Contents lists available at ScienceDirect

Future Generation Computer Systems

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

SG-PBFS: Shortest Gap-Priority Based Fair Scheduling technique for job scheduling in cloud environment

Saydul Akbar Murad [a,d], Zafril Rizal M. Azmi [a,*], Abu Jafar Md Muzahid [a], MD. Khairul Bashar Bhuiyan [b], Md Saib [c], Nick Rahimi [d], Nusrat Jahan Prottasha [e], Anupam Kumar Bairagi [f]

[a] *Faculty of Computing, College of Computing & Applied Sciences, Universiti Malaysia Pahang, Pekan, Pahang, 26600, Malaysia*
[b] *Brac University, Dhaka, 1212, Bangladesh*
[c] *South China University of Technology, Guangzhou, 510006, China*
[d] *School of Computing Sciences & Computer Engineering, University of Southern Mississippi, Hattiesburg, 39401, USA*
[e] *Stevens Institute of Technology, NJ, 07030, USA*
[f] *Khulna University, New Khulna, 9208, Bangladesh*

ARTICLE INFO

Article history:
Received 30 March 2023
Received in revised form 13 August 2023
Accepted 2 September 2023
Available online 7 September 2023

Keywords:
Cloud computing
Job scheduling
Backfilling technique
Gap search
Algorithm
Makespan time
Flow time
FCFS

ABSTRACT

Job scheduling in cloud computing plays a crucial role in optimizing resource utilization and ensuring efficient job allocation. But cloud resources may be wasted, or service performance may suffer because of under-utilization or over-utilization because of poor scheduling. Existing approaches often overlook the dynamic nature of cloud environments, resulting in issues like missed deadlines, prolonged flowtime, extended makespan, and unacceptable total tardiness. To address this issue, the main objective of this research is to improve the existing Priority Rules (PR) cloud schedulers by developing a new dynamic scheduling algorithm by manipulating the gaps in the cloud job schedule. Firstly, a Priority-Based Fair Scheduling (PBFS) algorithm has been introduced to schedule jobs so that jobs can access the required resources at optimal times. Then, a backfilling strategy called Shortest Gap - Priority-Based Fair Scheduling (SG-PBFS) is developed that attempts to manipulate the gaps in the schedule of cloud jobs. Finally, the performance evaluation demonstrates that the proposed SG-PBFS algorithm outperforms SG-SJF, SG-LJF, SG-FCFS, SG-EDF, and SG-(MAX-MIN) regarding flow time, makespan time, and total tardiness, which conclusively demonstrates its effectiveness. To conduct this experiment, we employed the CloudSim simulator, which is implemented using the Java programming language.

1. Introduction

The main goal of cloud computing is to make the resources easy that are far away or in different places. As the cloud environment continues to evolve and face new challenges, scheduling has emerged as a significant issue. The job allocation challenge is defined as scheduling job and resources in a cloud environment such that users can perform their tasks quickly and cheaply, besides boosting user and vendor satisfaction [1,2]. In a cloud context, there is currently no established norm for job scheduling. Distributed system job scheduling has been the subject of numerous in-depth research.

Several performance parameters play a crucial role in determining the advantages of job placement, including metrics such as throughput, flow time, resource utilization, makespan time, and total tardiness. Some scheduling algorithms also pay attention to Quality of service (QoS) parameters. Traditional scheduling techniques have some drawbacks and do not perform well in the cloud. Longest Job First (LJF), First Come-First Serve (FCFS), Shortest Job First (SJF), Round Robin, Max-Min, and Min-Min techniques are widely used for job scheduling. Currently, popular enterprise schedulers, like Maui [3], Sun Grid Engine (SGE) [4], Portable Batch System (PBS) [5], and Load Sharing Facility (LSF) [6] widely use to manage jobs because of their simplicity and flexibility of implementation. Job scheduling prioritization is a critical consideration to ensure the timely completion of each job. Numerous algorithms consider job priorities in order to address the challenge of meeting job deadlines.

The problem with current job scheduling techniques, such as in [7–9], is that they only focus on the power consumption of jobs

* Corresponding author.
E-mail addresses: saydulakbarmurad@gmail.com (S.A. Murad), zafril@ump.edu.my (Z.R.M. Azmi), mrumi98@gmail.com (A.J.M. Muzahid), kbashar707@gmail.com (MD. Khairul Bashar Bhuiyan), mdsaib45@gmail.com (M. Saib), nick.rahimi@usm.edu (N. Rahimi), jahannusratprotta@gmail.com (N.J. Prottasha), anupam@cse.ku.ac.bd (A.K. Bairagi).

https://doi.org/10.1016/j.future.2023.09.005
0167-739X/© 2023 Elsevier B.V. All rights reserved.

in cloud systems but never consider performance matrices like makespan time or flow time. These works used Priority Rule (PR), such as FCFS and SJF, which are proven to be not optimized. The critical performance metrics in cloud computing job scheduling are tardiness time, makespan time, and flow time. Makespan time defines the total time a machine takes to accomplish all jobs. If the makespan time is higher, it indicates the poor performance of this algorithm. Flow time, on the other hand, is the overall time required to go from one operation to the next, including any time spent waiting for equipment or job orders to arrive and any time wasted due to machine breakdowns, process delays, or component shortages. Lower flow time defines the better algorithm's performance. A delayed job indicates that the job was not completed on time due to poor scheduling.

Some researchers in existing research tried to reduce the makespan time but do not care about flow time, while others are only concerned with flow time and never mention total tardiness. Therefore, to improve the performance of PR in terms of makespan time, flow time, and total tardiness, the development of a high-performance job scheduler is required. PR schedulers have been the focus of this research because it has already been proven to optimize single performance metrics. Among them, SJF is the most powerful in minimizing flow time [10], LJF is effective in reducing makespan time [11], Earliest Deadline First (EDF) minimizes the total number of jobs that surpass the deadline [12], Mean Time to Detect (MTTD) reduces tardiness [13], and the popular FCFS guarantees the fairness of all jobs to be processed based on their seniority in the queue [14].

However, more performance metrics must be considered in a cloud environment to meet users requirements, as users expect their jobs to be finished before the deadline. Developing a new scheduling strategy is essential for achieving multiple objectives and ensuring user satisfaction. Previous research predominantly focused on single performance metrics, but now it is important for a competent scheduler to take into account multiple performance metrics. In order to validate the efficacy and efficiency of the proposed scheduler in this paper, it is essential to conduct a thorough study to ensure that the correct and optimal performance metric has been integrated into the scheduler. By doing so, a justification can be made that the scheduler has reached its optimum capability.

Another essential aspect of dynamic-based schedulers is the gap-filling or backfilling technique. Gap-filling is only a part of the big picture in the scheduling process. It is crucial because, in the dynamic nature of the cloud, there are possibilities for resources to change their states due to many reasons. The backfilling process handles this change by modifying the existing schedule. In addition, backfilling also acts as an optimizer for the current schedule generated by the initial scheduler. There are many techniques used in the backfilling process. Earliest Gap - Earliest deadline first (EG-EDF) [15] and Earliest Gap - First Come, First Served (EG-FCFS) [16] are the most popular backfilling scheduling algorithms. Since all other PR algorithms are also queue-based, the possibility of extending the backfilling technique for PR algorithms other than FCFS is high. Furthermore, the existing backfilling considers only small jobs to be forwarded. This situation is not efficient since larger jobs can fill gaps between jobs. If this limitation can be overcome, fair and high-performance backfilling can be achieved.

To address the aforementioned challenges, this study introduces a Priority-Based Fair Scheduling (PBFS) algorithm and a backfilling technique named Shortest Gap Priority-Based Fair Scheduling (SG-PBFS). The significant contributions of this work are summarized as follows:

- Firstly, we propose a PBFS algorithm to enhance job scheduling efficiency by reducing flow time, makespan time, and total tardiness.
- Secondly, we propose a backfilling technique named SG-PBFS to increase the efficiency of job distribution.
- Finally, we evaluate the performance of the proposed PBFS and SG-PBFS algorithms using the CloudSim simulator with an existing dataset.

This research paper is put together in the following way: In Section 2, a literature review is provided, which provide an overview of existing studies related to the topic. Section 3 describes the research methodology employed to accomplish the objectives of this research. It presents an operational framework for job scheduling, introducing a new priority-based algorithm called PBFS. Additionally, a backfilling technique is proposed to enhance the performance of the existing PR algorithm. Section 4 focuses on the results and discussion, where the performance of the proposed priority algorithm and backfilling technique is evaluated. Lastly, Section 5 concludes the research paper, summarizing the findings and providing insights based on the results obtained.

2. Literature review

2.1. Priority rules scheduler in cloud computing

The most widely used scheduler in a dynamic environment is from the cluster of PR [17]. As stated in [18], PR is crucial for several reasons, especially when project managers do not (or cannot) build a complete network model of their project. Cloud computing is typically an ad-hoc architecture that does not have an exact network model for the entire system. In cloud computing, the scheduler has to deal with multiple simultaneous incoming jobs. In such cases, PR determines which activities and projects should get priority.

In recent years, many have used the PR as an initial schedule for their novel cloud scheduler, such as the usage of FCFS and LJF as a primary scheduler for the Fuzzy Particle Swarm-based scheduling [19], using SJF as a primary scheduler for their Swift Scheduler algorithm [20] and the use of LJF and SJF as an initial schedule for the Genetic Algorithms (GA) based scheduler. Unfortunately, none of these have justified why they chose a particular PR instead of the other. It is crucial to conduct studies on the effect of PR on the overall scheduler performance because different PR will have different impacts on the cloud system. However, before the impact can be evaluated, the performance of each individual in the PR family should be examined first. With a somewhat different orientation, performance comparisons of PR have been covered elsewhere. [21] showed a comparison result between SJF and LJF for job scheduling in cloud computing. This experiment shows that SJF outperforms LJF. But SJF has a number of limitations. One limitation is that, in the short term, SJF cannot be used to schedule CPU time. Because there is no way to know for sure how long the next CPU burst will last.

[22] proposed a Priority Aware Longest Job First (PA-LJF) algorithm to increase the utilization of resources. As LJF executes the longest job first, the other jobs have to wait a long time in the waiting queue, which increases the flow time and total tardiness. [23] used the Earliest Deadline First (EDF) scheduling algorithm when jobs are scheduled in a heterogeneous multiprocessor. But EDF is less consistent due to the fact that job response times vary and are constant when using a rate monotonic or fixed priority algorithm. In particular, [24] have reviewed the PR with a mathematical model, while in [25], a survey based on simulation studies has been conducted. [26] proposed a two-stage planning strategy applicable in a geo-distributed data center environment. In the first stage, the appropriate data center is selected to handle the user request, and in the second stage, the appropriate VM is selected. The goal is to reduce SLA violations in terms

of reliability, response time, and availability. [27] proposed a new algorithm for prioritization-based job scheduling (PJSC) in cloud computing. This proposed method is based on a model of decision-making with numerous criteria.

[28] introduced a three-tier edge-cloud collaborative REM (EC-CREM) architecture in order to increase latency and processing speed. [29] show how a multiple-attribute and multiple-criteria decision-making model can be used to improve the priority-based job scheduling method in cloud computing. [30] provide a job scheduling method that considers dynamic job priorities. The technique performs min–max normalization and then calculates the dynamic threshold to assign jobs to one of the VMs.

In previous works, the focus has predominantly been on mathematical models and service disciplines, such as production scheduling, where mathematical modeling and analysis have shown some success. However, there is a need to extend this investigation to various PR that has not yet been thoroughly analyzed, particularly in the context of cloud computing environments where such analysis is lacking. As an example, the LJF scheduling algorithm is effective in minimizing makespan but performs poorly in terms of flowtime. Conversely, the SJF scheduling algorithm is effective in reducing flowtime but exhibits weak performance in terms of makespan. Additionally, considering that PR is already widely employed in scheduling cloud jobs, enhancing the performance of PR itself through efficient techniques presents an effective and cost-efficient approach to improving the current cloud scheduler.

2.2. Gap filling technique in cloud computing

In the actual implementation of cloud computing, PR has been widely used by Local Resource Management Systems (LRMS) such as Maui, SGE, PBS, and LSF to manage resources and jobs [25]. However, a traditional PR scheduler may not perform well because of the unique characteristics of the job itself. For example, SJF only allows the shortest job in the queue to be processed first while abandoning the longest job. FCFS does not consider anything other than the sequence of the jobs, which results in lower system utilization.

To improve this limitation, the original gap-filling technique has been introduced with the objective of improving the FCFS [31]. It is called backfilling because it allows short jobs to move ahead of long jobs, resulting in better system utilization and throughput [32]. Varieties of backfilling have been introduced by existing research. According to this, [33] explored the application of backfilling and rate monotonic (RM) algorithms for real-time job scheduling in cloud environments. But they never consider any comparison with other existing algorithms, which is a part of this research. [34] investigated the use of multi-criteria decision-making (MCDM) techniques in backfilling algorithms in cloud computing to accomplish deadline-based jobs. They simulate the backfilling technique as well as three MCDM mechanisms in order to avoid scheduling conflicts between activities that are identical. However, in real-world implementation, most of the existing backfilling techniques are based on the FCFS strategy [32,35] and only consider short jobs as candidate jobs to bind a waiting large job.

But the FCFS algorithm is typically not suitable for backfilling techniques in cloud job scheduling. The reason is that FCFS strictly follows the order in which jobs arrive, and once a job is scheduled, it continues execution until completion without interruption. This lack of flexibility prevents backfilling, which is a technique used to optimize resource utilization. FCFS does not allow for this dynamic scheduling approach. Once a job is scheduled, it occupies the resources until it completes, regardless of the availability of resources for other smaller jobs. This can lead to inefficient resource utilization and potentially increased waiting times for jobs, hindering the effectiveness of the backfilling technique.

However, there are also many research studies that combine backfilling with other PR schedulers. For example, in [15], SJF-Backfilling has been successfully implemented. The backfilling works by rearranging the existing job queue based on the increasing order of job execution times. A job eligible for advancement is one that is already present in the queue. Similarly, [36] conducted research that combined SJF with backfilling but in a different way. While preserving the job using FCFS, the SJF only applies to the jobs that want to be moved. These two techniques and several other PR-Backfilling schedulers have one thing in common, which is computationally expensive because the scheduler has to reconstruct a new schedule each time a new job arrives in the system [16,37].

Currently, the most efficient backfilling technique that handles this limitation is the Earliest Gap-Earliest Deadline First (EG-EDF) proposed by [16]. But, there is no strong justification for the use of EDF in EG-EDF. In addition, the EG procedure in EG-EDF that focus on filling the earliest gaps available in the resource schedule might not efficiently utilize gaps. This situation can occur if much smaller jobs are the candidate jobs that have been backfilled in the gaps that are too big for those jobs. This will lead to another gap being created or a resource remaining idle for the rest of the time when no suitable jobs can be fitted into the gap. To overcome those limitations, it is crucial to develop new backfilling techniques that harness the full potential of backfilling schedulers, which favor more powerful PR schedulers. In light of these challenges, we have introduced a novel backfilling technique that incorporates the utilization of the Shortest Gap (SG) approach. This technique is integrated with our proposed Priority-Based Fair Scheduling (PBFS) algorithm to address the aforementioned issues in cloud job scheduling.

3. Methodology

3.1. Operational framework

The operational framework of this work is presented in Fig. 1. Here, the section begins by describing the research design and procedure, which contains two methods. The research design and procedure are started by discussing the proposed priority rule algorithm. The primary concept of this algorithm is fair scheduling based on priority. The backfilling technique is discussed in the second phase of this section. Time complexity is another crucial topic that is discussed after the research design. Finally, we have ended the methodology by discussing the environmental setup.

3.2. Research design and procedure

The research design and procedure consist of two modules, namely PR and backfilling. The first module proposes a novel priority algorithm called PBFS. The other module uses two backfilling techniques named Earliest Gap - PBFS (EG-PBFS) and Shortest Gap - PBFS (SG-PBFS) to utilize the gap. The research design and procedure shown in Fig. 2 can be used to evaluate the interaction between these techniques. Performance problems have been solved by creating the PBFS scheduler and SG-PBFS, particularly when balancing the trade-off between various performance metrics. The design has been split into priority rule and backfilling phases. Users typically start by submitting jobs to the cloud. Then, these jobs are added to the waiting queue on an FCFS basis. The scheduler looks for gaps in the current resource (machine schedule). The job will be placed in the main queue and sorted according to the intended PR if there is no gap or the gap does not

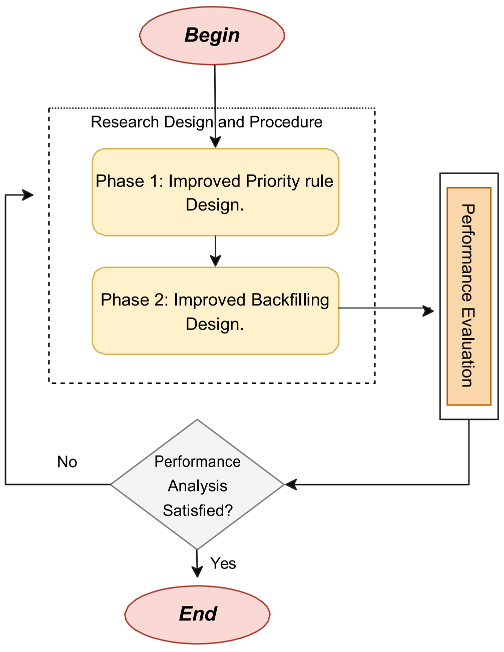

Begin
Research Design and Procedure
Phase 1: Improved Priority rule Design.
Phase 2: Improved Backfilling Design.
Performance Evaluation
Performance Analysis Satisfied?
No
Yes
End

Fig. 1. Operational framework.

Algorithm 1: The proposed PBFS algorithm

```
1:  Input: Cloud Job request set;
2:  Output: Queue of sorted jobs;
3:  Cloud VMs Initialization;
4:  for all submitted jobs in waiting queue do
5:    if $Job_i$ CPU time $< Job_{i+1}$ CPU time then
6:      Stored job in final queue list;
7:    else if $Job_i$ CPU time $>= Job_{i+1}$ CPU time then
8:      if $Job_i$ Arrival time $< Job_{i+1}$ Arrival time then
9:        Stored job in final queue list;
10:     else if $Job_i$ Arrival time $>= Job_{i+1}$ Arrival time then
11:       if $Job_i$ length $< Job_{i+1}$ length then
12:         Stored job in final queue list;
13:       else
14:         Stored job in temporary queue list;
15:       end if
16:     end if
17:   end if
18: end for
19: if temporary list $\neq 0$ then
20:   for jobs in temporary queue list do
21:     Stored job in the final queue list.;
22:   end for
23: end if
```

suit the job. This also means that the head of the main queue will only occasionally be the first job to arrive in the cloud. Backfilling is employed to distribute the jobs into gaps if there is a single gap or more than one available.

3.2.1. Proposed priority rule design in cloud computing

PR is one of the key components of this paper; this module begins by analyzing the performance of PR in a cloud environment. This step is crucial for setting the stage for this paper because the outcomes serve as benchmarks for the suggested schedules. Additionally, as the analysis findings are in line with those from the literature, the results confirm the simulator's reliability. LJF, FCFS, EDF, SJF, and MAX-MIN are five PR algorithms that are implemented in CloudSim, where the code is written in Java. The existence PRs drawback is that it only focuses on one performance parameter while ignoring the others. To overcome the limitations of SJF, FCFS, EDF, max–min, and LJF, a new priority-based job scheduling algorithm is proposed called PBFS. PBFS algorithm is presented in *Algorithm 1*. To begin, in this experiment, the cloud system's virtual machines (VMs) have been set up, including their power consumption, storage capacity, latency, reliability threshold, running duration, and available time.

Traditional PR algorithms, like FCFS, have the reasonably significant disadvantage of long wait times [38]. Once a CPU has been allocated to a job, it will not be released until the job has concluded. This means that if the first job placed has a significant turnaround time, all subsequent jobs must wait for its completion. To overcome the limitation of the existing PR algorithm, the main point of this research is to come up with a priority-based algorithm by combining CPU time, job length, and arrival time and then figuring out how well it works based on three performance (flow time, tardiness time, and makespan time) metrics. Fig. 3 depicts the PR scheduler's general architecture. After users submit jobs to the cloud system, the jobs are routed to a FCFS waiting list, where the first job to arrive will be at the head of the queue. The jobs in the waiting queue are then ordered according to their intended PR schedulers, which may be FCFS, SJF, LJF, EDF, MAX-MIN, or PBFS. This ordered queue is referred to as the primary queue.

Proposed PBFS algorithm. The response time is the amount of time in which a request for a service on the cloud platform must be processed. Otherwise, the cloud system will reject the job, and the accompanying system will be deemed unreliable. Therefore, in order to deliver dependable service, a prioritization-based scheduling algorithm is which can execute the service request in a minimum amount of time. A new priority-based algorithm called PBFS has been presented to ensure reliable service. The CPU time, the job arrival time, and the job length are all taken into consideration when developing this algorithm. CPU time refers to the duration that a computer's central processing unit (CPU) spends on executing computational tasks. It serves as a measure to assess the level of CPU utilization or intensity of a process or program in terms of computational demands. CPU time influences the overall performance of the system. By efficiently scheduling jobs based on their CPU time requirements, the scheduler can minimize job waiting times and reduce system response times. If the job j_i is scheduled on the nth VM (vm_n), the CPU time CPU (j_i, vm_n) can be expressed by Eq. (1). In this experiment, n max value is 30, and i max value 500.

$$\begin{aligned} CPU\ (j_i, vm_n) &= FT\ (j_i, vm_n) - WT\ (j_i) \\ &= FT\ (j_i, vm_n) - (AT\ (j_i) - \mathrm{EST}\ (j_i, vm_n)) \\ &= (FT\ (j_i, vm_n) - AT\ (j_i)) + \mathrm{EST}\ (j_i, vm_n) \\ &= \mathrm{TAT}\ (j_i, vm_n) + \mathrm{EST}\ (j_i, vm_n) \end{aligned} \tag{1}$$

Here, FT(j_i, vm_n) defines the finish time of job j_i in vm_n where WT(j_i) stands for the waiting time of job j_i in the queue. The CPU time is calculated by differentiating the waiting time from the finish time. The waiting time (WT(j_i)) is the differentiation of Arrival time (AT(j_i)) for job j_i and Execution start time (EST(j_i, vm_n)) for job j_i that started execution in vm_n. TAT(j_i, vm_n) stands for Turn around time, which is obtained by dividing the arrival time from the finish time. Finally, the CPU time (j_i, vm_n) is the summation of Turn around time TAT(j_i, vm_n) and execution start time (EST(j_i, vm_n)).

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, the job arrival time has been checked. Job arrival

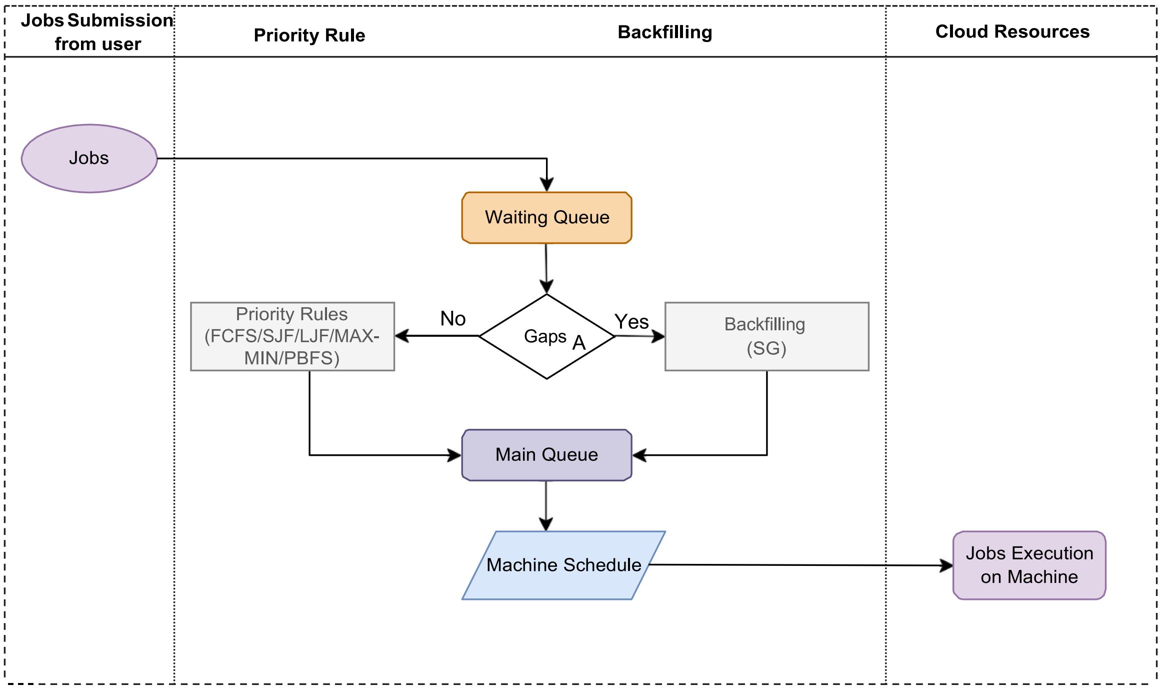


Fig. 2. Research design and procedure.

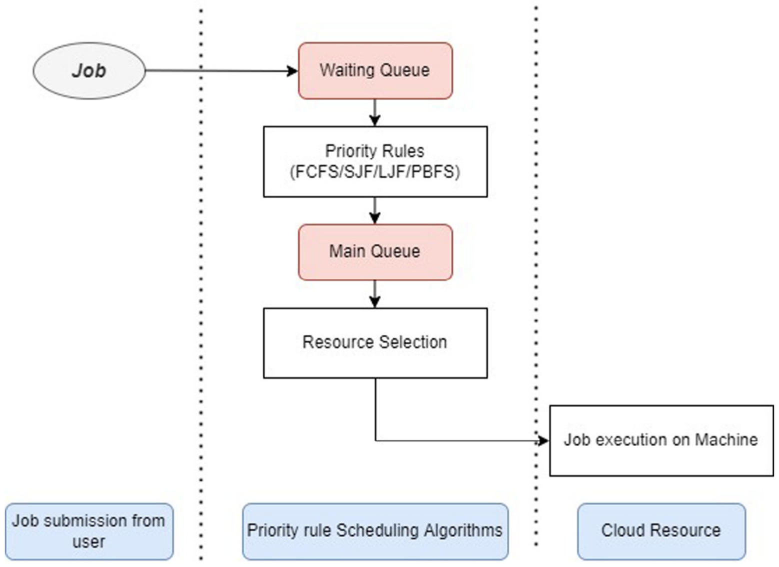


Fig. 3. Architecture of priority rules schedulers.

time is another important parameter for job scheduling that is considered in designing this algorithm. Job arrival time influences resource allocation decisions. By considering the arrival time, the scheduler can allocate resources to jobs efficiently, ensuring optimal utilization of available resources. When the CPU time and arrival time are the same for at least two jobs, the job length can be determined. SJF [39,40] is employed in this step. Job length impacts the overall system throughput and performance. Shorter jobs typically have quicker turnaround times, allowing more jobs to be processed within a given time frame. In this experiment, we have utilized two queues: the final queue and the temporary queue. Jobs that meet any of the three specified conditions are stored in the final queue. Conversely, jobs that fail to meet these conditions are placed in the temporary queue, following a First-Come-First-Served (FCFS) order.

Step 1 to 6: The goal of this work is to identify the most effective technique for each job. CPU time is most important for better performance as it reduces flow and makespan time [41]. For this, in our algorithm, we prioritize the minimization of CPU time as the key parameter. The initial phase involves identifying the job with the shortest CPU time. A job with the shortest CPU time is placed in the final queue, which represents the set of jobs ready for resource allocation.

Step 7 to 9: If the CPU time for both jobs is the same or greater, then the job arrival is compared. The job that came first in the waiting queue will be stored in the final queue. The job will be placed in the final queue, according to the FCFS. The earliest arrival job is given priority for fair scheduling.

Step 10 to 16: If the arrival time for both tasks is the same or greater, then the job length is compared. The one with the smallest length will be given preference in terms of receiving service, and it will be placed in the final queue. However, if the job lengths are equal, then the temporary queue will hold two jobs. To reduce the total tardiness the smallest length is given priority in this research.

Step 20 to 23: Here, the size of the temporary queue list is being evaluated. If the list contains jobs, the loop will continue in accordance with the size of the temporary queue list. The final queue will be used to store all of the jobs from the temporary queue list.

3.2.2. Proposed backfilling technique (SG- PBFS) algorithm

Traditional backfilling strategies backfill only current jobs in the queue; therefore, a new schedule must be generated for new jobs entering the system, which increases the cost of the operation as a whole. In contrast, the backfilling scheduler presented in this paper is intended to manage newly submitted jobs by cloud users by using the new job as a candidate job to be moved forward. By bypassing other jobs in the queue, this job will be immediately routed to the respective resource to be processed if starvation occurs at the top of the queue, avoiding the need to construct a new schedule. A new schedule is only required if new positions do not satisfy the backfill criteria. In contrast to the usual backfilling technique, which generates this schedule using FCFS, this paper has also examined employing other PR schedulers because they have been demonstrated to be more efficient than FCFS. In addition, earlier research on dynamic scheduling has determined that it is highly desirable to generate a new schedule frequently in order for newly arrived jobs can be incorporated into the schedule as soon as they arrive [42]. The primary issue with this method is that the continually shifting schedules might cause instability [43,44]. Instability produced by job-related disturbances during a rescheduling event. An incremental technique inspired by [45] has been implemented in the proposed scheduler to prevent instability, reduce computing costs, and successfully schedule jobs to cloud resources. By utilizing the most recently computed schedule as a starting point for constructing a new, up-to-date schedule, this method eliminates the excessive expense of creating a program from scratch.

This paper introduces the concept of backfilling, which is closely connected to the Shortest Gap (SG) presented by [46]. The shortest gap refers to the smallest available time interval between the start and end times of existing jobs. When applying the backfilling technique, the scheduler searches for smaller, shorter jobs that can fit within the shortest gap without violating any resource dependencies or job constraints. By filling these gaps with compatible smaller jobs, the scheduler aims to optimize resource utilization and improve system efficiency. The proposed Shortest Gap-Priority based fair scheduling (SG-PBFS) technique is specified in *Algorithm 2*.

Algorithm 2: SG – PBFS 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 PBFS to sort the jobs;
5:     Schedule = Place job into found gap [SG strategy];
6:   end if
7: end for
```

Step 1 and 2: According to the number of gaps in processing units, the loop will continue indefinitely. For instance, if the gap number is 5, the loop will repeat five times. After that, it is time to evaluate the abilities of the machines. If the machine is ready to execute the jobs, then the scheduler goes forward.

Step 3 to 5: In accordance with the FCFS, the initialqueue is the primary location where every job that has been submitted for backfilling is kept. After that, those are sorted by utilizing the PBFS, and then they are saved in the Readyqueue. Finally, all jobs are scheduled for resources according to the shortest Gap (SG).

In order to improve the performance of the cloud scheduler, a search module is created to scan for a suitable gap or idle slot that appears in the existing schedule created by the priority rule (PR). Once detected, the gap will be matched with a newly arrived job. This is done by taking into consideration the deadline of the jobs. By continuously filling the gaps with suitable jobs, more jobs can be completed before the deadline, and resources can be fully utilized. The detailed interaction between PR and the proposed backfilling technique can be seen in *Algorithm 3*.

Algorithm 3: Backfilling- Priority rules

```
1: resource_size = Total number of cloud-based resources;
2: for i = 0 to resource_size do
3:   if Total available resource < 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
```

In *Algorithm 3*, jobs submitted by users will be sent to the waiting queue based on FCFS where the head of the queue is the job that entered the queue first. The next step is to search for the best machine to process the job at the head of the queue. For this reason, all machines will be checked if a gap can be found in the specific machine schedule (machine schedule). Line 1 in *Algorithm 3* indicates that the parameter resourceSize will be used to store the number of resources available in the cloud. Availability means the resources are online, whether idle or in a processing mode. Looping in line 2 means the system will scan all online resources in search of the best solution.

3.3. Time complexity

The time complexity of the PBFS algorithm involves examining each step and determining how many times it is executed in terms of the input size (number of submitted jobs).

Cloud VMs Initialization: This step involves initializing cloud VMs, which typically takes a constant amount of time O(1).

Loop Over Submitted Jobs (Lines 4–18): The main loop iterates over each submitted job that is denoted by ‘n’. Within the loop, there are several nested conditions that compare job attributes such as CPU time, arrival time, and length. Every comparison takes constant time. The number of nested conditions does not depend on the input size ‘n’. It is a fixed number of comparisons. Therefore, the time complexity of the loop over submitted jobs (Lines 4–18) is O(n).

Loop Over Temporary Queue (Lines 20–22): This loop iterates over the temporary queue, which contains a subset of jobs from the main loop. For instance, the number of jobs in the temporary queue as ‘m’. This loop also takes O(m) time.

Table 1
CloudSim system Configuration.

Architecture	X86
Operating System (OS)	Linux
Processing Cost (PC)	3.0
Memory Cost (MC)	0.05
Time Zone	10
Storage Cost (SC)	0.1
Virtual Machine (VM)	1–30
Bandwidth Cost (BC)	0.1

Table 2
Job types of GoCJ dataset. [47].

Sr no	Job type	MI range	Distribution
1	Small	15000–55000	20%
2	Medium	59000–99000	40%
3	Large	101000–135000	30%
4	Extra large	150000–337500	6%
5	Huge	525000–900000	4%

Overall Complexity: Adding up the complexities of the individual steps:

Initialization of VMs: O(1)

Loop Over Submitted Jobs (Lines 4–18): O(n)

Loop Over Temporary Queue (Lines 20–22): O(m)

The dominant factor here is the loop over submitted jobs (Lines 4–18), as it has a complexity of O(n). So, the overall time complexity of the provided algorithm is O(n), where ‘n’ is the number of submitted jobs.

In terms of time complexity, both the PBFS algorithm and FCFS have a linear time complexity of O(n), which means they process jobs efficiently and directly proportional to the number of submitted jobs. SJF and LJF, on the other hand, involve sorting, which generally introduces a higher time complexity of O(n log n) due to the sorting step. EDF's time complexity can be variable based on implementation details and the nature of real-time scheduling. Though the complexity for PBFS and FCFS is the same, but the result shows that PBFS flow time, makespan time, and total tardiness are lower than the FCFS.

3.4. Environmental setup

This section sheds light on the simulation setup and delineates one benchmark workload employed in this study. We coded it in Java and used the Cloud Simulation Toolkit (CloudSim) to run it. CloudSim has a modular structure consisting of a number of components and packages. [48] employed CloudSim in their research article, where they used five VMs and set the RAM property for all virtual machines to 512 MB. They have used only 12 jobs. In experiments 1 and 2, [49] utilized 5 and 25 VM, respectively. In both experiments, 100 jobs were used. [50] developed a method in which the CloudSim tool is configured with two datacenters, 30 virtual machines, two hosts, and four processing elements (PEs), or CPU cores, for each host. For simplicity, implementation was done using 50 jobs, and analysis of the algorithms for additional jobs was also done. Above all research articles, the authors have used a small number of jobs, but the number of VMs is higher. This kind of configuration leads to higher costs. Considering all of these issues, this thesis configured the CloudSim according to the *Table* 1.

In order for scheduling to function effectively in the cloud, a proper abstraction of the accurate cloud computing system is necessary. This involves modeling various components such as datacenters, hosts, datacenter brokers, and virtual machines (VMs). To support the suggested scheduling method, modifications have been made to specific packages and classes, including “org.cloudbus.clousim” and “org.cloudbus.clousim.core” in CloudSim. The research involves the use of a cloud simulation layer and a user code layer to compare and evaluate different scheduling strategies and programs. The DatacenterBroker class in CloudSim provides the bindCloudletToVM() method, which allows for the assignment of specific workloads to particular VMs. By utilizing this class, the likelihood of selecting suitable VM resources for individual jobs is improved.

Using Google Cluster traces, [51] have developed the GoCJ data set, a realistic cloud workload. For evaluation purposes, the GoCJ data set is available in the Mendeley data repository as a collection of text files. The data is organized as rows of numerical numbers. The figure in millions of instructions reflects the magnitude of the cloud job. This data collection is comprised of five categories of occupations with varying proportions and is displayed in *Table* 2. Each text file of the GoCJ data set has a unique number of cloud workloads of varying sizes. The number of job requests at a particular time to a server is random. [52,53] have used the GoCJ dataset in their experiment, where their focus was primarily directed toward analyzing the performance of the algorithm with respect to a specific job, typically when the total number of jobs reached a specific point, such as 300 jobs. They did not thoroughly examine the variations and outcomes that occurred across all the jobs in the dataset. In this experiment, a total of 500 jobs were considered to evaluate the performance of the proposed algorithm under higher job load conditions.

4. Result and discussion

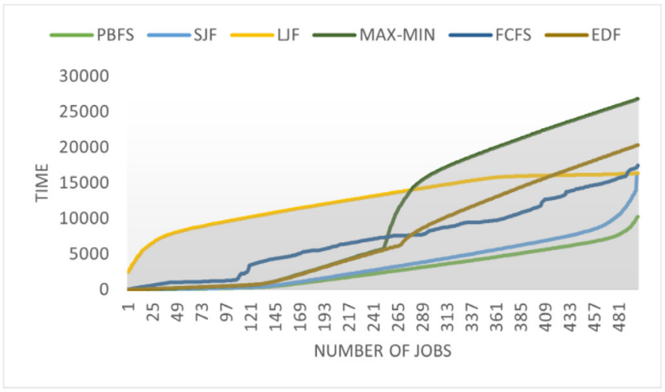
In this section, the experimental results have been analyzed. To complete the analysis 500 jobs have been considered for this experiment. Using the 500 jobs the proposed priority algorithm (PBFS) and backfilling technique (SG-PBFS) are compared with existing PR and backfilling techniques.

4.1. Experimental result of proposed PBFS, EG-PBFS and SG-PBFS using higher number job load

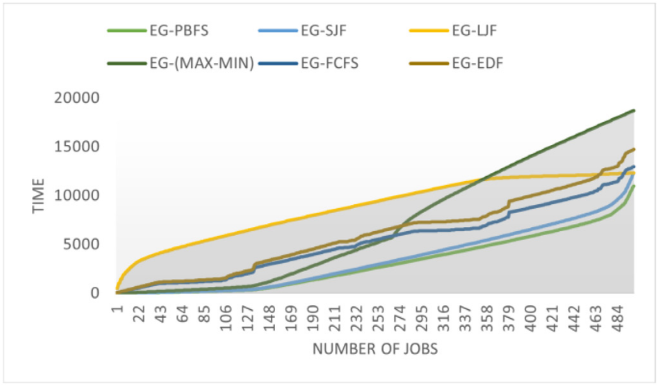
In this subsection, the experimental result of the proposed PR (PBFS) scheduler is evaluated by comparing it with the SJF [54], LJF [55], EDF [56], FCFS [57] and MAX-MIN [58] algorithms, where 500 jobs have been used. Besides, this subsection also compares the EG-PBFS with the EG-PR and the SG-PBFS with the SG-PR. Here, PR defines the five existing algorithms (SJF, LJF, EDF, FCFS, and MAX-MIN). EG is used to utilize the earliest gap, and SG is used for utilizing the shortest gap.

Fig. 4 shows the flow time comparison for six PR, EG-PR, and SG-PR algorithms gradually. *Fig.* 4*(a)* illustrate the flow time comparison between the proposed PR algorithm and existence algorithms. This figure shows that at the beginning, the LJF flow time is high than other algorithms, but with the increase of job load, this scenario is changed, where flow time has decreased. The opposite scenario is shown for the max–min algorithm. There is some fluctuation for FCFS and EDF. For SJF and PBFS, in the beginning, the performance is the same, but with the increase in the number of jobs, the flow time has decreased much for PBFS than SJF. When the number of jobs is 500, that time SJF flow time is 18% higher than the PBFS, and the max–min flow time is 93% higher than the PBFS. *Fig.* 4*(a)* shows that PBFS max flow time is around 10000 (milliseconds) but for SJF the value is more than 15000 (milliseconds). For all PR algorithms, the flow time range is 30000 (milliseconds). According to this experiment, in the *Fig.* 4*(a)*, flow time performance is best for proposed PBFS, and it is worst for LJF.

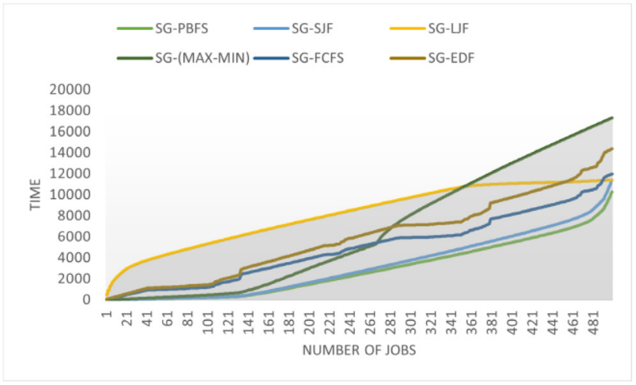
Fig. 4*(b)* shows the flow time comparison for EG-PR and proposed EG-PBFS. The flow time range is 20000 (milliseconds) in this figure where in *Fig.* 4*(a)* the time is 30000 (milliseconds),



(a)

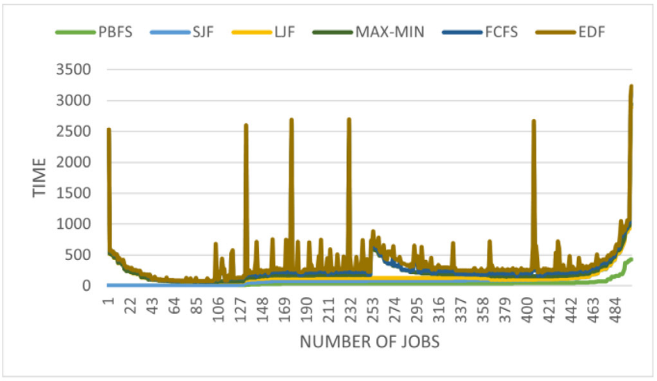


(b)

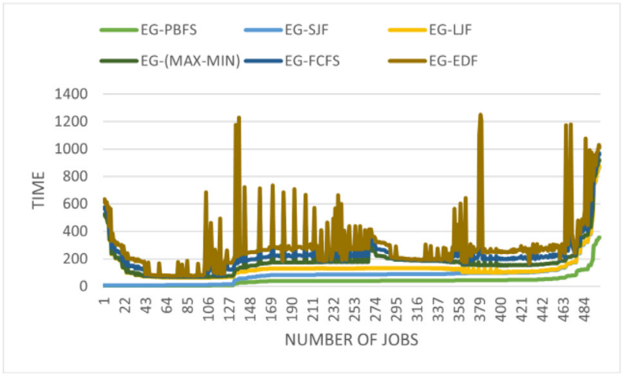


(c)

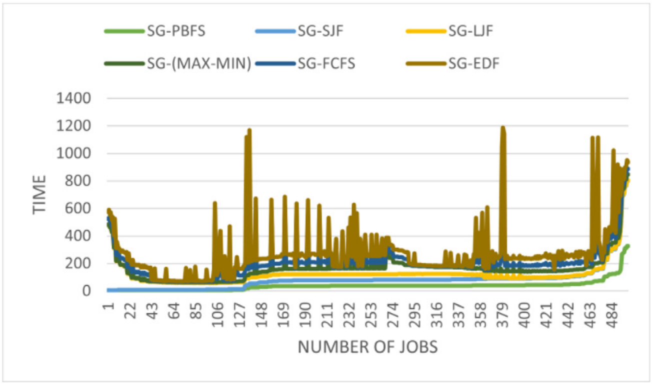
Fig. 4. Flow time comparisons between five algorithms for 500 Jobs. (a) PR Flow Time. (b) EG-PR Flow Time. (c) SG-PR Flow Time.



(a)



(b)



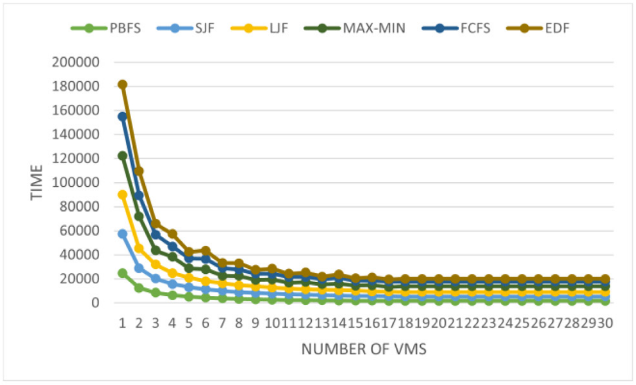
(c)

Fig. 5. Tardiness Time comparisons between five algorithms for 500 Jobs. (a) PR Tardiness Time. (b) EG-PR Tardiness Time. (c) SG-PR Tardiness Time.

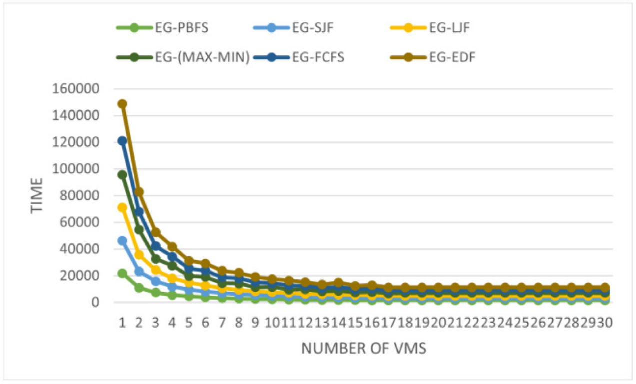
what is defined that utilizing the earliest gap all algorithms performance has been increased. According to Fig. 4*(a)*, this figure also shows that for EG-LJF and EG-(max–min), the flow time is higher compared to other algorithms. For EG-SJF and EG-PBFS, the flow time is low means that for both algorithms, the performance is better. In Fig. 4*(a)*, SJF flow time is 18% higher than PBFS, but the percentage decreases to 14% for EG-PBFS in Fig. 4*(b)*. For EG-FCFS and EG-EDF, flow times are almost similar. In Fig. 4*(a)*, the EDF flow time is higher at the beginning but increases after 270 jobs, but the scenario is different in Fig. 4*(b)*, where the EG-EDF flow time is low for all 500 jobs.

Fig. 4*(c)* describe the flow time comparison between six SG-PR algorithms. Instead of utilizing the earliest gap, when the shortest gap has been used, which has been a great improvement for all of these algorithms. For PR, the LJF flow time is higher, but when it is merged with SG, the flow time decreases a lot. According to the previous two figures, this figure shows the same result for all algorithms, but the flow time has decreased for all. Comparing all their figure, it is clear that SG-PBFS flow time is less compared to all PR, EG-PR, and SG-PR. In the point of 500 jobs, SG-SJF flow time is 13% higher than SG-PBFS.

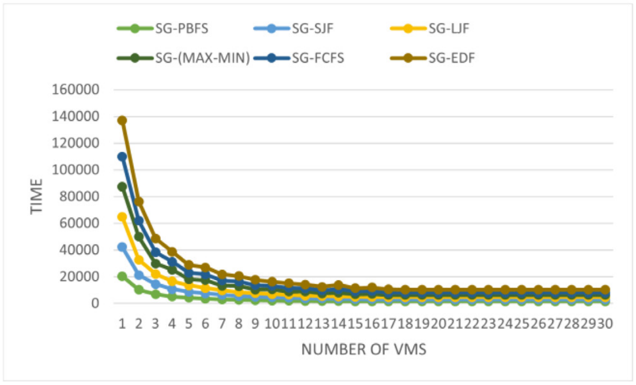
Fig. 5 illustrate the tardiness time for PR, EG-PR, and SG-PR where the number of jobs is 500. Fig. 5*(a)* shows the proposed PBFS performance comparison with five other existing PR algorithms. This figure shows that for EDF, at some points, the tardiness time is much higher, and this kind of scenario is also found for FCFS. Overall for all 500 jobs, EDF tardiness time is higher, which define that EDF performance is not good. The tardiness is less for LJF and SJF, though it is increased a lot after 463 jobs. The time is low for PBFS means that jobs do not have



(a)



(b)



(c)

Fig. 6. Makespan Time comparisons between five algorithms for 500 Jobs. (a) PR Makespan Time. (b) EG-PR Makespan Time. (c) SG-PR Makespan Time.

to need to wait a long time in the waiting queue. At the point of 500 jobs, the LJF tardiness time is 17% higher than the PBFS.

A great improvement has occur in Fig. 5(*b*) compared to Fig. 5(*a*). For PR, the tardiness time range is 3500 (milliseconds), but it decreases to 1400 (milliseconds) in EG-PR. Fig. 5(*b*) shows a great fluctuation for all algorithms. For EG-EDF, ED-FCFS, and EG-(MAX-MIN), the tardiness time is higher. For EG-LJF, the time is in the middle state. For EG-SJF and EG-PBFS, the performance is better, but especially for EG-PBFS, it is the best. From the beginning of the 358th job, the tardiness time is less for EG-SJF compared to EG-LJF. But after that, the EG-SJF tardiness time has increased greatly than EG-LJF. At the point of 500 jobs, EG-PBFS tardiness time is 15% less than EG-LJF.

When SG is merged with the PR, the tardiness time decreases a lot compared to the previous two techniques that are shown in Fig. 5(*c*). Though the performance is almost similar for all algorithms according to Fig. 5(*b*), but the tardiness has decreased for all algorithms. For EG-PBFS, the tardiness time maximum value is almost 400 (milliseconds), but in SG-PBFS, the time is less than 400 (milliseconds).

Fig. 6 explains the makespan time for PR, EG-PR, and SG-PR, where 30 VMs have been used. Fig. 6(*a*) show the makespan time comparison between PBFS and the other five PR algorithms. When the number of VMs is low, there is a large difference between all algorithms. The makespan time is higher for EDF, but the time is less for PBFS. When the number of VM is one, that time EDF makespan time is 93% higher than the PBFS. The performance is also good for SJF, and the makespan time is 37% is higher than PBFS. In Fig. 6(*b*) PR is merged with EG. The Makespan time range has been changed significantly decreased in EG-PR. For PR, the makespan time is 20000 (milliseconds), whereas EG-PR is 16000 (milliseconds). When PR is added with FG, the makespan time reduces noticeably, which is shown in Fig. 6(*c*). With the increase of VMs, the difference in makespan time has been reduced, which defines the better performance of algorithms.

5. Conclusion

This research began with an analysis of PR to determine its strengths and weaknesses with respect to makespan, flowtime, and total tardiness performance criteria. The most surprising result is that existing PR algorithms are not optimal schedulers for any of the performance criteria. This is significant evidence for the replacement of the current implementation of existing algorithms. To overcome this limitation, we have proposed a PR algorithm called PBFS. This PR outperforms comparing with existing algorithms. In cloud computing job scheduling, blocking is a big concern that refers to a situation where a job is unable to proceed or make progress due to various reasons, such as the unavailability of required resources or dependencies, resource conflicts, or scheduling constraints. When a job is blocked, it is unable to execute and may have to wait until the blocking condition is resolved before it can resume execution. This can result in delays, decreased resource utilization, and potentially impact the overall performance and efficiency of the system. However, current popular backfilling methods such as EG-EDF only address the blocking caused by EDF. Similarly, prior research articles have primarily focused on utilizing First-Come-First-Served (FCFS) as the sole backfilling technique. However, these existing backfilling techniques come with their own set of limitations and drawbacks. In order to overcome these limitations and enhance the efficiency of job scheduling, we propose a novel backfilling technique called Shortest Gap - Priority-Based Fair Scheduling (SG-PBFS). To evaluate the performance of our proposed backfilling technique, we conduct a comparative analysis with several existing algorithms, including EG-SJF, EG-LJF, and others. By conducting this comparison, we aim to assess the effectiveness and superiority of SG-PBFS over these established approaches. Through this research, we strive to contribute to the field of cloud computing job scheduling by introducing a new technique that overcomes the limitations of existing backfilling techniques. By leveraging the concept of shortest gap and prioritized fair scheduling, SG-PBFS has the potential to significantly improve the overall efficiency and performance of job scheduling in cloud computing environments.

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] A. Jakóbik, D. Grzonka, F. Palmieri, Non-deterministic security driven meta scheduler for distributed cloud organizations, Simul. Model. Pract. Theory 76 (2017) 67–81.

[2] Y. Pachipala, J. Alzubi, Managing the cloud storage using deduplication and secured fuzzy keyword search for multiple, Int. J. Pure Appl. Math. 118 (14) (2018) 563–565.

[3] M. Maray, J. Shuja, Computation offloading in mobile cloud computing and mobile edge computing: Survey, taxonomy, and open issues, Mob. Inf. Syst. 2022 (2022).

[4] Y.M. Ren, Y. Ding, Y. Zhang, P.D. Christofides, A three-level hierachical framework for additive manufacturing, Digit. Chem. Eng. 1 (2021) 100001.

[5] L.G. Garcia, E. Montoya, S. Isaza, R.A. Velasquez, An open edx extension for parallel programming assignments with automatic configurable grading, Int. J. Eng. Pedagogy 11 (4) (2021).

[6] M.K. Abhishek, D. Rajeswara Rao, A scalable framework for high-performance computing with cloud, in: ICT Systems and Sustainability, Springer, 2022, pp. 225–236.

[7] K. Fang, N. Uhan, F. Zhao, J.W. Sutherland, A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction, J. Manuf. Syst. 30 (4) (2011) 234–240.

[8] R. Chen, X. Chen, C. Yang, Using a task dependency job-scheduling method to make energy savings in a cloud computing environment, J. Supercomput. 78 (3) (2022) 4550–4573.

[9] J. Praveenchandar, A. Tamilarasi, Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing, J. Ambient Intell. Humaniz. Comput. 12 (3) (2021) 4147–4159.

[10] S. Elmougy, S. Sarhan, M. Joundy, A novel hybrid of shortest job first and round Robin with dynamic variable quantum time task scheduling technique, J. Cloud Comput. 6 (1) (2017) 1–12.

[11] S.M. Abdulhamid, M.S.A. Latiff, I. Idris, Tasks scheduling technique using league championship algorithm for makespan minimization in IAAS cloud, 2015, arXiv preprint arXiv:1510.03173.

[12] G. Gupta, V.K. Kumawat, P. Laxmi, D. Singh, V. Jain, R. Singh, A simulation of priority based earliest deadline first scheduling for cloud computing system, in: 2014 First International Conference on Networks & Soft Computing, ICNSC2014, IEEE, 2014, pp. 35–39.

[13] Z.R.M. Azmi, K.A. Bakar, A.H. Abdullah, M.S. Shamsir, W.N.W. Manan, Performance comparison of priority rule scheduling algorithms using different inter arrival time jobs in grid environment, Int. J.Grid Distribut. Comput. 4 (3) (2011) 61–70.

[14] S.H. Sahraei, M.M.R. Kashani, J. Rezazadeh, R. Farahbakhsh, Efficient job scheduling in cloud computing based on genetic algorithm, Int. J. Commun. Netw. Distribut. Syst. 22 (4) (2019) 447–467.

[15] J. Kopanski, Optimisation of job scheduling for supercomputers with burst buffers, 2021, arXiv preprint arXiv:2111.10200.

[16] K. Eng, A. Muhammed, M.A. Mohamed, S. Hasan, A hybrid heuristic of variable neighbourhood descent and great Deluge algorithm for efficient task scheduling in grid computing, European J. Oper. Res. 284 (1) (2020) 75–86.

[17] T. Aladwani, Types of Task Scheduling Algorithms in Cloud Computing Environment, IntechOpen, 2020, http://dx.doi.org/10.5772/intechopen.86873.

[18] L. Williams, Priority scheduling algorithm: preemptive, non-preemptive example, 2021, URL https://www.guru99.com/priority-scheduling-program.html.

[19] D. Dutta, S. Rath, Job scheduling on computational grids using multi-objective fuzzy particle swarm optimization, in: Soft Computing: Theories and Applications, Springer, 2022, pp. 333–347.

[20] H. Qin, S. Zawad, Y. Zhou, L. Yang, D. Zhao, F. Yan, Swift machine learning model serving scheduling: A region based reinforcement learning approach, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019, pp. 1–23.

[21] M.A. Alworafi, A. Dhari, S.A. El-Booz, A.A. Nasr, A. Arpitha, S. Mallappa, An enhanced task scheduling in cloud computing based on hybrid approach, in: Data Analytics and Learning, Springer, 2019, pp. 11–25.

[22] M. Kumar, S.C. Sharma, Priority aware longest job first (PA-LJF) algorithm for utilization of the resource in cloud environment, in: 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom, IEEE, 2016, pp. 415–420.

[23] S.H. Funk, EDF Scheduling on Heterogeneous Multiprocessors, The University of North Carolina at Chapel Hill, 2004.

[24] M. Kumar, S.C. Sharma, A. Goel, S.P. Singh, A comprehensive survey for scheduling techniques in cloud computing, J. Netw. Comput. Appl. 143 (2019) 1–33.

[25] J. Kumar, A.K. Singh, R. Buyya, Self directed learning based workload forecasting model for cloud resource management, Inform. Sci. 543 (2021) 345–366.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

[32] D. Carastan-Santos, R.Y. De Camargo, D. Trystram, S. Zrigui, One can only gain by replacing EASY backfilling: A simple scheduling policies case study, in: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, IEEE, 2019, pp. 1–10.

[33] H. Ali, M.S. Qureshi, M.B. Qureshi, A.A. Khan, M. Zakarya, M. Fayaz, An energy and performance aware scheduler for real-time tasks in cloud datacentres, IEEE Access 8 (2020) 161288–161303.

[34] S.C. Nayak, S. Parida, C. Tripathy, B. Pati, C.R. Panigrahi, Multicriteria decision-making techniques for avoiding similar task scheduling conflict in cloud computing, Int. J. Commun. Syst. 33 (13) (2020) e4126.

[35] O.A. Alzubi, J.A. Alzubi, A.M. Al-Zoubi, M.A. Hassonah, U. Kose, An efficient malware detection approach with feature weighting based on Harris Hawks optimization, Cluster Comput. (2022) 1–19.

[36] S. Rekha, C. Kalaiselvi, Load balancing using SJF-MMBF and SJF-ELM approach, 2021.

[37] T. Servranckx, M. Vanhoucke, A Tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs, European J. Oper. Res. 273 (3) (2019) 841–860.

[38] X.-Z. Gao, S. Gurumurthy, S. Venkatesan, Improved CPU utilization using advanced fuzzy based CPU scheduling algorithm (AFCS), Int. J. Electr. Sci. Eng. (IJESE) 1 (1) (2015) 1–5.

[39] J.P. Lehoczky, Fixed priority scheduling of periodic task sets with arbitrary deadlines, in: [1990] Proceedings 11th Real-Time Systems Symposium, IEEE, 1990, pp. 201–209.

[40] J.A. Alzubi, R. Manikandan, O.A. Alzubi, I. Qiqieh, R. Rahim, D. Gupta, A. Khanna, Hashed Needham Schroeder industrial IoT based cost optimized deep secured data transmission in cloud, Measurement 150 (2020) 107077.

[41] S. Gupta, S. Iyer, G. Agarwal, P. Manoharan, A.D. Algarni, G. Aldehim, K. Raahemifar, Efficient prioritization and processor selection schemes for HEFT algorithm: A makespan optimizer for task scheduling in cloud environment, Electronics 11 (16) (2022) 2557.

[42] R. Rangsaritratsamee, W.G. Ferrell Jr., M.B. Kurz, Dynamic rescheduling that simultaneously considers efficiency and stability, Comput. Ind. Eng. 46 (1) (2004) 1–15.

[43] K. Kang, et al., MAS equipped with ant colony applied into dynamic job shop scheduling, in: International Conference on Intelligent Computing, Springer, 2007, pp. 823–835.

[44] 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.

[45] R. Kumar, G. Sahoo, Cloud computing simulation using CloudSim, 2014, arXiv preprint arXiv:1403.3253.

[46] D. Klusacek, H. Rudova, Improving QoS in computational grids through schedule-based approach, in: Scheduling and Planning Applications Workshop At the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, 2008.

[47] D. Alsadie, TSMGWO: Optimizing task schedule using multi-objectives grey wolf optimizer for cloud data centers, IEEE Access 9 (2021) 37707–37725.

[48] D. Agarwal, S. Jain, et al., Efficient optimal algorithm of task scheduling in cloud computing environment, 2014, arXiv preprint arXiv:1404.2076.

[49] W. Lin, C. Liang, J.Z. Wang, R. Buyya, Bandwidth-aware divisible task scheduling for cloud computing, Softw. - Pract. Exp. 44 (2) (2014) 163–174.

[50] N. Bansal, A. Maurya, T. Kumar, M. Singh, S. Bansal, Cost performance of QoS driven task scheduling in cloud computing, Procedia Comput. Sci. 57 (2015) 126–130.

[51] A. Hussain, M. Aleem, GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures, Data 3 (4) (2018) 38.

[52] A. Hussain, M. Aleem, M.A. Iqbal, M.A. Islam, SLA-RALBA: Cost-efficient and resource-aware load balancing algorithm for cloud computing, J. Supercomput. 75 (10) (2019) 6777–6803.
[53] S.E. Shukri, R. Al-Sayyed, A. Hudaib, S. Mirjalili, Enhanced multi-verse optimizer for task scheduling in cloud computing environments, Expert Syst. Appl. 168 (2021) 114230.
[54] K. Hansen, [Hutch housing of calves [Seem, Dan, EM, Funen, SJF]], in: Orientering Statens Jordbrugstekniske Forsoeg (Denmark), SJF, 1985.
[55] F. Sluijter, L. van Wijngaarden, A brief summary of LJF Broer's work up till his retirement, Appl. Sci. Res. 37 (1) (1981) 4–19.
[56] Y. Eto, T. Tsuji, M. Takezawa, S. Takano, Y. Yokogawa, H. Shibai, Purification and characterization of erythroid differentiation factor (EDF) isolated from human Leukemia cell line THP-1, Biochem. Biophys. Res. Commun. 142 (3) (1987) 1095–1103.
[57] A. Marphatia, A. Muhnot, T. Sachdeva, E. Shukla, L. Kurup, Optimization of FCFS based resource provisioning algorithm for cloud computing, IOSR J.Comput. Eng. (IOSR-JCE) 10 (5) (2013) 1–5.
[58] Y. Lu, Y. Arkun, Quasi-min–max mpc algorithms for lpv systems, Automatica 36 (2000) 527–540.

Saydul Akbar Murad is a researcher who has completed B.Sc. Engr. degree in Information and Communication Engineering (ICE) department from Noakhali Science and Technology University (NSTU), Noakhali, Bangladesh. He is currently pursuing his master's degree in the Faculty of Computing from Universiti Malaysia Pahang, Malaysia. His current research interests are Cloud Computing, Edge Computing, machine learning, and Neural Networks. He has published total fourteen articles in some reputed journals and conferences. Besides he also worked as a reviewer for many journals and conferences.

Zafril Rizal Bin M Azmi is a senior lecturer at the Faculty of Computing, Universiti Malaysia Pahang, Malaysia. He has been working in Universiti Malaysia Pahang since 2015. He received his Bachelor, Master's and Ph.D. in computer science from Universiti Teknologi Malaysia. He has authored and coauthored around 30 publications, including refereed IEEE/ACM journals, and conference articles. Dr. Zafril is performing events as reviewer and editorial board member supportive of distinct scholarly journals. His research interests include Grid Computing, Mobile Computing and Cloud Computing.

Abu Jafar Md Muzahid received the B.Sc. and the M.Sc. degree in Statistics from the Shahjalal University of Science and Technology, Sylhet, Bangladesh. He also completed his second master's degree in the faculty of Computing at University Malaysia Pahang, Malaysia. His main areas of research interest are Artificial Intelligence, Machine Learning, and green automotive technology.

Md. Khairul Bashar Bhuiyan is an undergraduate research assistant at Control Application and Research Center (CARC). He has received Bachelor's degree in Electrical and Electronics Engineering from Brac University. He has previously worked in projects related to satellite training kit and lie-detector polygraph machine. He is currently working in projects related to Drone communication and Remote sensing. He is interested in research fields like Machine learning, IoT and satellite communication.

Md Saib, is a highly motivated and dedicated student currently pursuing a Bachelor of Engineering degree in Computer Science and Technology at South China University of Technology. His interest in the field of computer science and technology started at an early age and has been a driving force in his academic pursuits. His passion for technology has led him to the field of cloud computing, and his research interest is focused on how cloud computing can be used to improve the efficiency and scalability of various industries. He has been working on projects that aim to implement cloud computing solutions in various business domains. He look forward to making significant contributions to the field through his future research work.

Nick Rahimi is an Assistant Professor at the School of Computing Sciences & Engineering of the Southern Mississippi University, USA. He has earned two Bachelor of Science in Computer Software Engineering and Information Systems Technologies (Cybersecurity concentration) and obtained his Master's and Ph.D. degrees in Computer Science from Southern Illinois University, USA.

Nusrat Jahan Prottasha received the B.Sc. degree in computer science and engineering from Daffodil International University, in 2022. She is currently working with Data Science Platform as a Research Assistant at Stevens Institute of Technology. In 2020, she received the Best Paper Award from the International Conference of Cyber Security and Computer Science. Besides, in recognition of scholarly publication in the reputed indexed journal has been awarded for publishing four articles in Scopus journals from her university.

Anupam Kumar Bairagi (S'17-M'18-SM'22) received his Ph.D. degree in Computer Engineering from Kyung Hee University, South Korea and B.Sc. and M.Sc. degree in Computer Science and Engineering from Khulna University (KU), Bangladesh. He is an professor in Computer Science and Engineering discipline, Khulna University, Bangladesh. His research interests include wireless resource management in 5G and beyond, Healthcare, IIoT, cooperative communication, and game theory. He has authored and coauthored around 60 publications including refereed IEEE/ACM journals, and conference papers. He has served as a technical program committee member in different international conferences. He is a senior member of IEEE.