Date of publication DEC-31, 2025, date of current version AUG-08, 2025
www.computingonline.net/ computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.4.4337

A Predictive and Availability-Aware Job Scheduling Algorithm for Resource Management in Cloud

UDDALOK SEN[1], MADHULINA SARKAR[2], NANDINI MUKHERJEE[3]

[1]Dept. of Information Technology, MCKV Institute of Engineering, G T Road North, Liluah, Howrah, 711204, West Bengal, India.(e-mail: uddaloksen81@gmail.com)

[2]Dept. of Computer Science and Engineering, Govt. College of Engineering and Textile Technology Berhampore, 4, Cantonment Road, Murshidabad, 742101, West Bengal, India. (e-mail: madhulina.sarkar@gmail.com)

[3]Dept. of Computer Science and Engineering, Jadavpur University, 188, Raja S.C. Mallick Rd, Kolkata, 700032, West Bengal, India. (e-mail: nandini.mukhopadhyay@jadavpuruniversity.in)

Corresponding author: Uddalok Sen (e-mail: uddaloksen81@gmail.com).

ABSTRACT To propose an efficient scheduling algorithm in a large distributed heterogeneous environment like cloud, resource (CPU cycles, memory) requirement of jobs must be predicted prior to the execution. An execution history can be maintained to store execution profile of all jobs executed earlier on the given set of resources. A feedback guided job modelling scheme is proposed earlier to detect similarity between newly submitted job and previously executed jobs on that resource set. Based on the similarity the new jobs are categorized as either an exact clone or near-miss clone or miss-clone to the history jobs. However, researchers have shown that the actual resource consumption, and predicted resource requirement may differ to a great extent, especially for the near-miss-clone and miss-clone jobs. Furthermore, efficient resource scheduling based on the similarity of new jobs has not been addressed in the previous work. Some studies show that even if the resource requirements of jobs are predicted accurately, it is nearly impossible to predict the actual execution time on a given resource, and actual execution time is only available after the completion of the job. Ignoring uncertain facts at the time of scheduling may lead to unsuccessful completion of jobs, especially, where resources are available for the limited period of time, like in the case of cloud. In this work, we propose an efficient scheduling approach that selects a resource for a job based on two critical criteria. Firstly, the selected resource is evaluated to ensure a faster completion time. Secondly, the availability of the resource until the completion of the assigned jobs is ensured. In addition, this work proposes optimization of these two criteria during the resource selection process. Finally, we compare the efficiency of our scheduling algorithm with some well-known job scheduling algorithms.

KEYWORDS Job scheduler; heterogeneous system; resource availability; clone-based job modeling

I. INTRODUCTION

Assigning jobs to resources (matching) and executing jobs in the best order (scheduling, also known as mapping [1]), are well-known challenges in large distributed systems. Efficient methods are needed to address these challenges. The main objective of a mapping scheme is to ensure minimum make span (completion time) of the jobs. Mapping of jobs in a large, distributed heterogeneous environment is well known for its hardness as this problem belongs to the NP-Complete class. As an effective scheduling strategy, the researchers focus on the prediction of the resource requirements (CPU cycles, memory) for jobs prior to their execution. To predict the resource requirement of a newly submitted job in a large heterogeneous distributed system, a feedback-guided job modeling is proposed in [2]. The similarities between recently submitted jobs and jobs that have already been completed can be identified with clone detection technique and maintaining an execution history of previously executed

jobs [3]. As discussed in this paper, the 'before execution' parameters, i.e. job type, number of variables, loop, etc. are available prior to the execution of a job. Once a job is finished the 'after-execution' parameters, i.e. number of CPU cycles, memory consumed, etc. are available and recorded accordingly [3]. The jobs that have been executed earlier in the system are termed history jobs. Based on the similarity found between a newly arrived job with the history jobs, a new job can be classified either as an exact clone, or a near-miss clone, or a miss-clone. Based on the level of similarity of a new job with the history jobs, a feedback guided job modeling scheme predicts the resource requirements, i.e. the expected after execution parameters of a newly arrived job prior to its execution. The study presented in [3] demonstrates that although the proposed technique can effectively predict resource requirements for exact clones, for near-miss clone and miss-clone jobs, real resource consumption and anticipated resource requirements differ. It may also be noted that efficient resource scheduling based on the available knowledge has not been addressed in the above research work. Furthermore, some studies show that due to the uncertainty of the environment, even if the resource requirements of jobs are predicted accurately, it is almost impossible to predict their actual execution time. Actual execution time of a job is only available after its completion [4]. In distributed environment like cloud, resources are available for a limited period of time and inaccurate prediction of resource requirements may lead to either under or over provisioning of resources.

This paper attempts to address the above issues by proposing a resource scheduling algorithm. In this work, it is assumed that resources are available for a fixed period of time, i.e. contract period is fixed. No job can be executed on a resource once its contract period is over. During execution of a job, if the contract period of the resource on which it is executing ends, the job will be marked as failed. In this work, a scheduling algorithm is proposed that selects a resource for a job based on two critical parameters simultaneously: the resource should offer the fastest completion time for the job and have the highest probability of being available until the job is completed, compared to all other resources. We compare the efficiency of our scheduling algorithm with some well-known job scheduling algorithms.

The paper is organized as follows. A brief overview of related works is discussed in Section II. Models and assumptions are discussed in Section III along with the algorithms for resource scheduling. Some experimental results based on the algorithms are shown in Section IV to demonstrate the efficiency of proposed algorithm. Section V concludes with a direction for future work.

II. RELATED WORK

In distributed systems, a large set of heterogeneous pool of resources are managed by resource management component. By heterogeneity it is meant that resources have different processing speed and memory, different pricing scheme and different duration of available period. During the allocation of resources, a resource management component is responsible to allocate appropriate resources to the jobs. Scheduling tasks onto these heterogeneous resources in a distributed environment is a challenging task. Finding an optimal mapping of independent tasks onto the available resources has been identified as an NP-complete problem [5] [6]. During the last few decades, numerous researchers have addressed this problem and proposed a large number of mapping heuristics and task scheduling techniques. Braun et al [5] proposed a taxonomy for mapping heuristics. They defined the taxonomy primarily in three major parts, first the models used for applications and communication requests, second the models used for target hardware platforms, and third the characteristics of mapping heuristics.

In general, the scheduling problem is broadly classified as static and dynamic scheduling. In the case of static scheduling, it is assumed that information about resource requirements of tasks and the availability of resources is known in advance. On the other hand, in the case of a dynamic scheduler such information is not available beforehand. A survey of various static scheduling algorithms and their functionalities have been described in [7] in a contrasting fashion, and their relative merits were examined in terms of performance and time complexity. A taxonomy was also proposed that classifies the algorithms into different categories. In another study [8], a few simple and straightforward static heuristics, including Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min–min, Max–min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A* were compared by implementing them and observing the results. This study helped to develop insights into circumstances where one technique would outperform another. In [9], dynamic scheduling heuristics for a class of independent tasks have been discussed. The authors considered two types of mapping heuristics – immediate mode and batch mode. They also presented simulation results and observed that the choice of heuristics in a heterogeneous environment depends on parameters like structure of heterogeneity among tasks and machines, arrival rate of the tasks etc. However, the above studies do not address the issues related to modern state-of-the-art distributed environments, such as cloud.

During the recent years several researchers focused on task scheduling in cloud and other distributed environment. Some researchers proposed static scheduling algorithms, and others proposed dynamic algorithms in heterogeneous distributed computing environment aiming at minimizing makespan, reducing cost etc. Khan et al [10] presents an elaborate study of the state-of-the-art task scheduling algorithms in cloud and fog environments. They include both static and dynamic scheduling and observed that most of the scheduling algorithms are dynamic and non-preemptive in nature, and also in most cases independent tasks were considered. According to this study, scheduling

algorithms are broadly classified into three categories, such as heuristic, meta-heuristic, and hybrid meta-heuristic. The scheduling algorithms have different scheduling objectives including minimizing makespan, delay, energy consumption, maximizing resource utilization, load balancing, etc. The authors in [11] conducted a systematic literature review of task scheduling in cloud computing. They introduced a classification taxonomy and a comparative review of various techniques. The proposed taxonomy categorized metaheuristic scheduling techniques based on scheduling algorithms, problem nature, task types, scheduling objectives, task-resource mapping, scheduling constraints, and testing environments.

In [12], a Static Independent Task Scheduling method for cloud computing has been proposed, where tasks were assigned to virtual machines (VMs) based on resource availability, processing power, cost, and the number of processing elements. Here, tasks were grouped by instruction length, and the method was simulated using CloudSim. The work was compared considering two metrics, total execution time and execution cost with Shortest Job First (SJF) and First Come First Serve (FCFS) algorithms.Another resource management system RTF-RMS is proposed for executing Real-Time Online Interactive Applications (ROIA) on cloud infrastructures [13]. The system introduces a dynamic load-balancing strategy that selects among three possible actions: user migration, replication enactment, and resource substitution, based on current system conditions. Additionally, RTF-RMS supports cost-effective resource leasing by buffering unused resources. The Minimum Makespan Scheduling Framework (MMSF) and the Minimum Makespan Algorithm (MMA) for cloud task scheduling have been proposed in [14]. It aimed at minimizing total makespan and maximizing VM utilization, the problem was formulated as a multi-objective optimization. Experiments compared MMA performance with traditional scheduling algorithms in makespan reduction and VM utilization. A task scheduling approach that groups tasks based on users' resource demands and cost considerations has been proposed in [15]. Compared to traditional methods, this approach reduces bandwidth, memory, and storage costs while staying within budget constraints. A multi-objective Artificial Bee Colony Algorithm (TA-ABC) for cloud task scheduling by optimizing energy, cost, resource utilization, and processing time has been proposed in [16]. This work was simulated using CloudSim, and was compared with existing scheduling algorithms. In [17], a two-stage task scheduling method for cloud computing has been proposed. In the first stage, a Bayes-inspired job classifier used historical data to categorize tasks, enabling pre-creation of suitable VMs to save scheduling time. In the second stage, tasks were dynamically matched with VMs. Experimental results demonstrated scheduling performance and load balancing criteria compared to some traditional methods. Sanaj et al [18] proposed an Enhanced Round Robin (ERR) algorithm that improves performance while retaining the advantages of traditional Round Robin (RR) scheduling. This work was implemented using CloudSim and results showed that ERR reduces average waiting time compared to conventional RR. They also compared their work with other algorithms like ACO, GA, MPA, Min-Min, and PSO in execution time and energy efficiency. The TQ (Three Queues) cloud task scheduling algorithm, which uses dynamic priority and categorizes jobs based on data input/output in the Map phase, node task load, Map task completion time, and disk I/O rate, has been proposed in [19]. In this work, jobs are placed in corresponding queues to enhance hardware utilization and experiments show that TQ performs cloud scheduling by reducing total task completion time for mixed I/O-intensive and CPU-intensive jobs. Younes et al [20] presented a genetic algorithm-based solution to the task scheduling problem in distributed systems. They have considered dependent tasks and constructed a directed acyclic graph to show the dependencies. In [21], a dynamic resource allocation model for responsive cloud services, along with the Spacing Multi-Objective Antlion Algorithm (S-MOAL), has been proposed to minimize makespan and VM costs. It also examines fault tolerance and energy consumption. Simulations were carried out to compare S-MOAL with PBACO, DCLCA, DSOS, and MOGA algorithms, particularly focusing on makespan reduction. In another study [22], the Whale Optimization Algorithm (WOA) for cloud task scheduling using a multi-objective optimization model to enhance cloud system performance has been proposed. They presented simulation results to compare the performance of their algorithm with existing meta-heuristic algorithms in convergence speed, accuracy, and system resource utilization for both small and large-scale tasks. Another work [23] proposes a Particle Swarm Optimization (PSO) algorithm using heuristic initialization with Longest Job to Fastest Processor (LJFP) and Minimum Completion Time (MCT) methods. The LJFP-PSO and MCT-PSO algorithms are evaluated in terms of makespan, execution time, imbalance, and energy consumption metrics. They compared their work with conventional PSO and other recent scheduling methods. Task scheduling problem in a cloud computing environment has been addressed in [24]. A priority assignment strategy was proposed for the individual tasks upon their arrival. Additionally, waiting queue was implemented using Fibonacci heap for extracting the task with the highest priority. The scheduling algorithm was applied in a dynamic cloud computing environment. A decentralized Belief-Desire-Intention (BDI) agent-based scheduling framework has been proposed in [25] for cloud environments to effectively manage task scheduling under uncertainty. The framework utilizes asynchronous communication with a notify listener to prevent communication deadlocks caused by real-world disruptions. It introduces dedicated scheduling and rescheduling algorithms, along with a cycle recommendation algorithm to reduce synchronization overhead. A novel Unified Deep Learning (UDL)-based model for optimizing task scheduling and resource

allocation (TSRA) in cloud environments featuring multiple task queues and VM clusters is proposed in [26].In this work, the UDL model introduced a two-part DNN architecture—exploration and exploitation networks—to balance randomness and learning efficiency. It dynamically adjusts weights for optimizing energy consumption and task latency.

Chen et al, in their research work [4] argued that traditional scheduling approaches ignore the uncertainties in the scheduling environment, such as the uncertain starting and finishing time of the tasks, uncertain data transfer time among tasks, sudden arrival of new workflows etc. They proposed a scheduling architecture to improve the performance of cloud service platforms by reducing uncertainty propagation in scheduling workflow applications that have both uncertain task execution time and data transfer time.

The researchers also proposed the incorporation of a resource broker component for dynamic task scheduling in a distributed environment. A resource broker acts as a central coordinator, responsible for identifying, evaluating, and allocating available resources across a network to incoming tasks based on their requirements. A resource broker ensures optimal utilization of the system by matching jobs with the most suitable computing power and thereby ensures efficient task execution. A resource broker that addresses scheduling scenarios in large heterogeneous environment like grid using the concepts of virtualization has been discussed in [27]. Necessary protocols and services to support creation and management of virtual resources in the physical hosts were proposed in this research work. Several failure cases of application scheduling, such as nonavailability of enough computing nodes in a cluster, nonavailability of software execution environment in any of the grid resources were discussed. A genetic algorithm-based resource broker for a computational Grid has been proposed in [28]. It works with the objective of minimizing the total cost of running the jobs or maximizing the utilization of Grid resources. It was observed that the search space was large as the problem consisted of all possible allocations of submitted jobs to available resource providers and genetic algorithms were found to be efficient for such an optimization problems, particularly for a dynamic workload.

In contrast with the above research works, this paper proposes resource allocation based on similarity estimation of jobs submitted to a distributed computing environment. This research work considers two different criteria. One such criterion is completion time of the newly submitted job whose resource requirement and completion time are predicted based on its clone level with reference to the jobs executed earlier in the system (known as history jobs). The other criterion deals with uncertainty of the environment as the resources are available for a limited period of time and will no longer be available once it is over. Moreover, the actual completion time and predicted completion time of a job may differ. Considering this uncertainty, the proposed allocation strategy also tries to allocate a resource for a given job such that there is a maximum chance that the resource will be available until the job is finished.

III. MODELING AND FORMULATION

a: Resource Modeling:

In a distributed system, like cloud, different types of resources are provided by a service provider. These resources have different processing speeds, architectures, and pricing schemes. Here, we have assumed that all resources are provided by a single service provider. Consider a set of n resources represented as $R_1, R_2, \ldots, R_n$. Processing speed, i.e. number of CPU cycles per unit time offered by a resource is $capacity(R_j)$. The price of a resource (R_j) is denoted by $price(R_j)$. In a distributed environment, like cloud, resources are available only for a fixed period of time. The available period of a resource (R_j) is denoted by $availTime(R_j)$. Here, we have assumed that once $availTime(R_j) = 0$, i.e. the contract period of resource R_j is over, then $resourceR_j$ is no longer available in the system and no job can be executed on it.

b: Job Modeling:

In a distributed system, a variety of jobs can arrive dynamically for execution. Here, we have assumed that before and after execution information of previously executed jobs are kept as *history* [3]. As per the *feedback guided job modeling* [3] scheme, newly submitted jobs can be categorized as an $exact - clone$, $nearmiss - clone$ and $miss - clone$ based on the similarity found with one or more job(s) kept in *history*. Suppose, there are m jobs, which are represented as $J_1, J_2, \ldots, J_m$. The predicted CPU cycle of J_i is $predCPU(J_i)$. The predicted start time of job J_i on resource R_j is denoted by $predStart(i, j)$. If the contract period of a resource is finished during the execution of a job, the job will be considered as failed and cannot be rescheduled on any other resources. However, the jobs that are waiting to be executed on that resource can be rescheduled on other resources.

c: Scheduling Architecture:

In this study, a space-shared scheduling approach is considered [29], where jobs awaiting execution on resource R_j are placed in a queue and processed sequentially. The predicted execution time of job J_i on resource R_j is denoted by $predEx(i, j)$, which can be expressed by $Eq(1)$.

$$\text{predEx}(i, j) = \frac{\text{predCPU}(J_i)}{\text{capacity}(R_j)} \tag{1}$$

d: Problem Formulation:

Due to uncertainty in the environment, it has been observed that actual execution time of a job J_i on a resource R_j is denoted by $actualEx(i, j)$ and it is unpredictable. It is assumed here that the execution time follows a normal distribution [4].

e: Definition:

In this work, it is assumed that $actualEx(i,j)$ is a continuous random variable following a normal distribution with the task base execution time as the mean denoted by $predEx(i,j)$, and a relative task runtime standard deviation $\delta(i,j)$. Due to uncertainty in the environment, $predEx(i,j)$ can be at most K_j^i*$\delta(i,j)$ away from the mean. The $\delta(i,j)$ will be considered as per $Eq(2)$ as given below [4]:

$$\delta(i,j) = \text{predEx}(i,j) \times \text{VarianceET} \tag{2}$$

where, $VarianceET$ is the variation in task execution times. Then the following condition must be satisfied for the successful completion of the job J_i on the resource R_j.

predStart(i,j) + predEx(i,j) + k_j^i * $\delta(i,j)$ $\leq$ availTime(R_j) Hence,

$$k_j^i \leq \frac{\text{availTime}(R_j) - (\text{predStart}(i,j) + \text{predEx}(i,j))}{\delta(i,j)} \tag{3}$$

For the purpose of ranking the resources (Algorithm 1), we compute the maximum value of k_j^i, such that the Eq (3) is satisfied. Thus, $predCom(i,j)$ is the predicted completion time of job J_i on resource R_j which can be written as follows:

$$\text{predCom}(i,j) = \text{predStart}(i,j) + \text{predEx}(i,j) \tag{4}$$

It is obvious that the higher is the value of k_j, the higher is the chance that resource R_j will be available until the job J_i is completed, reliability of $resourceR_j$ is high for higher value of k_j^i higher for successful completion of job J_i.

f: Objective:

Due to uncertainty in a distributed environment, a resource will be withdrawn from the environment once its contract period is over. So, if its contract period is over during the execution of a job, then the job will fail to complete. Further resource allocation for that job is not considered in this work. Let S denote the set of successfully completed jobs. Then our objective is to Maximize $|S|$ and to Minimize $\sum_{J_i \in S} \text{actualCT}(J_i)$

The workflow of the proposed approach is represented in 1.

g: Algorithm Design:

In our approach, a heuristic based batch mode dynamic scheduling algorithm is proposed. The main objective of our scheduling algorithm is to select a resource for a job such that the selected resource offers high reliability, as well as low completion time. Our algorithm is divided into three parts. The first part will compute rank of the resources for a particular job based on reliability, i.e. higher the rank, higher the reliability of that resource for successful completion of the job. The following method computes every available resource for the job J_i.

Algorithm 1

Rank_Reliability$(J_i, R_{\text{available}})$

Input: Job J_i, available resource set $R_{\text{available}}$

1: $S^i_{\text{reliability}} = \emptyset$

2: for every resource $R_j \in R_{\text{available}}$

3: Compute k_j^i as per the equation (3)

4: end for (2)

5: Select resource R_y with minimum k_y^i

6: for every resource $R_x \in R_{\text{available}}$

7: $R^i_{\text{reliability},j} \leftarrow (k_x^i - k_y^i)$

8: $S^i_{\text{reliability}} = S^i_{\text{reliability}} \cup R^i_{\text{reliability}_j}$

9: end for (6)

10: return $S^i_{\text{reliability}}$

The next method is to compute the rank of resources for a job based on its completion time on respective resources, i.e. higher rank implies earlier completion time. The following method computes the rank of every resource considering the job completion time.

Algorithm 2

Rank_CTime$(J_i, R_{\text{available}})$

Input: Job J_i, available resource set $R_{\text{available}}$

1: $S^i_{\text{CTime}} = \emptyset$

2: for every resource $R_j \in R_{\text{available}}$

3: Compute $predCom(i,j)$ as per the equation (3)

4: end for (2)

5: Select resource R_z with maximum $predCom(i,z)$

6: for every resource $R_j \in R_{\text{available}}$

7: $R^i_{\text{CTime},j} \leftarrow \text{predCom}(i,z) - \text{predCom}(i,j)$

8: $S^i_{\text{CTime}} = S^i_{\text{CTime}} \cup R^i_{\text{reliability}_j}$

9: end for (6)

10: return S^i_{CTime}

The third part, i.e. a heuristic-based batched mode dynamic scheduling algorithm is described below. Our $UncertainityAwareResourceSelection(J,R)$ algorithm selects a resource which offers early completion time with high reliability where J is the set of jobs and R is the set of resources. To select an optimal resource for a job i, we will select the resource j that offers maximum of $R\_\text{CTR}_j^i$ where,

$$R\_\text{CTR}_j^i = \text{Rank\_CTime}(J_i, R_{\text{available}}) * \text{Rank\_Reliability}(J_i, R_{\text{available}}) \tag{5}$$

However, if multiple resources offer the same $R\_CTR_j^i$ then any one of them can be selected randomly. To avoid the dominance of any one of the multipliers over the other, we scale these two parameters [23], i.e. the values of the two multipliers are changed to values between 0 and 1. Here, we have used the following algorithm to scale multipliers $Rank\_CTime(J_i, R_available)$ and $Rank\_Reliability(J_i, R_{\text{available}})$. The algorithm $Scale(S)$ takes a set and return a scaled set where each of the element is between 0 and 1.

Algorithm 3
Normalise(S)
1: Select minimum S_{min} and maximum S_{max} of S
2: for every $S_i \in S$
3: if S_{max} = S_{min}
4: S_i = S_i / S_{max}
5: else:
6: if $S_{\text{max}} = S_{\text{min}} = 0$
7: $S_i = 0$
8: end if(6), end if (3)
9: $S_i = (S_i - S_{\text{min}})/(S_{\text{max}} - S_{\text{mim}})$
10: end for (2)
11: return S

Next, we present the algorithm for scheduling a set of jobs onto resources.

Algorithm 4
UncertainityAwareResourceSelection$(\mathbf{J}, \mathbf{R})$

Input: J, R
Initialization: $R\_CTR_{m \times n}$
1: for each job J_i in task-set (in an arbitrary order)
2: Compute $R\_CTR_j^i$ as per equation (4)
3: end for (1)
4: do until all jobs in task-set are mapped
5: for each job J_i find the resource R_j that obtains maximum of $R\_CTR_j^i$ and find the
6: $predCom(i, j)$
7: $R\_CTR_{\text{mXn}}[i][j]$ = $predCom(i, j)$
8: end for (5)
9: select $predCom(k, l)$ as maximum of R_CTR_{mXn}
10: assign job J_k to the resource l
11: delete job J_k from the task-set
12: if actualEx$(i, j) >$ availTime(R_j)
13: Mark J_k as failed job
14: remove R_j from $R_{\text{available}}$
15: end if (10)
16: update ready time of resource R_j
17: Enddo(4)

Our algorithm always selects a resource which provides optimal reliability and optimal completion time for a job. In this section, we shall prove that the selection policy always selects resource which is better than other available resources in terms of one of the performance criteria, i.e. reliability and completion time.

Proof. Selection of reliable resource:
As per Eq.3, the estimated reliability of the resources for J_i are computed as k_1^i,...k_x^i,...,k_y^i,..,k_n^i (all $k_n^i > 0$). Suppose, k_x^i is the minimum among these values, thus, resource R_x offers minimum reliability. Now if we subtract k_x^i from all k_i's then we have
$(k_1^i - k_x^i) > 0$, ..., $(k_x^i - k_x^i) = 0$, ..., $(k_y^i - k_x^i) > 0$ and $(k_n^i - k_x^i) > 0$.

Selection of faster resource:
Let the predicted completion time of job j_i on these resource are $predCom(i, 1)$, ..., $predCom(i, x)$, ..., $predCom(i, y)$, ..., $predCom(i, n)$, computed based on Eq.4. Suppose, $predCom(i, y)$ is the maximum among these values, i.e. it offers maximum completion time. Next we perform $predCom(i, y) - predCom(i, 1) > 0$, ..., $predCom(i, y) - predCom(i, x) > 0$, ..., $predCom(i, y) - predCom(i, y) = 0$ and $predCom(i, y) - predCom(i, n) > 0$.

Selection of optimal resource:
We will assign job J_i to the resource R_z where $(k_z^i - k_x^i) * ((predCom(i, y) - predCom(i, z))$ is maximum. Our resource selection heuristic always selects a resource which is optimal in terms of at least one of the performance criteria, i.e. either reliability or completion time or both. Now it is clear that resource R_x has minimum reliability and R_y offers maximum completion time. So, these two resources will not be selected for this job, as the product in Eq.5 results into zero.
Suppose R_z is selected as per our selection criteria. Assume that we have a resource R_r that offers better reliability $(k_r^i > k_z^i)$.
So, we can write $k_r^i - k_x^i > k_z^i - k_x^i$ as $k_x^i > 0$.
At the same time, assume that R_r offers faster completion time, i.e. $predCom(i, r) < predCom(i, z)$.
So we can write $predCom(i, y) - predCom(i, r) > predCom(i, y) - predCom(i, z)$
as $predCom(i, y) > predCom(i, r) > predCom(i, z)$ and $predCom(i, y) > 0$.
With the above assumptions, as per our resource selection policy, the following equation must be true.

$$(k_z^i - k_x^i) \times (\text{predCom}(i, y) - \text{predCom}(i, z)) > (k_r^i - k_x^i) \times (\text{predCom}(i, y) - \text{predCom}(i, r))$$

As per our previous assumption, if $k_r^i - k_x^i > k_z^i - k_x^i$, then $(k_r^i - k_x^i)/(k_z^i - k_x^i) > 1$. Based on our selection policy $((predCom(i, y) - predCom(i, z))/((predCom(i, y) - predCom(i, r)) > (k_r^i - k_x^i)/(k_z^i - k_x^i) > 1$
From the above discussion, we can conclude that $((predCom(i, y) - predCom(i, z)) > ((predCom(i, y) - predCom(i, r))$

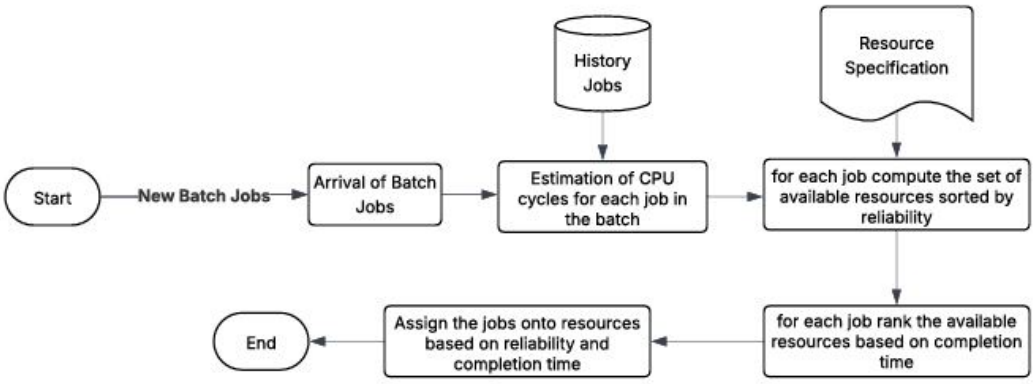

Figure 1. Workflow of the proposed approach

Then, $predCom(i,z) < predCom(i,r)$, which is contradictory to the previous assumption that $predCom(i,r)$ is minimum or in other words, R_r offers faster completion time. So, our selection policy always selects a resource which is better than other available resources in terms of one of the performance criteria. □

Complexity Analysis:

To analyse the complexity of our proposed algorithm, first we will analyse the complexity of algorithm Rank_Reliability$(J_i, R_{\text{available}})$ which is $O(n)$ where n is the number of resources. Similarly, the complexity of the algorithm Rank_CTime$(J_i, R_{\text{available}})$ is $O(n)$, where n is the number of resources. The complexity of Normalize (S) is also $O(n)$. The complexity of line 1-2 of $UncertainityAwareResourceSelection(J,R)$ is $O(nm)$, where n is the number of resources and m is the number of jobs to be scheduled. The complexity of line 5-16 is $O(mn)$. These lines will be executed $O(m)$ times. So overall complexity of our proposed algorithm is $O(m^2n)$ which is similar to the Min-Min and Min-Max scheduling.

IV. EXPERIMENTAL SETUP AND RESULT

To simulate our proposed scheduling algorithm, the modeling of resources and jobs is required. In this section, we represent the implementation details of resource modeling, job modeling and show the experimental results.

Resource Modeling:

We considered a set of resources for the purpose of experimentation. Characteristics of these resources are shown in Table 1. The resources are similar to the resources available from AWS (Amazon Web Services) EC2.

Availability of resources:

To avoid random initialization of available period of resources, we use the following equation to initialize the available period of a resource [29]. Here, parameter A_{factor} is used to set available period of a resource. A higher value of A_{factor} represents a longer available period for a resource.

$$mean\_avail = T_{\text{min}} + A_{\text{factor}} \times (T_{\text{max}} - T_{\text{min}})$$

where T_{min} = Time required to process all jobs in parallel on the fastest resource, and
T_{max} = Time required to process all jobs serially on the slowest resource

Job Modeling:

A set of sample *history jobs* have been extensively used in [3] for clone detection. Whenever a new job arrives in the system, their clone level is detected in comparison with these history jobs. A new job will be either an exact clone, near-miss clone or a miss clone of one or more of these history jobs. The description of history jobs used for our experiments is shown in Table 2.

Experimental Results:

The experiments have been carried out under different circumstances. Here, we have fixed $A_{\text{factor}} = 0.01$ and we varied $varianceET$. Fig 2, Fig 4, Fig 6 show comparison between our proposed algorithm and Min-Min Heuristics for number of successfully completed tasks for different variances (0.15, 0.20, 0.25) in execution time. Fig 3, Fig 5, Fig 7 present comparison between our proposed algorithm vs Min-Min Heuristics for number of failed tasks for different variances (0.15, 0.20, 0.25) in execution time. Now, in order to consider the uncertainty [4] in the execution time of jobs, the execution time of each job on a resource (assumed to follow normal distribution i.e., $N(\mu, \delta)$), where, μ(mean)

corresponds to task execution time on the resource and δ is defined as $Eq(2)$.

ET is the execution time of the job. Here we have considered that $varianceET$ fixed as well as varied as per [4]. In [3], predicted resource requirement (CPU cycles, memory) and actual resource consumption of a job may differ significantly for the near miss clone and miss clone job.

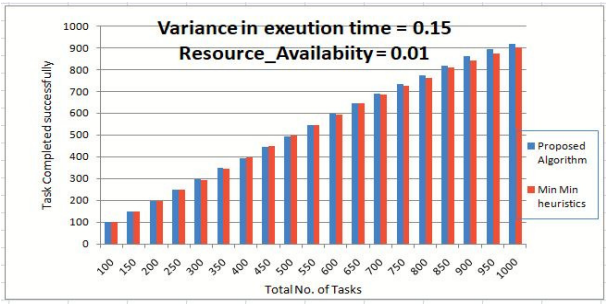

Figure 2. Proposed Algorithm vs MinMin Heuristics for no. of task completed successfully (varianceET $= 0.15$ and $A_{\text{factor}} = 0.01$)

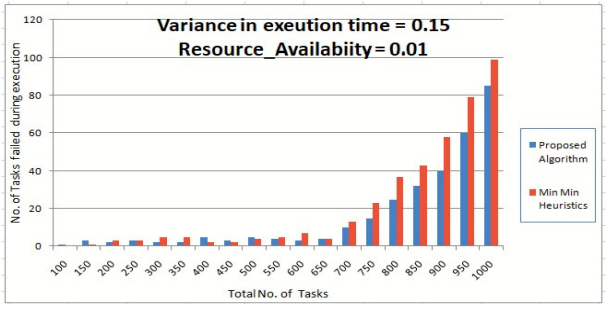

Figure 3. Proposed Algorithm vs MinMin Heuristics for no. of failed tasks ($varianceET$ = 0.15 and A_factor = 0.01)

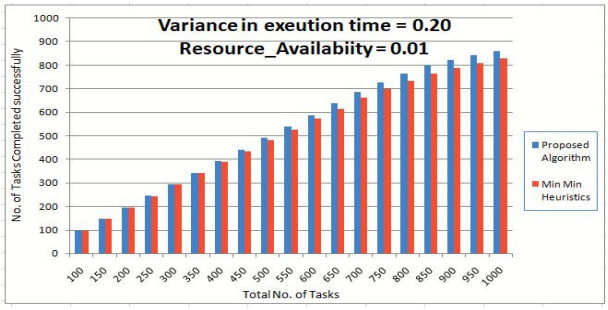

Figure 4. Proposed Algorithm vs MinMin Heuristics for no. of task completed successfully($varianceET$ = 0.20 and A_factor = 0.01)

V. CONCLUSION AND FUTURE WORK

In this paper, we addressed the challenge of efficient job scheduling in distributed cloud environments with fixed

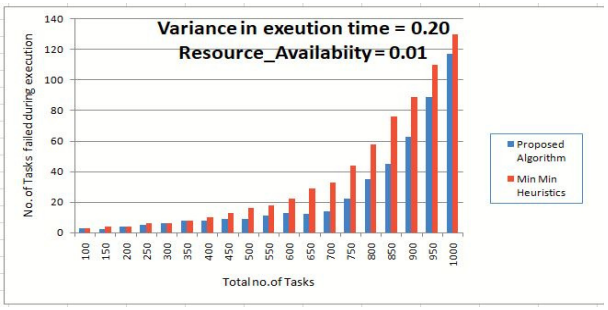

Figure 5. Proposed Algorithm vs MinMin Heuristics for no. of failed tasks ($varianceET$ = 0.20 and A_factor = 0.01)

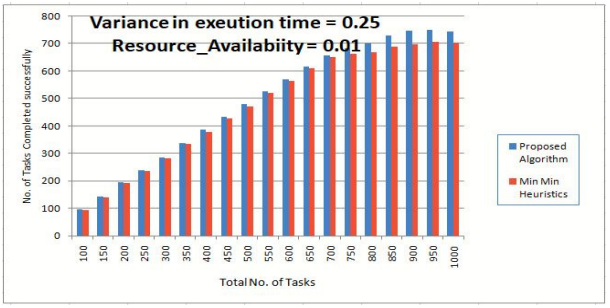

Figure 6. Proposed Algorithm vs MinMin Heuristics for no. of task completed successfully($varianceET$ = 0.25 and A_factor = 0.01)

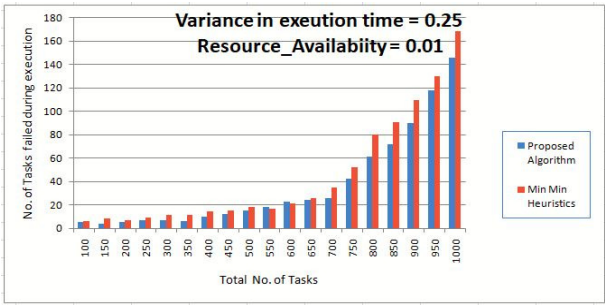

Figure 7. Proposed Algorithm vs MinMin Heuristics for no. of failed tasks ($varianceET$ = 0.25 and A_factor = 0.01)

resource availability periods. Recognizing the limitations of traditional scheduling algorithms in handling uncertain execution times and dynamic resource contracts, we proposed a novel scheduling algorithm that selects resources based on two key parameters: the expected job completion time by the resource and the probability of resource availability until the completion of the job. By incorporating both performance and reliability into the resource selection process, our method significantly reduces the risk of job failure due to premature resource termination. Comparative evaluations with established scheduling algorithms demonstrate that our approach achieves better job success rates and improved overall scheduling efficiency. This work contributes a practical and adaptive strategy for resource-aware scheduling in

Table 1. Description of resources used for simulation purpose

Resource Instance	Operating System	Purpose	Processor Speed	Usage Cost
t3.small	Windows	General Purpose	2.5 GHz Intel	$0.026/hr
t3.small	Windows	General Purpose	2.5 GHz Intel (scalable)	$0.026/hr
C5.12xL	Linux	Compute Optimized	3.6–3.9 GHz	$0.744/hr
C5n.2xlarge	Linux	High Performance Computing	3.0 GHz	$0.137/hr

Table 2. Description of sample history jobs

New Job	Similarity with History Job	Predicted CPU Cycles	Actual CPU Cycle
NJ37	Gauss Elimination	382822	334611.08
NJ39	Matrix Multiplication	6237417473	4122369868
NJ33	Bisection Method	186958	192040
NJ29	Molecular Dynamics Program	56121355302	56786287824
NJ40	Fast Fourier Transformation	1440778794	1374159120
NJ41	Bubble Sort	1479676.332	1637830
NJ26	Jacobi Method	89860	92005
NJ32	Calculation of Factorial	462048	461798
NJ36	Fibonacci Series Calculation	511808	584805
NJ34	Binary Search	68412	69435
NJ44	Matrix Summation	2492529458	2708065275
NJ44	Jacobi Method	89860	92005
NJ27	Calculation of GCD	105856	106572
NJ35	Series Addition	5090547447	4512225937
NJ31	Linear Search	70226	70512
NJ30	Steady StateHeat equation	1.92412E+11	1.92204E+11
NJ28	Inverse of a matrix	85907	91224

dynamic cloud environments and opens avenues for further research into fault-tolerant and predictive scheduling models. As a limitation, this work considers only independent tasks without inter-dependencies. In future research, we aim to extend our scheduling framework to handle cloned dependent tasks, where task dependencies and data flow between jobs will be accounted for to ensure coordinated execution and further enhance scheduling reliability and performance in more complex cloud scenarios.

References

[1] L. Jiang, F. Zhao, and X. Xi, "Matching operation constrained job shop scheduling problem based on backward heuristic scheduling algorithm," 12 2010.

[2] M. Sarkar, S. Roy, and N. Mukherjee, "Feedback-guided analysis for resource requirements in large distributed system," in 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 596–597.

[3] M. Sarkar, T. Mondal, S. Roy, and N. Mukherjee, "Resource requirement prediction using clone detection technique," Future Generation Computer Systems, vol. 29, no. 4, pp. 936–952, 2013, special Section: Utility and Cloud Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X12001835

[4] H. H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," IEEE Transactions on Services Computing, vol. 14, no. 4, pp. 1167–1178, 2021.

[5] T. Braun, H. Siegel, N. Beck, L. Oni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in Proceedings Seventeenth IEEE Symposium on Reliable Distributed Systems (Cat. No.98CB36281), 1998, pp. 330–335.

[6] K. B. Bey, F. Benhammadi, A. Mokhtari, and Z. Guessoum, "Independent task scheduling in heterogeneous environment via makespan refinery approach," in 2010 International Conference on Machine and Web Intelligence, 2010, pp. 211–217.

[7] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," ACM Comput. Surv., vol. 31, no. 4, p. 406–471, Dec. 1999. [Online]. Available: https://doi.org/10.1145/344588.344618

[8] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," Journal of Parallel and Distributed Computing, vol. 61, no. 6, pp. 810–837, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731500917143

[9] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," Journal of Parallel and Distributed Computing, vol. 59, no. 2, pp. 107–131, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731599915812

[10] Z. A. Khan, I. A. Aziz, N. A. B. Osman, and I. Ullah, "A review on

task scheduling techniques in cloud and fog computing: Taxonomy, tools, open issues, challenges, and future directions," IEEE Access, vol. 11, pp. 143 417–143 445, 2023.
[11] O. L. Abraham, M. A. B. Ngadi, J. B. M. Sharif, and M. K. M. Sidik, "Task scheduling in cloud environment–techniques, applications, and tools: A systematic literature review," IEEE Access, vol. 12, pp. 138 252–138 279, 2024.
[12] Y. T. H. Hlaing and T. T. Yee, "Static independent task scheduling on virtualized servers in cloud computing environment," in 2019 International Conference on Advanced Information Technologies (ICAIT), 2019, pp. 55–59.
[13] D. Meiländer, A. Ploss, F. Glinka, and S. Gorlatch, "A dynamic resource management system for real-time online applications on clouds," in Euro-Par 2011: Parallel Processing Workshops. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 149–158.
[14] N. Sasikaladevi, "Minimum makespan task scheduling algorithm in cloud computing," International Journal of Grid and Distributed Computing, vol. 9, pp. 61–70, 11 2016.
[15] M. A. Alworafi, A. Al-Hashmi, A. Dhari, Suresha, and A. B. Darem, "Task-scheduling in cloud computing environment: Cost priority approach," in Proceedings of International Conference on Cognition and Recognition, D. S. Guru, T. Vasudev, H. Chethan, and Y. S. Kumar, Eds. Singapore: Springer Singapore, 2018, pp. 99–108.
[16] R. K. Jena, "Task scheduling in cloud environment: A multi-objective abc framework," Journal of Information and Optimization Sciences, vol. 38, no. 1, pp. 1–19, 2017. [Online]. Available: https://doi.org/10.1080/02522667.2016.1250460
[17] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," IEEE Transactions on Automation Science and Engineering, vol. 15, no. 2, pp. 772–783, April 2018.
[18] M. S. Sanaj and P. M. Joe Prathap, "An enhanced round robin (err) algorithm for effective and efficient task scheduling in cloud environment," in 2020 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA), 2020, pp. 107–110.
[19] Y. Yu and Y. Su, "Cloud task scheduling algorithm based on three queues and dynamic priority," in 2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), 2019, pp. 278–282.
[20] A. Younes, A. Salah, T. Farag, F. Alghamdi, and U. Badawi, "Task scheduling algorithnm for heterogeneous multi processing computing systems," Journal of Theoretical and Applied Information Technology, vol. 97, pp. 3477–3487, 07 2019.
[21] A. Belgacem, K. Beghdad-Bey, H. Nacer, and S. Bouznad, "Efficient dynamic resource allocation method for cloud computing environment," Cluster Computing, vol. 23, no. 4, pp. 2871–2889, 2020.
[22] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A woa-based optimization approach for task scheduling in cloud computing systems," IEEE Systems Journal, vol. 14, no. 3, pp. 3117–3128, 2020.
[23] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of pso task scheduling algorithm in cloud computing," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 6, Part A, pp. 2370–2382, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157820305279
[24] S. Lipsa, R. Dash, N. Ivković, and K. Cengiz, "Task scheduling in cloud computing: A priority-based heuristic approach," IEEE Access, vol. 11, pp. 27 111–27 126, 2023.
[25] Y. Yang, F. Ren, M. Zhang, J. Yan, F. Xie, and W. Gao, "A bdi agent-based asynchronous scheduling framework for cloud computing," in 2024 IEEE International Conference on Agents (ICA), 2024, pp. 1–6.
[26] Q. Li, Z. Peng, D. Cui, J. Lin, and H. Zhang, "UDL: a cloud task scheduling framework based on multiple deep neural networks," Journal of Cloud Computing, vol. 12, no. 1, p. 114, 2023. [Online]. Available: https://doi.org/10.1186/s13677-023-00490-y
[27] T. Selvi Somasundaram, B. R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajendar, R. Rajiv, G. Kannan, G. Rajesh Britto, E. Mahendran, and B. Madusudhanan, "Care resource broker: A framework for scheduling and supporting virtual resource management," Future Generation Computer Systems, vol. 26, no. 3, pp. 337–347, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X09001551
[28] S. Singh, M. Sarkar, S. Roy, and N. Mukherjee, "Genetic algorithm based resource broker for computational grid," Procedia Technology, vol. 10, pp. 572–580, 2013, first International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212017313005598
[29] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," Concurrency and Computation: Practice and Experience, vol. 14, 11 2002.

U ddalok Sen is currently serving as a faculty member in the Department of Information Technology at MCKV Institute of Engineering, 243, G T Road North, Liluah, Howrah, West Bengal 711204 India. With over 16 years of teaching experience, he has developed significant expertise in cloud computing and distributed systems. His research interests focus on advancing the understanding and implementation of cloud architectures and distributed computing paradigms.

M adhulina Sarkar received the PhD degree in computer science from the Jadavpur University, Kolkata, West Bengal, India, in 2016. She is currently serving as Assistant Professor (stage II) in the Department of Computer Science and Engineering, Govt. College of Engineering and Textile Technology, Berhampore, West Bengal, India. She has been served as Head of the Department of Computer Science and Engineering, Govt. College of Engineering and Textile Technology, Berhampore, since 2017. Her research interests focus on distributed computing, cloud computing.

N andini Mukherjee received the PhD degree in computer science from the University of Manchester, UK, in 1999. She was a recipient of a Commonwealth Scholarship for her PhD study in the UK. Since 1992, she has been a Faculty Member with the Department of Computer Science and Engineering, Jadavpur University, Kolkata, India, where she has also been a Professor since 2006. She served as the Director for School of Mobile Computing and Communication, Jadavpur University for nearly six years and Head of the Department of Computer Science and Engineering, Jadavpur University for two years. She is an active researcher in her research areas, which currently include high performance parallel computing, wireless sensor networks and Internet of Things. Prof. Mukherjee is a senior member of the IEEE Computer Society and a member of the ACM.