU = \frac{\sum (CT\ [r_k])}{MS \times k} \tag{12}$$

ARU represents the Average Resource Utilization, and k represents the number of resources.

Proposed algorithm

The proposed task categorization and resource optimization technique first classifies tasks into compute-intensive and data-intensive categories, then allocates resources efficiently to compute-intensive functions, allowing the system to perform more effectively. In our algorithm, we first categorize both tasks and then allocate resources. Both algorithms are described below.

Task Categorization Slgorithm

This algorithm categorizes tasks into two types: data-intensive tasks and compute-intensive tasks. Data-intensive tasks involve large amounts of storage, facing challenges such as transferring data over a network and storing it efficiently. In contrast, compute-intensive tasks require substantial computational resources. Based on these properties, we classify both types of tasks. The steps for categorizing the tasks are discussed below, explaining how these tasks are differentiated.

- The input parameter that we have taken into consideration is the number of tasks T, server R, bandwidth BW, and processing speed $P[r_k]$, and task size $S(t_i)$.
- The tasks are transmitted through a bandwidth. We calculate the transmission time $TT(t_i)$ of each task using Eq. (1). The transmission time for each task is then compared with the transmission threshold coefficient; if the task's transmission time is less than or equal to this threshold, we send that task to the designated server, r'_k.
- Then, we calculate the CPU usage percentage of these tasks on a dedicated server. First, we find each task's execution time using Eq. (2). After that, we determine the CPU usage percentage with Eq. (3).
- Then, we compare the CPU usage percentage with the capacity threshold coefficient. If the CPU usage percentage exceeds the threshold coefficient, the task is considered data-intensive, while the remaining tasks are regarded as compute-intensive.
- In the end, we receive the output and classify tasks as data-intensive and compute-intensive. We then send data-intensive tasks to the dedicated server. The formal algorithm is presented in Algorithm 1.

Task Scheduling Algorithm

After categorizing the task, the critical aspect is to allocate resources efficiently and optimize QoS parameters. In this algorithm, we assign the task to the most suitable server. The steps are discussed below.

- For the data-intensive tasks, the allocation will be done to the dedicated server.
- For the compute-intensive tasks, first, we estimate the execution time for all tasks and calculate each task's completion time by adding the current load to the estimated execution time.
- Then, we allocate the server that takes the minimum running time for that task.
- Furthermore, we calculate the average response time according to Eq. (8).
- Based on the completion time, we determine the makespan and average resource utilization using the respective formulas.
- In this way, the optimal server is allocated to each task, aiming to enhance QoS parameters such as total completion time, response time, average resource utilization, and makespan. The formal algorithm is presented in Algorithm 2.

Algorithm 1

Input parameter: (T, R, $S(t_i)$, BW, $P[r_k]$) //Variables of task and resources
Output: ($T(t_{ci})$, $T(t_{di})$ //Task categorization
for each task t_i
  do
  Compute $TT(t_i)$ from Eq. (1)
    if ($TT(t_i) \geq \alpha_t$)
      Send those tasks to the dedicated server r'_k
      Compute $CU(t_i)$ by Eq. (3)
    end if
    if ($CU(t_i) \leq \alpha_c$)
      Add t_i as data-intensive tasks
      else
      Add t_i compute-intensive tasks
    end if
end for

Algorithm 2:

Input parameter: $T(t_{ci})$, $T(t_{di})$,
Output parameter: $RT(t_{ci})$, $Avg\ RT(t_{ci})$, $TCT[r_k]$, MS, ARU
for each data-intensive task $T(t_di)$
    Allocate r'_k to $T(t_{di})$
end for
for each compute-intensive task $T(t_{ci})$
for each server r_k
    do
    Compute the Execution Time $ET(t_{ci,\ r_k})$ by Eq. (5)
    Compute the Completion Time $CT[r_k]$ by Eq. (6)
    Find the appropriate server r_k with minimum $CT[r_k]$
    Schedule $T(t_{ci})$ accordingly
    Assign the task to the selected server for execution
end for
end for
  for R:= 0 to K
    Calculate $Avg\ RT(t_{ci})$, $TCT[r_k]$, MS, ARU, from Eqs. (5) to (12), respectively.
end for

Experimental study

This section covers the experimental study, offering detailed information about the system configuration language used for the program. Additionally, we describe the performance evaluation and then discuss the results obtained after executing the algorithm.

Experiment setup

For the evaluation of our proposed work, we used an MSI (2024 model) with an AMD Ryzen 5 7530U processor, Radeon Graphics, 2000 MHz, 6 Cores, 12 logical processors, 16 GB RAM, and a 512 GB SSD storage system. We simulated a cloud environment by randomly generating tasks and allocating servers to execute them. To assess the performance of the proposed work, PATC, we measured relevant QoS parameters. These parameters were evaluated on 100 to 500 tasks, considering five servers with processing capacities ranging from 10 to 50 MIPS. Task sizes varied between 1 and 10 MB. MB. The heterogeneity levels of task size and server capacity are motivated by the ETC Simulation Benchmark.[33]

Performance evaluation

Our proposed algorithm efficiently classifies tasks based on their resource needs. Here, we optimized the average response time, total completion time,

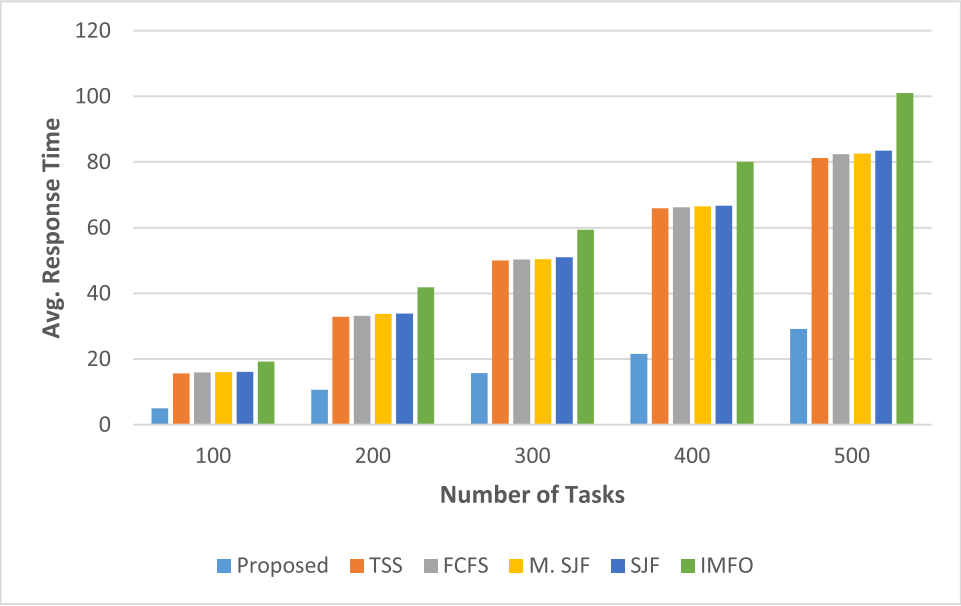

Fig. 2. Comparison of average response time.

makespan, and average resource utilization. PTAC improves these metrics more effectively than FCFS, SJF, Modified SJF, IMFO, and TSS.

In FCFS,[11] tasks that arrive first are scheduled for execution. That's why its average response time is better compared to SJF and Modified SJF. On the other hand, if a large task arrives first, it can delay the execution of smaller or higher-priority tasks. Therefore, there is a significant increase in total completion time, makespan, and average resource utilization.

In contrast, SJF[12] prioritises jobs with the shortest execution times. Unlike FCFS, it initially allocates resources to tasks that require minimal execution time. SJF executes smaller tasks first but does not consider computational demand, which can lead to task starvation for larger tasks. Additionally, it is a non-preemptive approach and cannot handle the dynamic arrival of tasks. Therefore, it only performs better than FCFS in terms of total completion time, makespan, and average resource utilization.

Modified SJF takes into account the changing nature of tasks and resources, considering both task duration and resource availability. It assigns priorities based on these factors to allocate resources effectively.[12] Although Modified SJF includes some solutions for dynamic load, it still falls short in performance. Modified SJF outperforms FCFS and SJF across all evaluated parameters because it not only schedules tasks with the shortest execution time but also considers task size and resource availability.

On the other hand, TSS[13] is more resource-oriented, which slightly enhances the values of all considered QoS parameters compared to other heuristics. The TSS classifies tasks by matching their requirements with the resource pool. TSS also considers memory, processing power, and bandwidth when allocating resources to tasks.

The improved IMFO[14] schedule task is based on virtual machine properties and does not consider the task requirements. Since big data tasks are heterogeneous in nature, ignoring task properties leads to resource mismatch and inefficient performance. Additionally, this strategy, which allocates resources/VMs based on their functionalities, does not effectively optimize major scheduling parameters.

Our proposed method initially separates tasks based on CPU usage and transmission time and then assigns resources according to their needs. As a result, it outperforms other approaches and optimises various QoS parameters.

Results and discussions

The algorithms used for comparison are implemented and tested within a simulation environment that varies the heterogeneity levels of the considered simulation parameters. Each technique is implemented and evaluated against the specified QoS parameters. We run each test 10 times and take the average results to assess the algorithms' performance. Since big data encompasses heterogeneous tasks of varying sizes and servers with different processing speeds, executing the algorithms only once may not produce reliable results. Therefore, we run the algorithms 20 times and average the outcomes before comparing the various parameters.

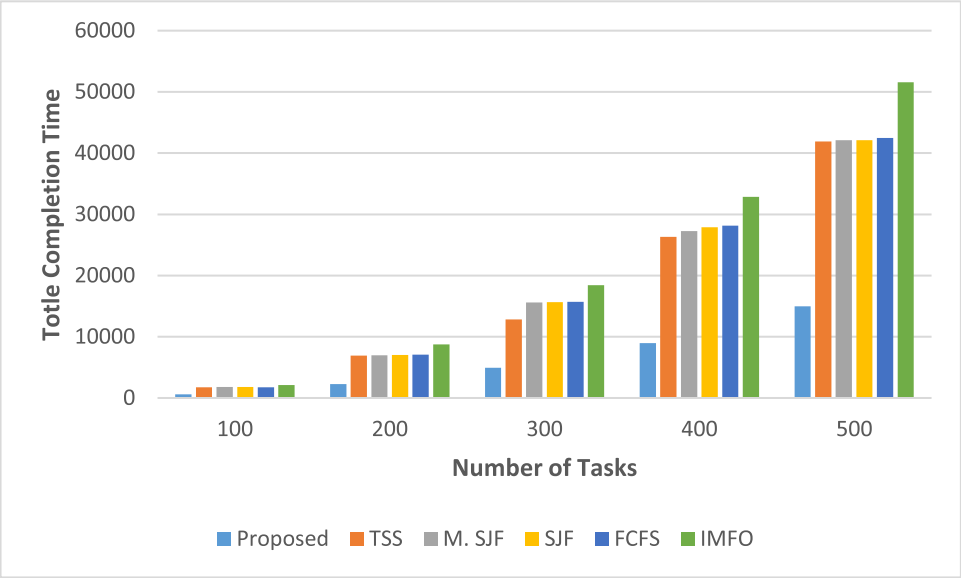

Fig. 3. Comparison of total completion time.

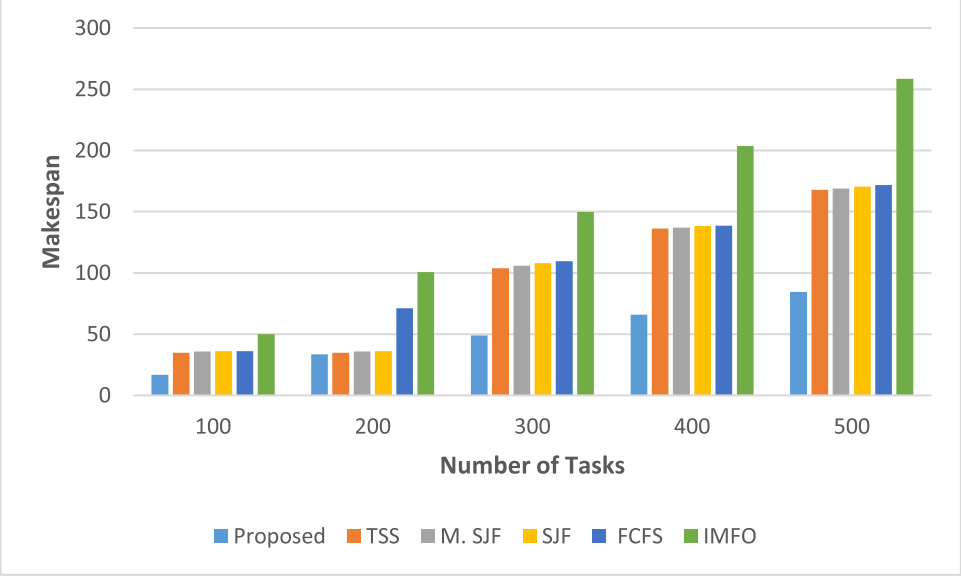

Fig. 4. Comparison of makespan.

Average Response Time

Our proposed algorithm responds faster than others when tested on various task sizes from 100 to 500. The average response time of PTAC ranges from 5 to 29 seconds across 100 to 500 tasks. The average response time of TSS is better than that of the other techniques, while SJF has a higher response time than all the other heuristics considered. The proposed algorithm is 66.67% faster than TSS and 67.39% faster than SJF. Compared to modified M.SJF, FCFS, and IMFO, PTAC responds 67.15%, 66.97%, and 72.84% faster across 100 to 500 tasks. A comparison of different techniques is shown in Fig. 2.

Total Completion Time

PTAC executes the task faster than the other considered heuristics. The total completion time of PTAC ranges from 571.794 to 14980.78 μs, over 100 to 500 tasks. Subsequently, TSS performs better than the others. However, the FCFS method reports the highest total completion time. Our proposed method outperforms TSS by 64.64% and FCFS by 66.67%. Additionally, PTAC optimized the total completion time by 66.15%, 66.42%, and 72.01% as compared to the Modified SJF, SJF, and IMFO techniques. A graph of the total completion time results is shown in Fig. 3.

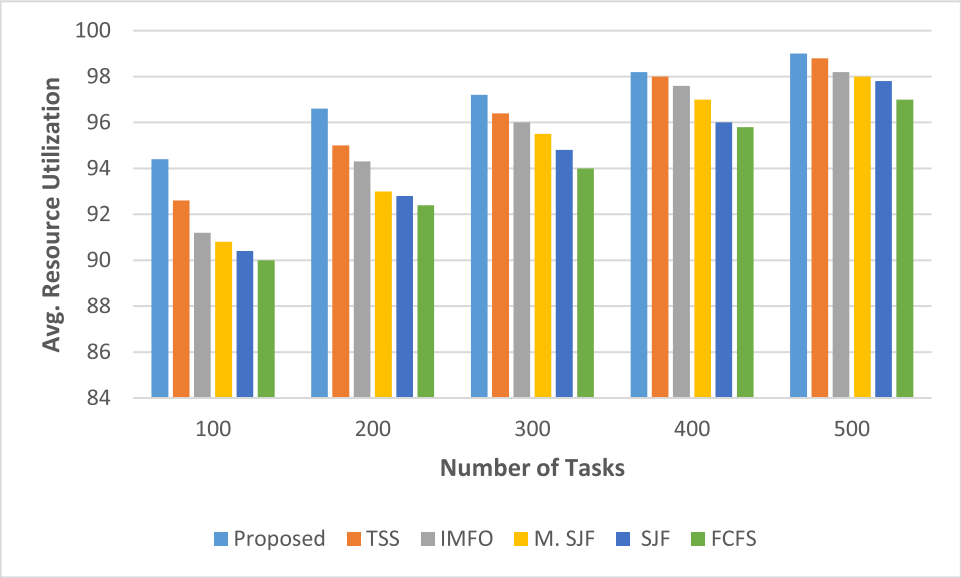

Fig. 5. Comparison of average resource utilization.

Makespan

The makespan of the proposed algorithm for 100 to 500 tasks ranges from 17 to 84 μs. The makespan of our method is shorter than that of the other techniques considered. TSS performs better than SJF, Modified SJF, and FCFS. FCFS yields the highest makespan among all the approaches considered. With the modified TSS, PTAC performs 47.62% better, surpassing FCFS by 52.55%. When compared with modified SJF, SJF, and IMFO, it performs 48.530%, 48.89%, and 67.72% better, respectively. The graph illustrating the makespan is shown in Fig. 4.

Average Recourse Utilization

Our proposed algorithm utilizes resources more efficiently than the other algorithm. PTAC increases resource utilization from 94% to 99% for 100 to 500 tasks. TSS also improves resource efficiency, similar to our algorithm. In contrast, FCFS results in the lowest resource utilization. The proposed algorithm uses 0.94% more resources than TSS and 3.34% more than FCFS. When compared with IMFO, Modified SJF, and SJF, the proposed method outperforms IMFO by 1.66%, Modified SJF by 2.16%, and FCFS by 2.80%. The graph of average resource utilization is shown in Fig. 5.

Conclusion

Big data is generated from diverse sources containing varying numbers of tasks. These tasks require different types of resources for execution. Therefore, there is a need for a resource allocation strategy that efficiently supplies resources to these various types based on their needs. Our proposed task categorization and resource optimization algorithm, called PTAC, classifies these tasks and allocates resources according to their requirements. Data-intensive tasks demand more storage capacity, while compute-intensive tasks require greater computational power. Through task categorization, we allocate computational resources to compute-intensive tasks in a way that ensures each task receives the best possible server while minimizing its runtime. We also compare our proposed algorithm with FCFS, SJF, M.SJF, and TSS based on QoS metrics, including average response time, total completion time, makespan, and average resource utilization. The proposed algorithm optimizes these metrics effectively, achieving average resource utilization from 94% to 99%, average response times ranging from 4 to 16, total completion times from 571 to 14980, and makespan from 16 to 84 over 100 to 500 tasks. We used Python to create the environment. In the future, we will further optimize data-intensive tasks and improve other QoS parameters.

Limitations: Although the proposed PACT algorithm categorizes tasks and allocates resources efficiently, it still has some limitations. Handling real-time workload is challenging in this approach because it can decrease the system's efficiency and cause overload during dynamic task execution. It is important to manage dynamic load effectively to enable the algorithm to manage real-time workload effectively. Additionally, the threshold coefficient needs improvement, as tasks are dynamic and may face difficulties when the task volume and server processing power increase. Furthermore, designing the

algorithm to be reliably applicable in time-sensitive systems is essential.

Future Work: In the future, we will enhance our algorithm by incorporating features such as task priority, task arrival time, etc. We also plan to expand our experiments to include a larger number of servers and real-world workloads to better study scalability and practical application. We will also include more parameters for efficient task scheduling. Additionally, we can incorporate a dynamic, AI/ML-driven strategy suitable for elastic cloud conditions to show resiliency and increased significance in the proposed research. The same can be tested for real real-world cloud environment. Moreover, we can modify our approach by including a load-balancing strategy for both data-intensive and compute-intensive tasks. Besides, we can also consider other categories of tasks like memory and I/O, among others.

Authors' declaration

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are ours. Furthermore, any Figures and images that are not ours have been included with the necessary permission for republication, which is attached to the manuscript.
- No animal studies are present in the manuscript.
- No human studies are present in the manuscript.
- Ethical Clearance: The project was approved by the local ethical committee at Lovely Professional University.

Authors' contributions statement

P.B. and S. S. conceived the core idea of the study, contributed to the development of the proposed PATC model, and led the simulation and evaluation processes. Participated in reviewing the existing mechanisms, enriched the paper with theoretical background, and helped align the study with current research standards. P. B., S. S., and A. N. assisted in algorithm design, conducted performance analysis, and contributed significantly to the writing and technical refinement of the manuscript. Also, enhanced the methodological framework and contributed to the interpretation of QoS evaluation metrics. A. N. and R. N. supported the literature review, comparative analysis with existing methods, and helped validate simulation outcomes. Also, supervised the overall research activity, ensured academic and technical quality, and finalized the manuscript for submission.

References

1. Janev V, Pujić D, Jelić M, Vidal ME. Chapter 9 survey on big data applications. In Knowledge Graphs and Big Data Processing 2020 Jul. 16 pp. 149–164. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-53199-7.
2. Hussain T, Sanga A, and Mongia S. Big data hadoop tools and technologies: A review. In Proceedings of International Conference on Advancements in Computing & Management (ICACM). 2019 Oct 1. pp. 574–578. https://dx.doi.org/10.2139/ssrn.3462554.
3. Duarte F. Amount Data Created Daily (2026). Exploding Topics. 2026.
4. Abualigah L, Masri BA. Advances in MapReduce big data processing: platform, tools, and algorithms. Artificial intelligence and IoT: Smart convergence for eco-friendly topography. 2021 Feb 13:105–28. https://doi.org/10.1007/978-981-33-6400-4_6.
5. Cai Y, Llorca J, Tulino AM, Molisch AF. Compute-and data-intensive networks: The key to the metaverse. In 2022 1st international conference on 6G networking (6GNet) 2022 Jul 6. pp. 1–8. IEEE. https://doi.org/10.1109/6GNet54646.2022.9830429.
6. Compute-Intensive vs Data-Intensive Workloads | Seagate India, Last Updated: 2025.
7. Pauley W. Cloud provider transparency: An empirical evaluation. IEEE Security & Privacy. 2010 Aug 19;8(6):32–9. https://doi.org/10.1109/MSP.2010.140.
8. Younis MF. Enhancing cloud resource management based on intelligent system. Baghdad Sci J. 2024 Jun 1;21(6):2156–66. https://doi.org/10.21123/bsj.2023.8507.
9. Yao C, Wang J, Sun H, Chu H, Jin T, Xiang Q. A data-driven method for adaptive resource requirement allocation via probabilistic solar load and market forecasting utilizing digital twin. Solar Energy. 2023 Jan 15;250:368–76. IEEE. https://doi.org/10.1016/j.solener.2023.01.005.
10. Raicu I, Foster I, Zhao Y, Szalay A, Little P, Moretti CM, at el. Towards data intensive many-task computing. InData Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management. IGI Global Journal. 2012:28–73. https://doi.org/10.4018/978-1-61520-971-2.ch002.
11. Shiekh S, Shahid M, Sambare M, Haidri RA, Yadav DK. A load-balanced hybrid heuristic for allocation of batch of tasks in cloud computing environment. Int. J. Pervasive Comput. Commun. 2023 Nov 16;19(5):756–81. https://doi.org/10.1108/IJPCC-06-2022-0220.
12. Pachipala Y, Sureddy KS, Kaitepalli AS, Pagadala N, Nalabothu SS, Iniganti M. Optimizing task scheduling in cloud computing: An enhanced shortest job first algorithm. Procedia Comput. Sci. 2024 Jan 1;233:604–13. https://doi.org/10.1016/j.procs.2024.03.250.
13. Yakubu IZ, Aliyu M, Musa ZA, Matinja ZI, Adamu IM. Enhancing cloud performance using task scheduling strategy based on resource ranking and resource partitioning. Int. J. Inf. Technol. 2021 Apr;13(2):759–66. https://doi.org/10.1007/s41870-020-00594-7.
14. Radhika D, Duraipandian M. Optimized dynamic task scheduling in cloud computing for big data processing. Wirel. Netw. 2025 May 5:1–2. https://doi.org/10.1007/s11276-025-03954-y.
15. Ma S, Chen J, Zhang Y, Shrivastava A, Mohan H. Cloud based resource scheduling methodology for data-intensive smart cities and industrial applications. Scalable Computing: Practice and Experience. 2021 Oct 24;22(2):227–35. https://doi.org/10.12694/scpe.v22i2.1899.

16. Alkhalaileh M, Calheiros RN, Nguyen QV, Javadi B. Data-intensive application scheduling on mobile edge cloud computing. J. Netw. Comput. Appl. 2020 Oct 1;167:102735. https://doi.org/10.1016/j.jnca.2020.102735.
17. Ibrahim IA, Bassiouni M. Improvement of job completion time in data-intensive cloud computing applications. J. Cloud Comput. 2020 Feb 7;9(1):8. https://doi.org/10.1186/s13677-019-0139-6.
18. Sadegh S, Zamanifar K, Kasprzak P, Yahyapour R. A two-phase virtual machine placement policy for data-intensive applications in cloud. J. Netw. Comput. Appl. 2021 Apr 15;180:103025. https://doi.org/10.1016/j.jnca.2021.103025.
19. Tuli S, Sandhu R, Buyya R. Shared data-aware dynamic resource provisioning and task scheduling for data intensive applications on hybrid clouds using Aneka. Future Gener. Comput. Syst. 2020 May 1;06:595–606. https://doi.org/10.1016/j.future.2020.01.038.
20. Wang H, Liu G, Shen H. Cooperative job scheduling and data allocation in data-intensive parallel computing clusters. IEEE Trans. Cloud Comput. 2022 Sep 13;11(3):2392–406. https://doi.org/10.1109/TCC.2022.3206206.
21. Zhang S, Zhao Z, Liu C, Qin S. Data-intensive workflow scheduling strategy based on deep reinforcement learning in multi-clouds. J. Cloud Comput. 2023 Aug 26;12(1):125. https://doi.org/10.1186/s13677-023-00504-9.
22. Mishra K, Rajareddy GN, Ghugar U, Chhabra GS, Gandomi AH. A collaborative computation and offloading for compute-intensive and latency-sensitive dependency-aware tasks in dew-enabled vehicular fog computing: A federated deep Q-learning approach. IEEE Trans. Netw. Service Manag. 2023 Jun 5;20(4):4600–14. https://doi.org/10.1109/TNSM.2023.3282795.
23. Gao Z, Liu W, Suo L, Li J, Lu Y. Deep reinforcement learning based compute-intensive workload allocation in data centers with high energy efficiency. In 2021 IEEE/CIC International Conference on Communications in China (ICCC) 2021 Jul 28. pp. 334–339. IEEE. https://doi.org/10.1109/ICCC52777.2021.9580316.
24. Zhang Z, Yang D, Zhou X, Cheng D. MCFuser: High-performance and rapid fusion of memory-bound compute-intensive operators. InSC24: International Conference for High Performance Computing, Networking, Storage and Analysis 2024 Nov 17. pp. 1–15. IEEE. https://doi.org/10.1109/SC41406.2024.00040.
25. Rahmani AM, Chamzini EY, Pourshaban M, Hosseinzadeh M. Scheduling of big data workflows in the hadoop framework with heterogeneous computing cluster. Arab. J. Sci. Eng. 2024 Nov 25:1–3. https://doi.org/10.1007/s13369-024-09779-9.
26. Chai J, Meng Y, Wang W, Lyu Y, Yue H, Liu X. Task scheduling of computation-intensive graph jobs in UAV-assisted hybrid vehicular networks. Veh. Commun. 2023 Aug 1;42:100630. https://doi.org/10.1016/j.vehcom.2023.100630.
27. Ouhakki H, Elmoufidi A. An implementation of GPU accelerated mapreduce: using hadoop with openCL for breast cancer detection and compute-intensive jobs. Int. J. Inf. Techno. 2024 Sep 2:1–8. https://doi.org/10.1007/s41870-024-02171-8.
28. Li Q, Edahiro M. Template-based automatic library function generation with halide for compute-intensive simulink models. In 2024 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS) 2024 Apr 17. pp. 1–6. IEEE. https://doi.org/10.1109/COOLCHIPS61292.2024.10531173.
29. Arora R. An introduction to big data, high performance computing, high-throughput computing, and hadoop. In Conquering Big Data with high performance computing 2016 Sep 17. pp. 1–12. Cham: Springer International Publishing.. https://doi.org/10.1007/978-3-319-33742-5_1.
30. Yakubu IZ, Aliyu M, Musa ZA, Matinja ZI, Adamu IM. Enhancing cloud performance using task scheduling strategy based on resource ranking and resource partitioning. Int. J. Inf. Techno. 2021 Apr;13(2):759–66. https://doi.org/10.1007/s41870-020-00594-7.
31. Ahmad SG, Liew CS, Rafique MM, Munir EU. Optimization of data-intensive workflows in stream-based data processing models. J. Supercomput. 2017 Sep;73(9):3901–23. https://doi.org/10.1007/s11227-017-1991-0.
32. Xuan P, Luo F, Ge R, Srimani PK. DynIMS: A dynamic memory controller for in-memory storage on HPC systems. arXiv preprint arXiv: 1609.09294. 2016 Sep. https://doi.org/10.48550/arXiv.1609.09294.
33. Braun TD, Siegel HJ, Beck N, Bölöni LL, Maheswaran M, Reuther AI, *et al.* A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. 2001 Jun 1;61(6):810–37. https://doi.org/10.1006/jpdc.2000.1714.

تحسين جودة الخدمة في الحوسبة السحابية: مصنف مهام مُراعي للأداء لأحمال عمل البيانات الضخمة غير المتجانسة

باوان بهاكر[1]، صوفيا شيخ[1]، رينتو ناث[2]، أجاي ناين[1]

[1] قسم تطبيقات الحاسوب، جامعة لوفلي بروفيشنال، فاغوارا، البنجاب، الهند.
[2] قسم العلوم والتكنولوجيا، نيودلهي، الهند.

الخلاصة

تنشأ البيانات الضخمة من مصادر متنوعة وغير متجانسة حول العالم. ونظرًا لتنوع مصادر توليد البيانات، فإنها تتضمن مهامًا متعددة. بعضها يتطلب موارد تخزين، بينما يحتاج البعض الآخر إلى موارد حاسوبية. بالإضافة إلى ذلك، توفر الحوسبة السحابية موارد تخزين وحوسبة مركزية لتنفيذ هذه المهام. ومع ذلك، لا يزال تحسين الموارد وجدولة المهام بكفاءة يمثلان تحديًا. علاوة على ذلك، يُمثل تصنيف المهام بدقة وتخصيص الموارد بفعالية بناءً على طبيعتها تحديًا للحوسبة السحابية، لا سيما عند إسناد المهام إلى موارد غير متجانسة لتقليل وقت إنجازها. ولمعالجة هذه المشكلات، تقترح هذه الورقة البحثية خوارزمية تصنيف المهام الواعية بالأداء (PATC) التي تصنف المهام وتخصص الموارد وفقًا لذلك مع تحسين معايير جودة الخدمة. تقسم خوارزمية تصنيف المهام المهام إلى فئتين: مهام كثيفة البيانات ومهام كثيفة الحوسبة. وتُستخدم معاملات عتبة لوقت الإرسال واستخدام وحدة المعالجة المركزية لتصنيف المهام. علاوة على ذلك، تُخصص الموارد للمهام كثيفة الحوسبة لتقليل وقت تنفيذها قدر الإمكان. لتقييم أداء النظام، تم تحليل معايير جودة الخدمة المختلفة، مثل متوسط زمن الاستجابة، وإجمالي زمن الإنجاز، ومدة إنجاز جميع المهام، ومتوسط استخدام الموارد. تُظهر نتائج المحاكاة أن منهجنا يُحسّن هذه المعايير بفعالية مقارنةً بالأساليب الحالية، مثل خوارزمية أسبقية الوصول (FCFS)، وخوارزمية أقصر مهمة أولاً (SJF)، وخوارزمية أقصر مهمة أولاً المعدلة، واستراتيجية جدولة المهام (TSS)، وخوارزمية تحسين اللهب المحسّنة (IMFO).

الكلمات المفتاحية: البيانات الضخمة، الحوسبة السحابية، المهام المكثفة، جودة الخدمة، تحسين الموارد، تصنيف المهام.