Sustainable Computing: Informatics and Systems 41 (2024) 100942

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Priority based job scheduling technique that utilizes gaps to increase the efficiency of job distribution in cloud computing☆

Saydul Akbar Murad [a,d], Zafril Rizal M. Azmi [a,*], Abu Jafar Md. Muzahid [a], Md. Murad Hossain Sarker [b], M. Saef Ullah Miah [a], MD. Khairul Bashar Bhuiyan [c], Nick Rahimi [d], Anupam Kumar Bairagi [e]

[a] Faculty of Computing, College of Computing & Applied Sciences, Universiti Malaysia Pahang, Pekan, Pahang, 26600, Malaysia
[b] Comilla University (CoU), Comilla, 3506, Bangladesh
[c] Brac University, Dhaka, 1212, Bangladesh
[d] School of Computing Sciences & Computer Engineering, University of Southern Mississippi, Hattiesburg, 39401, USA
[e] Khulna University, Khulna, 9208, Bangladesh

ARTICLE INFO

Keywords:
Cloud computing
Job scheduling
Backfilling
Resource management
Gap searching
SJF
LJF
FCFS

ABSTRACT

A growing number of services, accessible and usable by individuals and businesses on a pay-as-you-go basis, are being made available via cloud computing platforms. The business services paradigm in cloud computing encounters several quality of service (QoS) challenges, such as flow time, makespan time, reliability, and delay. To overcome these obstacles, we first designed a resource management framework for cloud computing systems. This framework elucidates the methodology of resource management in the context of cloud job scheduling. Then, we study the impact of a Virtual Machine's (VM's) physical resources on the consistency with which cloud services are executed. After that, we developed a priority-based fair scheduling (PBFS) algorithm to schedule jobs so that they have access to the required resources at optimal times. The algorithm has been devised utilizing three key characteristics, namely CPU time, arrival time, and job length. For optimal scheduling of cloud jobs, we also devised a backfilling technique called Earliest Gap Shortest Job First (EG-SJF), which prioritizes filling in schedule gaps in a specific order. The simulation was carried out with the help of the CloudSim framework. Finally, we compare our proposed PBFS algorithm to LJF, FCFS, and MAX–MIN and find that it achieves better results in terms of overall delay, makespan time, and flow time.

1. Introduction

With the rapid adoption of virtualized Information technology (IT) components such as databases, servers, and on-demand storage, a growing variety of cloud services are being delivered to consumers and businesses under a billing model over the wired and wireless Internet. This is because cloud computing, which is the most popular model for resource delivery, allows any resource to be delivered as a service, including networks, platforms, software, and applications. Similar to social economy services, quality of service (QoS) requirements in the cloud, such as cost, energy consumption, security, and reliability, are the main challenges for cloud service providers, which are receiving more and more attention from enterprises and academia [1–3]. Among these obstacles, the flow time, makespan time, and resource utilization are three of the most important concerns for cloud customers and providers.

Currently, popular enterprise schedulers, like First Come First Served (FCFS) [4], are widely used to manage jobs because of their simplicity and flexibility of implementation. FCFS schedules jobs based on the order in which they are submitted and the availability of resources required for each job. Although FCFS offers several benefits and is widely used, its fairness attribute often results in low utilization of resources because jobs cannot be executed if there are older jobs in the queue, even if the resources are available that are required. In addition, the FCFS approach is ineffective for scheduling interactive

☆ All persons who have made substantial contributions to the work reported in the manuscript (e.g., technical help, writing and editing assistance, general support), but who do not meet the criteria for authorship, are named in the Acknowledgments and have given us their written permission to be named. If we have not included an Acknowledgments, then that indicates that we have not received substantial contributions from non-authors.

* Corresponding author.
E-mail addresses: zafril@ump.edu.my (Z.R.M. Azmi), mrumi98@gmail.com (A.J.M. Muzahid), mh6367828@gail.com (M.M.H. Sarker), md.saefullah@gmail.com (M.S.U. Miah), kbashar707@gmail.com (M.K.B. Bhuiyan), nick.rahimi@usm.edu (N. Rahimi), anupam@cse.ku.ac.bd (A.K. Bairagi).

https://doi.org/10.1016/j.suscom.2023.100942
Received 10 April 2023; Received in revised form 17 November 2023; Accepted 22 November 2023
Available online 30 November 2023
2210-5379/© 2023 Elsevier Inc. All rights reserved.

users because it cannot guarantee a reasonable flow time. Alternative schedulers can be used instead of FCFS, which also belongs to the priority rule (PR) family. Each of these algorithms has its own merits that affect how well the scheduler performs. However, there are no meaningful details from previous work that explain the advantages and disadvantages of these algorithms in a cloud environment.

In the field of job scheduling, priority rule algorithms using a queue-based technique have been widely used. The primary issue with those algorithms is to ensure that the performance of the various metrics remains in balance. When it comes to certain metrics, these algorithms perform well, but they do not perform as well when it comes to other things. Shortest Job First (SJF) [5] has a low flow time but has a long makespan. On the other hand, Minimum Time to Due Date (MTTD) [6], and Longest Job First (LJF) [7], have a low makespan, while LJF has a high flow-time, high delay, and low resource utilization.

The backfilling technique and runtime estimates make the schedule-based techniques more flexible than the queue-based techniques [8]. Scheduling is done by backfilling, where there is no fixed order. As jobs arrive at different times, there are gaps in the machine's schedule. These gaps are filled by short jobs. In this way, resources are utilized even better. People said backfilling was like getting something for nothing, or a benefit without a cost. In a queue-based algorithm, backfilling is possible, but scheduling is done as needed by the queues. On the other hand, in a schedule-based algorithm, incoming jobs are backfilled in advance based on the user's estimated processing time.

Conservative Backfilling (CONS) [8] and Extensible Argonne Scheduling System (EASY) [9] are the two most popular backfilling scheduling algorithms. EASY uses a "queue-based" method, while CONS uses a "schedule-based" method. The goal of the EASY method is to make the best use of resources by backfilling the gaps with unfinished jobs. The jobs that are further ahead in the queue have to wait longer due to this method. EASY only cares that the first job does not fall behind the schedule. The rest of the jobs fall behind because the backfilled job takes the place of the jobs waiting in the queue. CONS uses a schedule-based method to fill jobs that are not completed. To work around the vulnerabilities of EASY, the user estimates the runtime. Runtime estimates inform the scheduler about the capabilities and availability of resources, as well as when jobs will start and how long they will take.

In order to solve the above challenges, this paper presents an architecture for cloud resource management and analyzes the reliability of cloud service execution and proposes a backfilling technique based on the Earliest Gap Shortest Job First (EG-SJF), and a priority-based fair scheduling algorithm (PBFS). In this paper, we represent diverse fundamental contributions that can be summarized as follows::

1. First, we build an architecture for resources management in cloud computing, which consists of submitting jobs, service request authentication, resource availability, resource selection, resource scheduling and allocation, physical resources layer, and VMs layer.
2. Secondly, we study the execution reliability of cloud services caused by VM software and physical resources and analyze the service request overflow.
3. Thirdly, a scheduling strategy for cloud computing has been developed. All of the performance metrics utilized in our experiment are analyzed here.
4. Fourthly, we checked the availability of the resources provided by the cloud service provider using a gap search technique.
5. Fifthly, a priority-based fair scheduling (PBFS) is used to increase the efficiency of job scheduling. The algorithm tries to reduce the flow time, makespan time, and total tardiness.
6. Sixthly, we propose a backfilling technique called the Earliest Gap Shortest Job First (EG-SJF) to utilize the available resources.
7. Finally, we employ the CloudSim simulator to test how well the proposed method works. Compared to LJF, FCFS, and MAX–MIN, the proposed PBFS algorithm significantly reduces flow time, makespan time, and total tardiness.

This research paper is put together in the following way: In Section 2, related work is looked at. In Section 3, we describe an architecture for cloud resource management. In Section 4, we talked about cloud service reliability analysis. In Section 5, we talked about the cloud computing scheduling strategy. Here, we proposed a priority-based algorithm and a backfilling technique. In Section 6, we look at the results of our experiments and evaluate our proposed PBFS algorithm. In Section 7, we talked about what we think needs to be done next in terms of research.

2. Related work

Recently, both cloud users and providers have been paying close attention to the QoS of cloud computing systems, which includes factors such as makespan time, security, energy consumption, response time, resource utilization, cost, and reliability [10–16]. These QoS concerns have a great impact on the performance of cloud computing systems. In their study, [12] provided multi-criteria decision models to evaluate cloud services from numerous QoS aspects for system selection. In most cases, we only pay attention to these QoS characteristics. [15] presented the efficient resource utilization (ERU) model, which aims to use cache resources more efficiently while reducing last-level cache failure and meeting job deadline requirements. In their study, [14] focused their attention on the reliability of QoS-aware task scheduling and time consumption in cloud systems. They developed the quantitative fault-tolerant scheduling (QFEC) algorithms that reduce both the execution cost and the length of scheduling. In the research paper, [11] proposed a Best Response Algorithm (BRA) to minimize the total cost of cooperative cloud and edge systems. This was done in response to the constraint of quality-of-service response time. In this paper, [17] proposes a Deep Reinforcement Learning (DRL)-based task scheduler that reduces the cost of executing jobs on virtual instances. According to the results of this research, QoS-aware techniques have the potential to significantly improve the efficiency of cloud computing systems and meet the needs of a wide range of cloud users. Therefore, it is a useful research work to increase the QoS of cloud services.

Cloud computing service reliability is also a major concern for industry and academia as an element of the QoS of a cloud computing system [18]. As a strategy to improve the quality of service reliability, checkpointing is a method that preserves the current execution state as an image file. Cloud computing systems with their huge VM populations require a lot of additional storage, so this is an impractical solution. Another useful mechanism for ensuring reliability is fault-tolerant techniques based on replication, such as trying to map each primary VM to a single or more backup VMs [18–23]. In this study, [18] provides an approach to optimize the location of redundant VMs to improve the reliability of cloud services. A heuristic method was used to fix the optimization problem of reassigning tasks to VMs. Defects Per Million (DPM), developed by [20], is a service-oriented reliability metric that measures how many user requests out of one million are aborted due to VM failures. Based on the checkpointing strategy, they provide an analytical modeling approach to determine the DPM metric in different replication systems. [24] proposed an optimal strategy for placing VMs in accordance with the performance requirements of application services. In addition, this technique was tolerant to the failure of a number of k-host servers.

In this paper, [25] proposed a backfilling technique for scheduling deadline-based jobs. In this study, the current system time was employed as a scheduling parameter alongside the start time and the due date. In this article [26] provides an algorithm for reliable and fast response to cloud clients' queries. Cooperative Computing

System (CCS) concepts are used for the algorithm. The performance of the algorithm is evaluated based on the average task delay during a simulation session. The goal of this study [27] is to solve the job scheduling problem by dividing tasks into different processing threads based on their duration. Here, a new concept for a backfilling algorithm is implemented to minimize the idle time of the processors. In addition to tabu search, this study [12] presents a hybridization of the best-gap mechanism known as the Swift-gap mechanism. In addition, a new decision rule was incorporated into the outcome mechanism based on the time taken to complete the task. The goal of this study [28] is to develop an algorithm for backfilling grid services that results in increased resource utilization, improved fault tolerance, and the elimination of deadlocks. However, to avoid the limitations of the existing backfilling technique, we have proposed the EG-SJF algorithm. The objective of gap filling is to increase system usage of the FCFS-based scheduler. Although FCFS is a simple and commonly utilized policy, it suffers from poor system utilization.

Jobs can be prioritized and scheduled using the priority scheduling method. The scheduler uses a weighted relevance ranking to choose which jobs to do. This study [29] proposes a two-phase approach to planning for a geo-distributed data center. In the first phase, a suitable data center is chosen to process the user's request, and in the second phase, a suitable VM is chosen. The objective is to lessen the number of incidents where Service-Level Agreements (SLAs) are broken with regard to availability, reaction time, and dependability. A new algorithm for Prioritization-based Job Scheduling (PJSC) in cloud computing was proposed in this study [30]. The proposed strategy is founded on a multi-factor decision-making paradigm. In this research [31], a three-tier Edge–Cloud Collaborative REM (ECCREM) architecture is presented in order to increase latency and processing speed. In this research [32], we see how the priority-based work scheduling approach in the cloud can be enhanced by employing a multi-attribute, multi-criterion decision-making model. [33] offers a technique for planning workflows that take variable job priority into account. To determine which VM should be responsible for a given job, the method first conducts min–max normalization and then computes the dynamic threshold. This study intends to enhance the current PR cloud schedulers, such as FCFS, by creating a novel dynamic scheduling algorithm that takes use of gaps in the cloud work schedule to solve this problem.

The drawback of current scheduling techniques is that they only focus on the migration of the data or processes from local/mobile processor to the cloud server, whether complete or partial offload, without taking into consideration that in the cloud, a single process might be handled by more than single resources which requires proper scheduling. This is because, the works mentioned in the above references used priority rules (PR) such as FCFS and SJF, which have been proven to be not optimized. Therefore, to improve the performance of PR in terms of machine usage, number of delayed jobs, flowtime, makespan, and total tardiness, it requires a high-performance cloud scheduler. In order to achieve this, a new gap-filling technique is needed to efficiently utilize gaps between jobs in the schedule generated by PR. *Table* 1 provides an overview of the comparison between our study and other comparable work.

3. Cloud resource management architecture

The diagram in *Fig.* 1 shows how cloud computing systems manage their resources. The architecture consists of job requests from users, a job queue, authentication, resource availability, resource selection, resource scheduling, resource allocation, a layer of VMs, and a layer of physical resources. This system is in charge of handling job requests and showing users the results of computations. This architecture supports the scheduling strategy and service reliability analysis. In this section, first, we define the labels for cloud job requests. Then, we discuss the authentication of the requested services and the availability of the resources. After that, we discuss resource selection, scheduling, and allocation. After that, we introduce a cloud virtualization layer and a list of virtual machine attributes. In this section, the reliability of cloud services is defined, and the causes of service failures are also discussed. *Table* 2 shows the notation of symbols used in the paper.

3.1. Cloud computing job requests

Service-Oriented Architecture (SOA) is a popular method for cloud computing systems to handle a large number of concurrent service requests from customers [38]. The requests for cloud services are made by the users and placed in the queue of the system. Finally, these job requests are assigned to VMs based on parameters such as finish time, cost, and reliability. VMs can be dynamically created based on the characteristics of the user service request.

In real-world applications, there are many separate and interrelated cloud job requests. With the increasing popularity of the Internet of Things (IoT) and 5G wireless technology, microservice requests are becoming more common, and many of them are unrelated. Therefore, in this paper, we consider cloud job requests as autonomous and refer to them as a collection of services or jobs $J = j_1, j_2, j_3, \ldots \ldots j_n$. We describe the request size of cloud service as $W(j_i)$. We also represent the storage required for job j_i as sr (j_i) (G). Here, G denotes the Gigabytes (GB). art(j_i)(S) stands for the user's arrival time when submitting the job request, where S stands for second. Cloud users often have an expectation that their job requests will be successfully fulfilled within an acceptable amount of time, referred to as response time. A user's job request is submitted, and the response time rt(j_i)(S) measures the time it takes to receive the correct response. ft(j_i)(S) is the finish time of the user-submitted job.

3.2. User request authentication and availability of resources in cloud computing

Cloud-based authentication was introduced to protect companies from unauthorized access to sensitive data. Authorized users from around the world can securely access data stored in the cloud through what is known as cloud authentication [39]. Requests for cloud services are made by users and placed in the system queue. Before resource allocation, the system first checks whether the request is authentic or not. If the request is authentic, the availability of resources is checked; otherwise, the request is rejected.

The primary goal of moving to the cloud is high availability. Customers and employees should be able to access products, services, and tools from anywhere, at any time, and on any device, as long as the device has an Internet connection. The reliability of the cloud is linked to its accessibility. To ensure proper response to cloud users' requests under the given parameters and within a reasonable time frame, cloud computing systems must be reliable [21]. Cloud QoS places great emphasis on this aspect of trustworthiness, and both customers and service providers are concerned about it. Due to problems with cloud physical resources, VM software execution errors, and network delays, many cloud services cannot meet their service guarantees. In our proposed architecture, service availability is first monitored based on user requests. If not, this request is queued again unless the resources are available.

3.3. Resource selection, scheduling and allocation in cloud computing

At a given time, a large number of users will request services; however, the requirements of each user are different. Based on the nature of the request, the provider selects which resources are required for that request; this process is called resource selection [40]. Cloud resource scheduling refers to the process of allocating and assigning resources among cloud users according to certain rules in a given cloud environment. Cloud computing relies on a fundamental technology

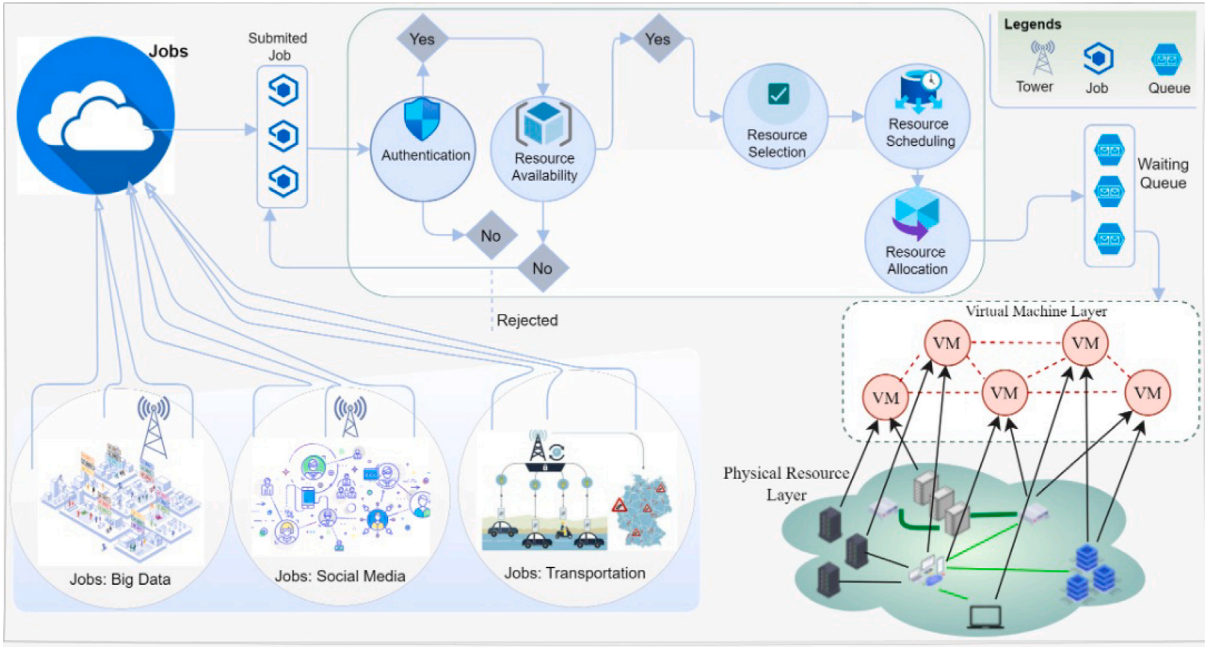

Fig. 1. Architecture of resources management in cloud computing.

Table 1
Comparison of Related Work.

Studies	Performance Improvement	Backfilling	Gap Search	Multi Objective	Reliability analysis
[18]	✓	✗	✗	✗	✓
[34]	✗	✗	✗	✓	✗
[11]	✓	✗	✗	✓	✗
[25]	✓	✓	✓	✗	✗
[35]	✓	✗	✗	✗	✓
[14]	✓	✗	✗	✓	✓
[36]	✗	✗	✗	✓	✗
[27]	✓	✓	✓	✓	✗
[37]	✓	✗	✗	✗	✓
[28]	✓	✓	✗	✓	✗
Our Work	✓	✓	✓	✓	✓

Table 2
Definition of symbols.

Symbol	Definition
J	Total Number of jobs or services.
W(J_i)	Size of requested job j_i.
rt(j_i)(S)	Response time of requested job (j_i) in second (S).
vm_j	the jth cloud systems VM.
ET(j_i, vm_j)	The execution time.
pt(j_i, vm_j)	The processing time of cloud job j_i on VM vm_j.
FT(j_i, vm_i)	The finish time.
RT(j_i, vm_j)	The running time.
T_j	Total tardiness of executed job j_i.
WT(j_i)	The Waiting Time.
AT(j_i)	Arrival Time
EST(j_i, vm_i)	Execution start time.
TAT(j_i, vm_i)	Turnaround Time

known as resource scheduling [41]. The CPU, network, and storage are assigned precise and accurate tasks via resource scheduling. The ultimate goal is to use the available resources as efficiently as possible. Both cloud service providers and cloud users need well-coordinated planning. The practice of allocating available resources to needed cloud applications over the Internet is called Resource Allocation (RA). If resource allocation is not handled accurately, services suffer. This challenge is solved by resource provisioning, which allows service providers to manage resources for each module.

3.4. Cloud VMs and physical resources

In most cloud computing systems, the physical resource layer consists of networks, storage, physical servers, and other devices. As seen in the physical resource layer in *Fig.* 1, these resources are concentrated, regionally scattered, or heterogeneous. Cloud virtualization platforms such as Xen, VMware, and KVM enable the dynamic partitioning of physical resources into several VMs or the reconstruction of those resources into a single VM.

The cloud virtualization layer in this study consists of a pool of VMs denoted by $\omega = vm_1, vm_2, \ldots..vm_n$. The processing capacity, computational cost, storage cost, and resource reliability guarantee of each cloud VM, $vm_j \in \omega$, are connected with several factors. The processing capacity of the VM is measured in millions of instructions per second (MIPS) and is denoted by the letter w(vm_j). Here, the cloud service j_i processing time pt(j_i, vm_j) of on VM vm_j is defined as:

$$pt\left(j_i, vm_j\right) = \frac{w\left(j_i\right)}{w\left(vm_i\right)} \tag{1}$$

The processing time is calculated by dividing the size of the job request $w(j_i)$ by the computational capacity of the cloud VM w(vm_j) and is measured in seconds.

3.5. Cloud service reliability

One of the most difficult aspects of cloud computing is ensuring the reliable execution of services on VMs [21]. In this part, we first present the method for computing the reliability of the cloud VM. In the next section, we address the problem of overflowing job requests. Each job has a certain amount of time to get the service. When this amount of time expires, the service is considered timeout. The most important factors in cloud computing are database and network outages, which also has been covered in the subsequent section.

4. Cloud service reliability analysis

One of the most difficult aspects of cloud computing is ensuring the reliable execution of services on virtual machines. In this part, we first present the method for computing the reliability of the cloud VM physical resources. In the next section, we address the problem of overflowing job requests. Each job has a certain amount of time to get the service, When this amount of time expires, the service is considered timeout. That is all we have been talking about. The most important factors in cloud computing are database and network outages. These issues will also be covered.

4.1. Failures of physical resources in cloud

Cloud computing systems include a variety of physical resources, and the majority of them are built on large Integrated Circuits (IC). Due to circuit crosstalk and delays, these high-density circuits are prone to sudden malfunctions. Second, high-density IC is susceptible to electron migration, gate oxygen breakdown, and hot carrier degradation, all of which lead to permanent failures. Wear, random failure, ageing, and lifetime failure are examples of physical resource failure. These failures are expected to follow a Poisson process [42].

The VM vm_i of cloud computing systems are essentially a set of A physical server, where PS = $ps_1, ps_2, \ldots \ldots ps_a$, networks ($sn$), and storage ($ss$). We consider that each resource has a fixed failure rate λ, such that the physical server, storage, and networks have failure rates of λ_{ps_a}, λ_{ss}, and λ_{sn}, respectively [42]. For a physical resource, the exponential probability density function (f), where, physical server λ_{ps_a}, is $\lambda_{ps_a}(t) = \lambda_{ps_a} e^{-}\lambda_{ps_a t}$ t. Consequently, the reliability function P_{ps_a} (t) of this resource can be given as follows [43].

$$P_{p_a}(t) = 1 - \int_0^t f_{ps_a}(t)dt = e^{\lambda_{ps_a t}} \tag{2}$$

The start execution time ET (j_i, vm_j) for the cloud service j_i processed by VM (vm_j). Here ET (j_i, vm_j) is the updated time of cloud available time ava(vm_j) and job arriving ar(j_i) of the VM, written as:

$$ET\left(j_i, vm_j\right) = \text{Max}\left\{\text{ar}\left(j_i\right), \text{ava}\left(vm_j\right)\right\} \tag{3}$$

As a result, the running time (s) for the service j_i on the VM vm_j is the sum of the start execution time, the VM running time, and the service processing time, as given by

$$RT\left(j_i, vm_j\right) = spt\left(j_i, vm_j\right) + ET\left(j_i, vm_j\right) + \text{runT}\left(vm_j\right) \tag{4}$$

4.2. Fault prediction of cloud VM software

The reliability of software execution ensures the delivery of the appropriate services under predetermined operating conditions, [35]. In contrast to the stability of the cloud's physical resources, software errors are produced by flaws in the coding, design, and intrinsic logic of user service requests. As time goes on, changes in cloud services, system upkeep, and cloud computing environment are the main causes of VM software execution issues. Numerous studies have demonstrated that software execution failure are time-series event [37,44,45]. Additionally, the majority of requests for cloud services have the characteristics of random, two-dimensional (2D) time series. For instance, during the working day, a high frequency of cloud online office requests happen between 8:10 and 11:50, while weekend cloud service requests are often few. In general, the likelihood of software errors increases with system workload. Therefore, software failure happens in a cloud VM is 2D, just like the time and day characteristics of the cloud system workload.

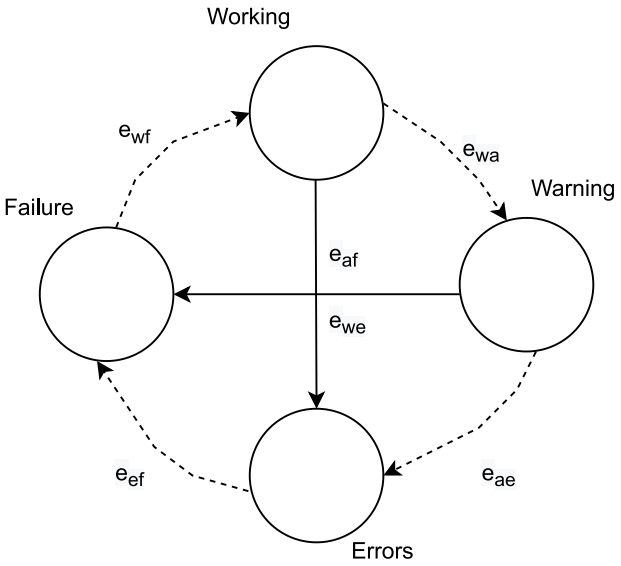

Fig. 2. Virtual machine running state diagram.

This study employs a failure prediction approach based on the AdaBoost Hidden Markov Model (AHMM) [33,46] to boost VM reliability. A AHMM is made up of the following: initial state probability π, state transition probability P, output observation probability B, hidden state space Y, and observation space X. First, VM's state observation set, and hidden state set are figured out. Here, the hidden state of VM is broken down into four states [46]: normal, error, warning, and failure, that is n = 4. For this, the transfer matrix of the probability of the hidden state is:

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix}$$

$$\sum_{j=1}^{4} p_{ij} = 1 \tag{5}$$

While a VM is active, the cloud infrastructure supports its use. The VM transitions to the warning state when its execution time grows if a software error (e_{wa}) occurs. Over time, the VM could enter an error (e_{ae}) or fault condition (e_{af}) if the cloud platform monitoring software does not catch the problem. In such a case, the VM fails because it is unable to complete its task. Errors (e_{we}), a failure state (e_{wf}), and an failure state ($e_e f$) can all be entered into a VM. Therefore, the AHMM is employed here to reveal the hidden security status of the VM in the cloud platform, as depicted in *Fig.* 2.

4.3. Overflow of service request

The maximum number of requests that can wait in the request queue should be limited. In the absence of this, new requests must sit in the queue for an excessive amount of time, which can increase the frequency of Timeout failures. Consequently, if the queue is full when

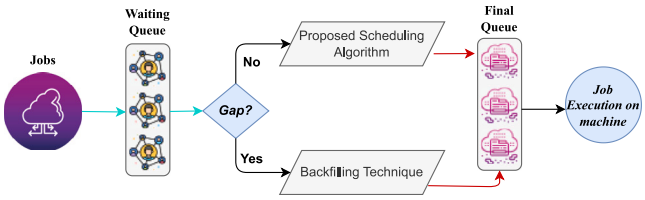

Fig. 3. Working procedure in cloud computing using new proposed (PBFS) algorithm and backfilling technique.

a new task request comes in, it is just dropped and the user cannot receive service, which is referred to as an overflow failure [47].

A threshold value (δ) for the job length may be provided by the service provider as a means of addressing the issue of an excessive number of service requests (OSR). Every new job that comes in has been labeled as J_i. A comparison is made between the length of the job and the threshold value. If the value of J_i is higher than the threshold value, then the job is placed in a primary queue known as Q_p. The value, on the other hand, is kept in the final queue Q_f. The overflow service request has been expressed by the following equation.

$$OSR = \max\left(\delta, J_i\right) \tag{6}$$

$$Q_p = OSR(OSR \approx \delta) \tag{7}$$

$$Q_f = \mathrm{OSR}\left(OSR \approx J_i\right) \tag{8}$$

5. Cloud computing scheduling strategy

In this part, we discuss the proposed PBFS algorithm. The primary objective of this algorithm is to enhance the overall tardiness, makespan time, and flow time. The subsequent section provides an explanation of the PBFS algorithm after a discussion on several performance indicators, namely Makespan Time, Total Tardiness, and Flow Time. The backfilling technique is also addressed in this section. In conclusion, an analysis is conducted to determine the temporal complexity of the PBFS algorithm.

The architecture of the cloud computing scheduler with priority rules is shown in *Fig. 3*. After users submit jobs to the cloud system, the jobs are forwarded to a FCFS queue, i.e., the first arriving job is placed at the top of the queue. First, the jobs in the queue are compared with the gap. The empty position is indicated by the availability of VM. If VM is available, the backfilling procedure is applied; otherwise, the PBFS algorithm is used. A new sorted queue — called the main queue, is then formed. The first job in the main queue is processed before all others.

5.1. Gap searching in cloud computing

To enhance the efficiency of the cloud scheduler, a search module has been developed to identify an appropriate gap or idle slot within the current schedule generated by the priority rule (PR) [48,49]. Upon detection, the gap will be aligned with a just-arrived job. This is accomplished by duly considering the time constraints associated with the completion of the jobs. Through the consistent allocation of appropriate jobs, a greater number of jobs can be accomplished prior to the established deadline, so ensuring the optimal utilization of available resources. The intricate relationship between PR and the suggested gap-search technique is outlined in *Algorithm 1*.

In accordance with the algorithm outlined in *Algorithm 1*, user-submitted jobs will be placed in the waiting queue using the FCFS scheduling policy, wherein the job that arrived first will be positioned at the head of the queue. The subsequent procedure involves doing a search for the most optimal machine capable of executing the job

Algorithm 1: Gap Search Technique

```
1: resource_size = Total number of resources in the cloud;
2: for i = 0 to resource_size do
3:    if number of available resource < Number of resource requested
      by Job_m then
4:       break operation;
5:    else
6:       machine_i is ready to perform Job_m;
7:    end if
8:    gapInfo = perform SearchGap for machine_i schedule;
9:    if gapInfo = true (Available gap found in machine_i schedule)
      then
10:      Insert Job_m into machine_i schedule based on backfilling
         technique;
11:   end if
12:   if gapInfo = false (Suitable gap is not found in machine_i
      schedule) then
13:      Insert Job_m into machine_i schedule based on proposed PBFS
         algorithm;
14:   end if
15: end for
```

positioned at the forefront of the queue. In light of this rationale, a comprehensive examination of all machines will be conducted to ascertain the presence of any gaps within the designated machine schedule ($machine_i$). The first line of *Algorithm 1* specifies that the variable *resourceSize* will be utilized to hold the number of resources present in the cloud. The term *resourceSize* refers to the state in which resources are accessible online, regardless of whether they are idle or actively processing data. The process of looping in line 2 entails the systematic examination of many internet resources with the objective of identifying the most optimal answer.

As the gap only appears in the queue when several jobs exist at the same cycle time, the FCFS is the initial scheduler used to allocate the job to the specific machine schedule. Based on the FCFS the waiting queue is created. Our proposed algorithm PBFS will be activated if there is no suitable gap for the respected job. However, if there is even one gap that can be filled in a new job, the backfilling technique is used. It also means that backfilling technique always be the main priority compared to PR. Unlike the traditional technique, this backfilling technique considers not only small jobs but applies to all new jobs that enter the system.

5.2. Backfilling technique (EG-SJF)

To improve the performance of job scheduling, we have introduced a gap-filling technique called backfilling. Various types of backfilling have been introduced. In our experiment, we have used the Earliest Gap-Shortest Job first (EG-SJF) scheduling technique that is specified in *Algorithm 2*.

Algorithm 2: EG-SJF Scheduling technique

```
1: for Number of gaps in PU do
2:    if machine_i is suitable to perform job then
3:       Initialqueue = Stores all jobs according to FCFS ;
4:       Readyqueue = Perform SJF to sort the jobs;
5:       Schedule = Place job into found gap [EG strategy];
6:    end if
7: end for
```

Step 1 and 2: Based on the number of gaps present in the processing units, it may be inferred that the loop will persist indefinitely. For example, in the case when the number of gaps is five, the loop will iterate a total of five times. Subsequently, it is necessary to assess the

capabilities of the equipment. Once the machine is prepared to carry out the assigned tasks, the scheduling procedure will proceed.

Step 3 to 5: According to the FCFS scheduling policy, the *initialqueue* serves as the principal repository for all jobs that have been submitted for backfilling. Subsequently, the aforementioned items are arranged in accordance with the SJF algorithm and subsequently stored within the *readyqueue*. Ultimately, the allocation of resources for all tasks is determined based on the Earliest Gap (EG) scheduling methodology.

5.3. Proposed priority algorithm

The fulfillment of service requests on the cloud computing platform is contingent upon meeting a reasonable period, commonly known as the response time. Alternatively, if the cloud system fails to accept the service, it will result in the classification of the accompanying system as being unreliable. Hence, in order to provide reliable service, it is necessary to employ a prioritization-based scheduling algorithm that can efficiently process service requests within the shortest possible timeframe. The proposed PBFS algorithm aims to guarantee the provision of dependable service. This algorithm incorporates considerations of CPU time, job arrival time, and work length. Within the realm of computing, the term "CPU time" pertains to the duration during which a central processing unit (CPU) engages in the processing of data. This metric serves as a means to assess the level of computational intensity exhibited by a given process or program. The CPU time, denoted as CPU(j_i, vm_i), can be represented when the job j_i is scheduled on the ith virtual machine vm_i. The CPU time CPU(j_i, vm_i) can be expressed as

$$\begin{aligned} &CPU\left(j_i, vm_i\right) = FT\left(j_i, vm_i\right) - WT\left(j_i\right) \\ &= FT\left(j_i, vm_i\right) - \left(AT\left(j_i\right) - \mathrm{EST}\left(j_i, vm_i\right)\right) \\ &= \left(FT\left(j_i, vm_i\right) - AT\left(j_i\right)\right) + \mathrm{EST}\left(j_i, vm_i\right) \\ &= \mathrm{TAT}\left(j_i, vm_i\right) + \mathrm{EST}\left(j_i, vm_i\right) \end{aligned} \tag{9}$$

The job will be given priority execution in the VM if the CPU time is less. However, when the CPU time for two jobs is the same, it is necessary to verify the time at which each work arrives. SJF is employed in this step [50]. SJF's pseudocode is defined by *Algorithm 3*.

Algorithm 3: Pseudocode of FCFS

```
Input: Cloud Job request set;
Output: Sort list of requested jobs;
for Number of jobs in the queue do
    Queue ← sort (j_i,j_(i − 1))
end for
return Queue
```

When the first two criteria match for two or more jobs, the third condition is met. The determination of a job's length is possible when there are at least two jobs with identical CPU time and arrival time. SJF [51] is employed in this step. SJF's pseudocode is defined by *Algorithm 4*.

Algorithm 4: Pseudocode of SJF

```
Input: Job request set;
Output: Sort list of requested jobs;
for i=0 to i< main queue size do
    if job_(i+1) length < job_i length then
        add job_(i+1) before of job_i in the queue;
    end if
end for
if main queue size = 0 then
    add job_i to the end of the main queue;
end if
```

SJF, FCFS, and LJF have limitations that we have proposed a new algorithm to address, and we call it PBFS. *Algorithm 5* presents the PBFS algorithm's pseudocode. To begin, we set up the cloud system's VMs, including their storage capacity, power consumption, latency, running duration, reliability threshold, and available time.

Algorithm 5: Pseudo-code of PBFS algorithm

```
1: Input: CLientt Job request;
2: Output: Ordered Job queue;
3: for All Requested Jobs do
4:    for Size of Waiting Queue do
5:       if J_i CPU time < J_(i+1) CPU time then
6:          Placed in The Final Queue;
7:       else if J_i CPU time >= J_(i+1) CPU time then
8:          if J_i AT < J_(i+1) AT then
9:             Placed in The Final Queue;
10:         else if J_i AT >= J_(i+1) AT then
11:            if J_i length < J_(i+1) length then
12:               Placed in The Final Queue;
13:            else
14:               Placed in Temporary Queue;
15:            end if
16:         end if
17:      end if
18:   end for
19: end for
20: if Temporary Queue ≠ 0 then
21:    for Jobs in Temporary Queue do
22:       Placed in The Final Queue;
23:    end for
24: end if
```

Step 1 to 7: The loop will persist for as long as there are jobs waiting to be processed. The initial step of the algorithm is to find the minimum amount of processing time needed. If the job's CPU time is shorter than the preceding one, it will be placed in the final queue for research allocation.

Step 8 to 10: Once the CPU time for both jobs is equal, it is now appropriate to proceed with comparing the arrival times of the jobs. The job that holds the highest priority in the queue will be executed before any other jobs. The job will be allocated to the final queue based on the FCFS scheduling policy.

Step 11 to 16: If both jobs require the same arrival time, then the time has come to evaluate the duration of each position. The shortest line will be served first, while the longest will be moved to the back of the line. If the work durations are similar, however, the temporary queue can accommodate two jobs.

Step 21 to 25: The length of the temporary queue is being calculated at this time. If there are items in the list, the loop will run indefinitely. All jobs from the interim list will be moved to the final queue.

5.4. Time complexity

This PBFS method has a time complexity that may be stated in terms of the number of services n, and the number of virtual machines m, that are hosted within the cloud system. The following PBFS algorithm's time complexity is broken down as follows in the following analysis. In Step-0 of the method, all of the parameters of the virtual machines that make up the cloud systems are set to the value O(m). Steps 3 through 5 make an attempt to compute the CPU time, which can be accomplished in the allotted amount of time O (x). The time complexity of steps 6 to 15 is O(x+y+z), meaning that they take an exponential amount of time to complete. These steps compute the task arrival time as well as the job length. In this case, x denotes the outer else if condition, y the inner if else condition, and z the determiner for the last inner condition. The amount of time complexity is O(n) from step 19 to step 22. In the PBFS method that we have described, Steps 1 and 2 have two loops, each of

which can be completed in time n^2. Therefore, the PBFS method overall time complexity is O(n^2).

5.5. Performance metrics

In this study, we compare three current algorithms using three performance metrics—total tardiness, makespan time, and flowtime—to assess the effectiveness of the proposed schedulers. The following sections provide extensive explanations of these measurements.

Total Tardiness Metric: Successful completion of all orders before the agreed-upon due dates is one of the main goals of scheduling technology. If the service provider cannot deliver on this promise, it negatively impacts its credibility. The degree to which a project is late can be measured by comparing the date of completion, denoted by C_j, with the due date, denoted by d_j. This measure is called the degree of work delay and can be calculated using the following formula.

$$T_j = \max\left(0, C_j - d_j\right) \tag{10}$$

In contrast, the formula for total delay for a group of n jobs processed individually by a single machine is as follows:

$$T = \sum_{j=1}^{n} \max\left(0, C_j - d_j\right)$$
$$C_j = S_j + p_j \tag{11}$$

Where C_j is the completion time of job j in machine m and S_j is its start time. For each job j there is a processing time p_j and a due date d_j. It is difficult to solve the total tardiness problem, especially for large values of n.

Makespan Time: A common performance indicator for evaluating planning algorithms is the so-called makespan). A low value of makespan indicates that the scheduler is effectively and efficiently scheduling the workload of resources. The makespan of a schedule is the time that elapses between the start of the execution of the first task and the end of the execution of the last task. Makespan can be represented as follows:

$$C_{\max} = \max(1 \le j \le nC_j) \tag{12}$$

Simply put, makespan is the time it takes to complete the last task.

Flowtime: The total time required to complete each operation is denoted by the letter C_j and is the Flowtime, also known as the response time. The concept of flowtime can be expressed by the following mathematical formula:

$$F = \sum_{j=1}^{n} C_j \tag{13}$$

When conducting scheduling studies, researchers usually focus on reducing Flowtime and Makespan as two of their main objectives. On the other hand, reducing the makespan to an absolute minimum inevitably leads to an increase in response time.

6. Performance evaluation

In the next sections, we compare our proposed algorithm PR PBFS with LJF, FCFS, and MAX–MIN. The LJF tries to rank a task with a long duration. The FCFS algorithm is given a scheduling priority on a FCFS. The MAX–MIN algorithm attempts to schedule a task with the longest duration first, followed by a task with the shortest duration. Total delay, makespan time, and flowtime were selected as the key performance metrics for evaluating performance.

This section begins with a discussion of the experimental setup, including the data center, virtual machine, and host. It then evaluates the PBFS method and compares it to four other existing algorithms.

Table 3
CloudSim system Configuration.

Architecture	X86
Processing Cost (PC)	3.0
Operating System (OS)	Linux
Memory Cost (MC)	0.05
Storage Cost (SC)	0.1
Time Zone	10
Bandwidth Cost (BC)	0.1

6.1. Environmental setup

To measure the performance of the proposed PBFS algorithm, we create a large discrete event simulator. We coded it in Java and used the Cloud Simulation Toolkit (CloudSim) to run it. Additionally, "Datacenter" is one of the primary classes that is utilized while modeling the cloud. It is a resource in the cloud that features a virtualized host list. The "Processing Element" (PE) class is another type that may be utilized. This class represents the central processing unit, which is measured in terms of millions of instructions carried out each second (MIPS). The CloudSim was set up with three data centers, each of which had between three and five physical resources, also known as "hosts", as indicated in *Table* 3.

Table 4 provides an overview of the host's specifications. Five to thirty virtual machines (VMs) reside on each of the hosts. Memory and storage capacity is measured in megabytes (MBytes), bandwidth is measured in megabits per second (Mbps), and the number of cores indicates how many processor elements are in the host computer. Each processor element has the same amount of processing power. Consequently, cloudlets (and thus VMs) must be allocated to hosts using work scheduling mechanisms to maximize the use of available resources. Poisson distribution is used to send cloudlets to service providers. Custom job scheduler rules, such as those provided in this work, can be tested on this platform, which compares them to other job schedulers.

The job creation component is responsible for implementing user actions by creating jobs of various sizes during the scheduling period. It is well known that such activities require a multithreaded component to run concurrently. Therefore, in this prototype, a multithreaded component is used to create jobs and submit them to the cloud system simultaneously. *Table* 5 lists the number and requirements of the cloudlet's component used in this experiment.

6.2. Simulation result

These tests compare the proposed PBFS algorithm with cloud services on the first test platform, which were randomly generated. We considered three parameters, including flow time, makespan time, and delay time, to evaluate the performance of our proposed approach. We included 50, 100, and 200 jobs in our simulation to better compare PBFS with the other three algorithms.

Fig. 4 presents a comparison of the flow times for each of the four algorithms when 50 jobs and 5 VMs are taken into account. Taking a closer look at this graph, we see with the increase in job load, the flow time is higher. The LJF method has the highest ratio among the four different algorithms. The value, in terms of the 50-number job, is quite close to 600 s. The FCFS has a maximum time of around 450 s for the 50 jobs, which is the second highest. The minimum time taken by the proposed algorithm (PBFS), which defines the best performance of this algorithm.

To compare flowtime, we considered 100 jobs as shown in *Fig.* 5. It can be seen that PBFS performs significantly better overall than the other three algorithms. However, the performance of LJF and FCFS is low and remarkably close. The performance of MAX–MIN is also strong.

In *Fig.* 6, we considered a total of 200 jobs. This graph shows that the difference in flow time is much larger for PBFS than for the other

Table 4
Configured Resources in CloudSim.

Datacenter	Host					
	Name of Host	RAM (MB)	MIPS	Bandwidth (Mbps)	ROM (MB)	Cores
Datacenter_0	Host_I	4096	1000	10000	500,000	1
	Host_II	1024	1000	10000	50,000	3
	Host_III	2048	2000	5000	1,000,000	2
Datacenter_1	Host_I	2048	4200	500	1,500,000	1
	Host_II	4096	7100	5000	50,000	2
	Host_III	2048	12,100	1000	1,500,000	3
Datacenter_2	Host_I	2048	5000	1000	500,000	1
	Host_II	1024	9495	1000	50,000	2
	Host_III	8192	11,900	1000	1,000,000	3
	Host_IV	4096	8500	1000	1,500,000	4

Table 5
Details about number of cloudlet and required resources.

cloudlet number	50/100/200
Output Size	300
Length	Heterogeneous
Input Size	300
Processing number	1

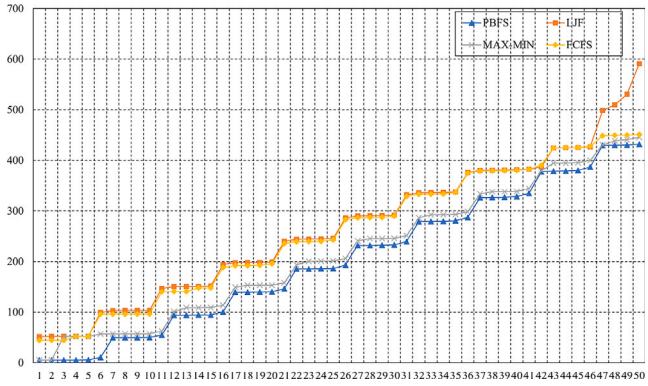

Fig. 4. Flow time comparison for 50 jobs.

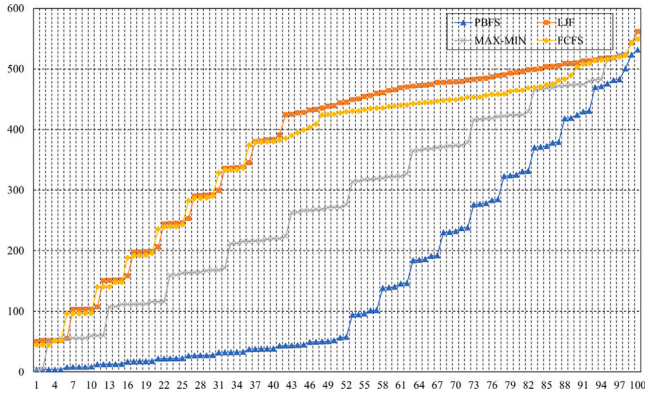

Fig. 5. Flow time comparison for 100 jobs.

three algorithms. FCFS performs well up to 106 jobs, but after that the waiting time increases significantly. performance is the worst for MAX–MIN and LJF.

Figs. 7, 8, *and* 9 define the makespan time for 50, 100 and 200 jobs, respectively, and consider the iterations 10, 20, and 30, respectively. *Figs.* 7,8,9 show that with the increased number of iterations, makespan time decreases gradually. *Fig.* 7 demonstrates that the makespan time for PBFS during the first iteration is around 2000 s; however, as the number of iterations increases, this time decreases to less than 500 s.

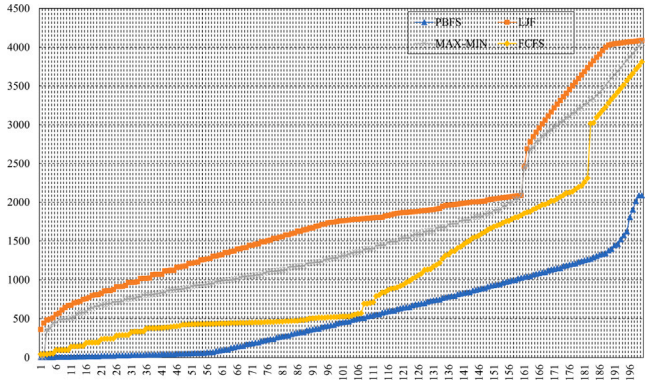

Fig. 6. Flow time comparison for 200 jobs.

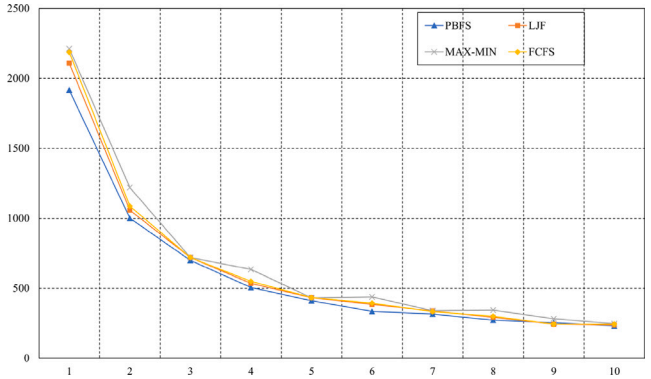

Fig. 7. Makespan time comparison for 50 jobs.

Within the context of the whole period, the performance of PBFS is at its highest point wherever MAX–MIN is at its lowest point. *Fig.* 8 shows that the makespan time for PBFS was more than 1500 s at the beginning of the iteration but decreased to about 250 s after 20 iterations. Although the FCFS performance was poor at the beginning, it improved steadily over time. From *Fig.* 9, it can be seen that the performance of the PBFS algorithm was better than the other three algorithms in the first iteration. However, in the seventh to fourteenth iterations, the performance of each method was essentially identical. Thereafter, an improvement is made to the PBFS algorithm.

Fig. 10 shows the total amount of delays for 50 different jobs. In the beginning, when there were fewer jobs to complete, the delay rate was lower for PBFS. However, this number has increased over time. After completing 40 jobs, the performance of the LJF and FCFS algorithms is improved. But overall, PBFS is much better than the other three.

On the other hand, at 100 jobs, the PBFS tardiness time is good for half of the jobs is shown in *Fig.* 11. However, at point 52, the time is

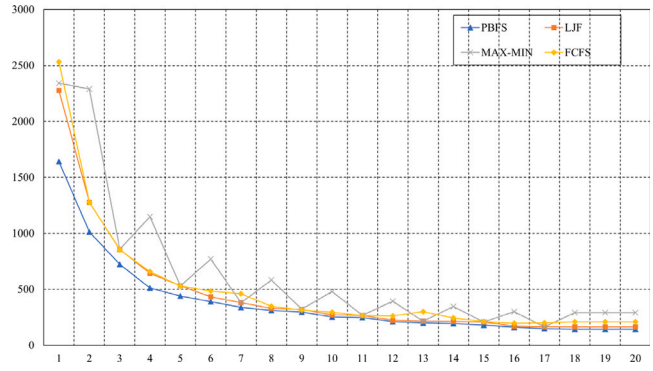

Fig. 8. Makespan time comparison for 100 jobs.

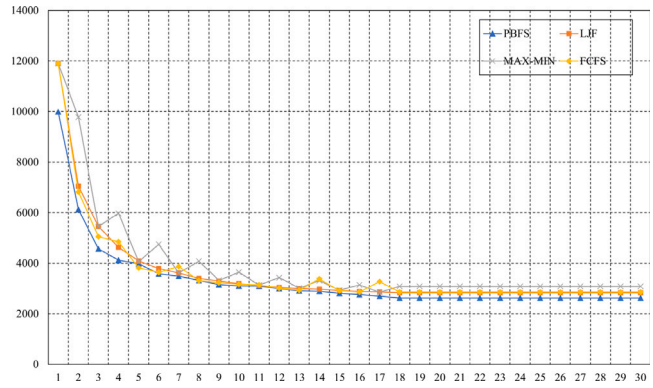

Fig. 9. Makespan time comparison for 200 jobs.

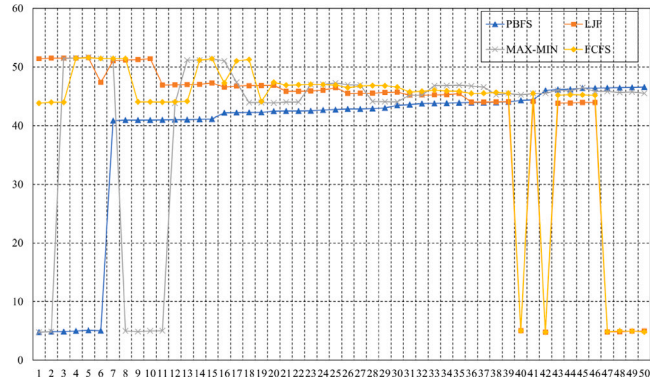

Fig. 10. Total tardiness comparison for 50 jobs.

drastically increased. After that, the time increases gradually. For the other algorithms, the tardiness time fluctuates.

The performance of the four different algorithms is identical when applied to 200 jobs, as shown in *Fig.* 12. However, the tardiness time for PBFS increases slightly once the 180 jobs are completed. For the algorithm MAX–MIN, the total delay increases significantly between numbers 161 and 166. This also applies to FCFS between the numbers 181 and 186.

7. Conclusion and future work

This paper addresses the complex issue of job scheduling in environments with constrained resources. Our initial step involved the development of a resource management framework to rigorously evaluate the reliability of cloud service operations. Building upon this

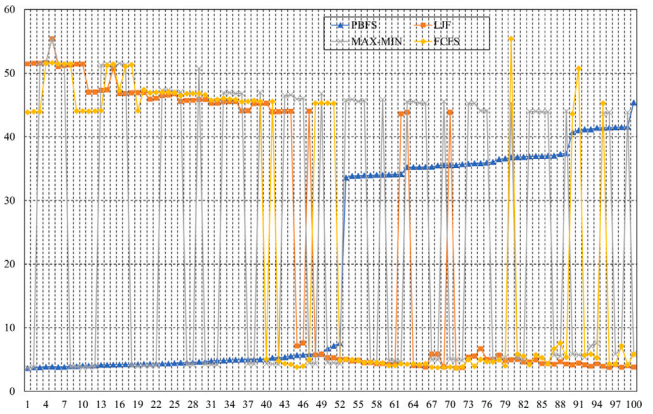

Fig. 11. Total tardiness comparison for 100 jobs.

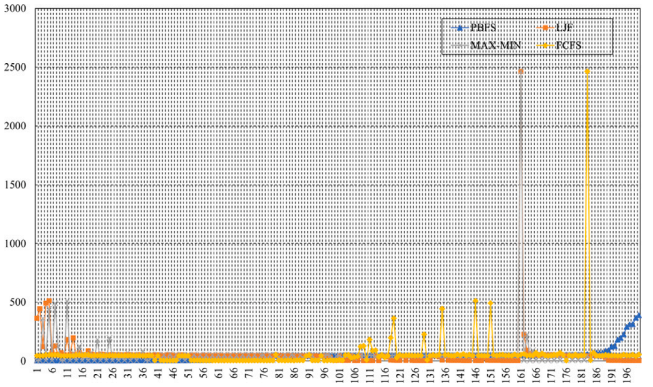

Fig. 12. Total tardiness comparison for 200 jobs.

foundation, we then introduced the Priority-Based Fair Scheduling (PBFS) algorithm, a novel job scheduling technique. The PBFS algorithm uniquely considers three critical parameters: CPU utilization time, the timing of job arrivals, and the individual lengths of the jobs. Further innovating in this domain, we proposed an enhanced backfilling technique termed EG-PBFS (Earliest Gap Priority-Based Fair Scheduling). This technique specifically aims to optimize scheduling efficiency by identifying and utilizing the earliest available time slots in the scheduling queue. To comprehensively assess the efficacy of PBFS, we conducted a comparative analysis with existing Priority Reduction (PR) techniques. This analysis focused on identifying both the strengths and limitations of our proposed algorithm. Performance metrics were pivotal in this evaluation, encompassing aspects such as makespan time (the total time taken to complete all scheduled jobs), flowtime (the aggregate waiting time of all jobs), and the overall delay experienced in the job processing. Our experimental results reveal a significant enhancement in scheduling efficiency with the implementation of the PBFS algorithm. Compared to three other prevalent algorithms in the domain, PBFS demonstrated superior performance across all assessed metrics, underscoring its effectiveness in managing job scheduling in resource-limited cloud environments.

Future studies on priority-based scheduling techniques, backfilling, and reliability can be conducted in the following four directions:

1. In our proposed method, three parameters are taken into account; however, for job scheduling in cloud computing, there are a variety of parameters that can be taken into account.
2. Information such as the estimated processing time is crucial in backfilling schedulers, as underestimation can lead to premature termination, while overestimation can result in long waiting times and the possibility of excessive CPU quota loss.

In this work, job processing times are based on actual estimates. Previous work, such as [52], suggests that overestimation can be beneficial because it creates opportunities for backfilling. A study of the overestimated and underestimated runtime of jobs for the scheduler can be done for future work.

3. With an increase in energy usage, the reliability of the cloud system decreases. Looking at the energy consumption of cloud systems will be an interesting endeavor.
4. The use of different platforms such as Microsoft Azure or Google Cloud is a really fascinating comparative study, which we intend to expand in the future.

CRediT authorship contribution statement

Saydul Akbar Murad: Writing – original draft. **Zafril Rizal M. Azmi:** Supervision. **Abu Jafar Md. Muzahid:** Methodology. **Md. Murad Hossain Sarker:** Conceptualization, Visualization. **M. Saef Ullah Miah:** Data curation, Formal analysis. **MD. Khairul Bashar Bhuiyan:** Visualization. **Nick Rahimi:** Writing – review & editing. **Anupam Kumar Bairagi:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] G.L. Stavrinides, H.D. Karatza, An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations, Future Gener. Comput. Syst. 96 (2019) 216–226.

[2] K. Gu, N. Wu, B. Yin, W. Jia, Secure data sequence query framework based on multiple fogs, IEEE Trans. Emerg. Top. Comput. 9 (4) (2019) 1883–1900.

[3] O.A. Alzubi, J.A. Alzubi, M. Alazab, A. Alrabea, A. Awajan, I. Qiqieh, Optimized machine learning-based intrusion detection system for fog and edge computing environment, Electronics 11 (19) (2022) 3007.

[4] K. Ramkumar, G. Gunasekaran, Preserving security using crisscross AES and FCFS scheduling in cloud computing, Int. J. Adv. Intell. Paradigms 12 (1–2) (2019) 77–85.

[5] M. Hamayun, H. Khurshid, An optimized shortest job first scheduling algorithm for CPU scheduling, J. Appl. Environ. Biol. Sci 5 (12) (2015) 42–46.

[6] A.R. Oskooei, M. Mirza-Aghatabar, S. Khorsandi, Introduction of novel rule based algorithms for scheduling in grid computing systems, in: 2008 Second Asia International Conference on Modelling & Simulation, AMS, IEEE, 2008, pp. 138–143.

[7] T.C.E. Cheng, H.G. Kahlbacher, A proof for the longest-job-first policy in one-machine scheduling, Nav. Res. Logist. 38 (5) (1991) 715–720.

[8] O. Dakkak, S.A. Nor, S. Arif, Scheduling through backfilling technique for HPC applications in grid computing environment, in: 2016 IEEE Conference on Open Systems, ICOS, IEEE, 2016, pp. 30–35.

[9] A. Jain, R. Gupta, Gaussian filter threshold modulation for filtering flat and texture area of an image, in: 2015 International Conference on Advances in Computer Engineering and Applications, IEEE, 2015, pp. 760–763.

[10] M. Grami, An energy-aware scheduling of dynamic workflows using big data similarity statistical analysis in cloud computing, J. Supercomput. 78 (3) (2022) 4261–4289.

[11] S. Long, W. Long, Z. Li, K. Li, Y. Xia, Z. Tang, A game-based approach for cost-aware task assignment with QoS constraint in collaborative edge and cloud environments, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2020) 1629–1640.

[12] M. Eisa, M. Younas, K. Basu, I. Awan, Modelling and simulation of QoS-aware service selection in cloud computing, Simul. Model. Pract. Theory 103 (2020) 102108.

[13] M. Jawad, M.B. Qureshi, M.U. Khan, S.M. Ali, A. Mehmood, B. Khan, X. Wang, S.U. Khan, A robust optimization technique for energy cost minimization of cloud data centers, IEEE Trans. Cloud Comput. 9 (2) (2018) 447–460.

[14] Z. Xia, Y. Zhu, X. Sun, Z. Qin, K. Ren, Towards privacy-preserving content-based image retrieval in cloud computing, IEEE Trans. Cloud Comput. 6 (1) (2015) 276–286.

[15] N.P. Sodinapalli, S. Kulkarni, N.A. Sharief, P. Venkatareddy, An efficient resource utilization technique for scheduling scientific workload in cloud computing environment, IAES Int. J. Artif. Intell. 11 (1) (2022) 367.

[16] A.A. Movassagh, J.A. Alzubi, M. Gheisari, M. Rahimi, S. Mohan, A.A. Abbasi, N. Nabipour, Artificial neural networks training algorithm integrating invasive weed optimization with differential evolutionary model, J. Ambient Intell. Humaniz. Comput. (2021) 1–9.

[17] F. Cheng, Y. Huang, B. Tanpure, P. Sawalani, L. Cheng, C. Liu, Cost-aware job scheduling for cloud instances using deep reinforcement learning, Cluster Comput. 25 (1) (2022) 619–631.

[18] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R.N. Chang, M.R. Lyu, R. Buyya, Cloud service reliability enhancement via virtual machine placement optimization, IEEE Trans. Serv. Comput. 10 (6) (2016) 902–913.

[19] A. Zhou, S. Wang, Q. Sun, H. Zou, F. Yang, FTCloudSim: A simulation tool for cloud service reliability enhancement mechanisms, in: Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference, 2013, pp. 1–2.

[20] S.K. Mondal, F. Machida, J.K. Muppala, Service reliability enhancement in cloud by checkpointing and replication, in: Principles of Performance and Reliability Modeling and Evaluation, Springer, 2016, pp. 425–448.

[21] X.-Y. Li, Y. Liu, Y.-H. Lin, L.-H. Xiao, E. Zio, R. Kang, A generalized petri net-based modeling framework for service reliability evaluation and management of cloud data centers, Reliab. Eng. Syst. Saf. 207 (2021) 107381.

[22] F. Tao, L. Zhang, Y. Liu, Y. Cheng, L. Wang, X. Xu, Manufacturing service management in cloud manufacturing: Overview and future research directions, J. Manuf. Sci. Eng. 137 (4) (2015).

[23] A.P. Raveendran, J.A. Alzubi, R. Sekaran, M. Ramachandran, A high performance scalable fuzzy based modified asymmetric heterogene multiprocessor system on chip (AHt-MPSOC) reconfigurable architecture, J. Intell. Fuzzy Systems 42 (2) (2022) 647–658.

[24] F. Machida, M. Kawato, Y. Maeno, Redundant virtual machine placement for fault-tolerant consolidated server clusters, in: 2010 IEEE Network Operations and Management Symposium-NOMS 2010, IEEE, 2010, pp. 32–39.

[25] S.C. Nayak, S. Parida, C. Tripathy, P.K. Pattnaik, Dynamic backfilling algorithm to increase resource utilization in cloud computing, Int. J. Inf. Technol. Web Eng. (IJITWE) 14 (1) (2019) 1–26.

[26] M. Hasan, M.S. Goraya, Priority based cooperative computing in cloud using task backfilling, Lect. Notes Softw. Eng. 4 (3) (2016) 229–233.

[27] J. Natarajan, Parallel queue scheduling in dynamic cloud environment using backfilling algorithm, Int. J. Intell. Eng. Syst. 11 (2) (2018) 39–48.

[28] S.M. John, M. Mohamed, Novel backfilling technique with deadlock avoidance and migration for grid workflow scheduling, Indian J. Sci. Technol. 8 (12) (2015) 1.

[29] L. Sheikhani, H. Lu, C. Gu, Priority-based scheduling approach to minimize the SLA violations in cloud environment, in: 2021 7th International Conference on Computer and Communications, ICCC, IEEE, 2021, pp. 1449–1457.

[30] S. Ghanbari, M. Othman, A priority based job scheduling algorithm in cloud computing, Procedia Eng. 50 (2012) 778–785.

[31] L. Ruan, Y. Yan, S. Guo, F. Wen, X. Qiu, Priority-based residential energy management with collaborative edge and cloud computing, IEEE Trans. Ind. Inform. 16 (3) (2019) 1848–1857.

[32] S.J. Patel, U.R. Bhoi, Improved priority based job scheduling algorithm in cloud computing using iterative method, in: 2014 Fourth International Conference on Advances in Computing and Communications, IEEE, 2014, pp. 199–202.

[33] I. Gupta, M.S. Kumar, P.K. Jana, Efficient workflow scheduling algorithm for cloud computing system: A dynamic priority-based approach, Arab. J. Sci. Eng. 43 (12) (2018) 7945–7960.

[34] S.A. Murad, Z.R.M. Azmi, A.J.M. Muzahid, M. Al-Imran, Comparative study on job scheduling using priority rule and machine learning, in: 2021 Emerging Technology in Computing, Communication and Electronics, ETCCE, IEEE, 2021, pp. 1–8.

[35] Y. Lian, Y. Tang, Y. Wang, Objective Bayesian analysis of JM model in software reliability, Comput. Statist. Data Anal. 109 (2017) 199–214.

[36] S.A. Murad, A.J.M. Muzahid, Z.R.M. Azmi, M.I. Hoque, M. Kowsher, A review on job scheduling technique in cloud computing and priority rule based intelligent framework, J. King Saud Univ.-Comput. Inf. Sci. (2022).

[37] Y. Sharma, W. Si, D. Sun, B. Javadi, Failure-aware energy-efficient VM consolidation in cloud computing systems, Future Gener. Comput. Syst. 94 (2019) 620–633.

[38] Y.-C. Lu, C.-K. Lin, K.-C. Lai, M.-H. Tsai, Y.-J. Wu, H.-Y. Chang, K.-C. Huang, Service deployment and scheduling for improving performance of composite cloud services, Comput. Electr. Eng. 74 (2019) 616–634.

[39] M. O'Connor, Cloud authentication services security for enterprise IT, 2021, 1Kosmos, URL https://www.1kosmos.com/authentication/cloud-authentication-services/.

[40] H. Mikailu, H. Bello, L. Mathias, An efficient resource selection and allocation in cloud computing using artificial nutrients distribution model, Fudma J. Sci. 4 (3) (2020) 721–730.

[41] M. Githiru, L. Lens, F. Adriaensen, J. Mwang'ombe, E. Matthysen, Using science to guide conservation: From landscape modelling to increased connectivity in the Taita Hills, SE Kenya, J. Nat. Conserv. 19 (5) (2011) 263–268.

[42] Z. Wen, J. Cała, P. Watson, A. Romanovsky, Cost effective, reliable and secure workflow deployment over federated clouds, IEEE Trans. Serv. Comput. 10 (6) (2016) 929–941.
[43] P. Jogalekar, M. Woodside, Evaluating the scalability of distributed systems, IEEE Trans. Parallel Distrib. Syst. 11 (6) (2000) 589–603.
[44] J. Wang, C. Zhang, Software reliability prediction using a deep learning model based on the RNN encoder–decoder, Reliab. Eng. Syst. Saf. 170 (2018) 73–82.
[45] J.A. Alzubi, O.A. Alzubi, A. Singh, T. Mahmod Alzubi, A blockchain-enabled security management framework for mobile edge computing, Int. J. Netw. Manag. 33 (5) (2023) e2240.
[46] Z. Li, L. Liu, D. Kong, Virtual machine failure prediction method based on AdaBoost-hidden Markov model, in: 2019 International Conference on Intelligent Transportation, Big Data & Smart City, ICITBS, IEEE, 2019, pp. 700–703.
[47] Y.-S. Dai, B. Yang, J. Dongarra, G. Zhang, Cloud service reliability: Modeling and analysis, in: 15th IEEE Pacific Rim International Symposium on Dependable Computing, Citeseer, 2009, pp. 1–17.
[48] S.A. Murad, Z.R.M. Azmi, A.J.M. Muzahid, M.K.B. Bhuiyan, M. Saib, N. Rahimi, N.J. Prottasha, A.K. Bairagi, SG-PBFS: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment, Future Gener. Comput. Syst. 150 (2024) 232–242.
[49] S.A. Murad, Z.R.M. Azmi, F.J. Brishti, M. Saib, A.K. Bairagi, Priority based fair scheduling: Enhancing efficiency in cloud job distribution, in: 2023 IEEE 8th International Conference on Software Engineering and Computer Systems, ICSECS, IEEE, 2023, pp. 170–175.
[50] W. Zhao, J.A. Stankovic, Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems, in: 1989 Real-Time Systems Symposium, IEEE Computer Society, 1989, pp. 156–157.
[51] S.S. Prabhu, H. Kapil, S.H. Lakshmaiah, Safety critical embedded software: Significance and approach to reliability, in: 2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI, IEEE, 2018, pp. 449–455.
[52] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, Characterization of backfilling strategies for parallel job scheduling, in: Proceedings. International Conference on Parallel Processing Workshop, IEEE, 2002, pp. 514–519.