J Supercomput
DOI 10.1007/s11227-013-0994-8

Bi-level fuzzy based advanced reservation of Cloud workflow applications on distributed Grid resources

Sahar Adabi · Ali Movaghar · Amir Masoud Rahmani

Abstract The increasing demand on execution of large-scale Cloud workflow applications which need a robust and elastic computing infrastructure usually lead to the use of high-performance Grid computing clusters. As the owners of Cloud applications expect to fulfill the requested Quality of Services (QoS) by the Grid environment, an adaptive scheduling mechanism is needed which enables to distribute a large number of related tasks with different computational and communication demands on multi-cluster Grid computing environments. Addressing the problem of scheduling large-scale Cloud workflow applications onto multi-cluster Grid environment regarding the QoS constraints declared by application's owner is the main contribution of this paper. Heterogeneity of resource types (service type) is one of the most important issues which significantly affect workflow scheduling in Grid environment. On the other hand, a Cloud application workflow is usually consisting of different tasks with the need for different resource types to complete which we call it heterogeneity in workflow. The main idea which forms the soul of all the algorithms and techniques introduced in this paper is to match the heterogeneity in Cloud application's workflow to the heterogeneity in Grid clusters. To obtain this objective a new bi-level advanced reservation strategy is introduced, which is based upon the idea of first performing global scheduling and then conducting local scheduling. *Global-scheduling* is responsible to dynamically partition the received DAG into multiple sub-workflows

S. Adabi (✉) · A.M. Rahmani
Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
e-mail: Adabi.Sa@ieee.org

A.M. Rahmani
e-mail: Rahmani@srbiau.ac.ir

A. Movaghar
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
e-mail: Movaghar@sharif.ac.ir

Published online: 17 August 2013

that is realized by two collaborating algorithms: (1) The Critical Path Extraction algorithm (CPE) which proposes a new dynamic task overall critically value strategy based on DAG's specification and requested resource type QoS status to determine the criticality of each task; and (2) The DAG Partitioning algorithm (DAGP) which introduces a novel dynamic score-based approach to extract sub-workflows based on critical paths by using a new Fuzzy Qualitative Value Calculation System to evaluate the environment. *Local-scheduling* is responsible for scheduling tasks on suitable resources by utilizing a new Multi-Criteria Advance Reservation algorithm (MCAR) which simultaneously meets high reliability and QoS expectations for scheduling distributed Cloud-base applications. We used the simulation to evaluate the performance of the proposed mechanism in comparison with four well-known approaches. The results show that the proposed algorithm outperforms other approaches in different QoS related terms.

Keywords Large-scale workflow · Cloud application scheduling · Application graph · QoS · Multi-cluster Grid · Advanced reservation · Fuzzy Inference System

1 Introduction

Today's concept of Cloud computing is extremely used by a High Performance Cloud Computing (HPCC) environment to perform an advanced reservation for large-scale workflow application on high-performance resources which may result in an Infrastructure as a Service (IaaS) delivery model. To fulfill the demands of a large-scale application, a large number of resources have to be available for HPCC. On the other hand, Grid [1, 52] computing infrastructures provide sharing and dynamic allocation of distributed and high-performance computational resources while minimizing the ownership and cost. This makes the Grid an ideal infrastructure for handling the massive amount of computational workloads of Cloud large-scale high-performance applications. Suitable elastic mechanisms for managing Grid clusters enable the Cloud to utilize an environment, which exactly matches its actual demand and workload while eliminates the need to purchase and operate private hardware and software.

Large-scale distributed applications are usually submitted in the form of workflows with certain Quality of Services (QoS) requirements. Some workflows are composed of small and compact tasks but are numerous. Some others consist of heavy and long tasks. Directed Acyclic Graph (DAG) is mostly used by users to express their workflow applications [2, 3].

As the Grid infrastructure is chosen for scheduling Cloud applications, the scheduling of the workflow application focuses on assigning and managing the execution of tasks on Grid resources to satisfy some performance criterion [4]. Many heuristics have been proposed to solve the workflow scheduling problem in Grid environment, but only few researches focus on scheduling Cloud applications in Grids. The heterogeneity and a variety of the available resource types (service types) in each Grid cluster is an issue which dramatically affects the complexity of scheduling workflows. On the other hand, a Cloud application workflow consists of different tasks with the need for different resource types (service types) to complete which

we call it heterogeneity in workflow. The main idea which forms the soul of all algorithms and techniques introduced in this paper is to match the heterogeneity in Cloud application's workflow to the heterogeneity in Grid clusters. Thus, this paper mainly focuses on using the idea of matching heterogeneity between clusters and dynamically extracted sub-workflows to maximize the possibility of scheduling sub-workflows locally. In order to obtain the idea mentioned above, we face two main challenges: (1) to efficiently create sub-workflows from the main one, and (2) to assign tasks of extracted workflows to the appropriate resources. To overcome such problems, we proposed a new system with three main units: Cluster Observation Unit (COU), Cluster Coordination Unit (CCU) and Application Management Unit (AMU).

The AMU focuses on the extraction of sub-workflows from the main workflow regarding critical paths. Unlike the previous works that use the critical path heuristic to minimize the execution time, the AMU is equipped with two new algorithms: (1) the Critical Path Extraction (CPE) algorithm, and (2) the DAG Partitioning algorithm (DAGP). The CPE algorithm extracts critical paths based on the task's Overall Criticality Value (OCV). The OCV indicates the criticality of each task and is calculated by considering DAG specification and the status of the resources with requested resource type. Grid environments are composed of the autonomous and unpredictable resources that can be added to or taken away from the environment continuously without notice, and the status of the resources can be changed in the time. Therefore, task's OCV is dynamically calculated by CPE algorithm.

The DAGP algorithm introduced to extract sub-workflows from the main workflow according to the results of CPE algorithm (i.e., extracted critical paths). The DAGP does this work step-by-step while distributed resources are being reserved. Note that in this paper we use the term "partition" for the sub-workflows extracted by DAGP algorithm. The DAGP extracts the partitions according to both features of the application's main graph and the status of target resources by using the Cluster Resource Type Quality Value (CRTQV) that indicates the suitability degree of a Grid's cluster for executing a specific task. The CRTQV is calculated by Cluster Observation Unit.

The COU is equipped with a new Fuzzy Qualitative Value Calculation System (FQVCS) to evaluate the clusters' suitability for assigning a specific task. This output of FQVCS is the CRTQV value, which is used by DAGP to extract "good" partitions. The FQVCS evaluates each cluster's overall qualitative value according to four parameters: cluster's tasks score density, cluster's availability, cluster's computational power, and cluster's request rate.

The CCU is responsible for scheduling partitions generated by the DAGP. The objective function of this unit is to create a schedule that satisfies the QoS constraints. The Multi-Criteria Advance Reservation (MCAR) algorithm is introduced for this unit to schedule the tasks of each created sub-workflow.

The novel features of this work are the following:

- Introducing a new dynamic mechanism to extract critical tasks from the submitted Cloud application DAG according to DAG specifications and requested resource types (like service type) status. To the best of the authors' knowledge most of the

proposed methods before this work use the parameters originated from application's DAG (for example, computational cost and communication cost) to assign score (that is, priority) to each task [5, 7]. This point of view has been chosen by previous works even though the current and the future QoS status of requested resources by the application's tasks are crucial when we want to determine the criticality of the application's tasks. It is also important to be aware of Grid's dynamic nature in which the status of resources may change. To show the point, assume that current time is 10 and the workload history shows that there may be a workload peak in the time between 11 and 11.50. To decrease the rejection rate of more important tasks in the mentioned time, we have to schedule more critical tasks which requested some resources during the time 11–11.50, more quickly. This example shows that both resource status and task status must be noticed.

- Proposing a new dynamic partitioning algorithm which efficiently extracts sub-workflows according to critical paths by taking into account the data and control dependencies implied by the Cloud application's DAG. In most of the previous literature, static partitioning mechanisms are used for extracting sub-workflows from the submitted DAG. This means that as soon as the DAG is submitted by the user, the partitions (the sub-workflows) are extracted from the DAG before the scheduling starts. The created partitions remain intact during the scheduling. As the static strategy does not suit for Grid environment, we introduce a dynamic partitioning algorithm which considers the dynamic nature of Grid resources. To the best of our knowledge, no other research reflects this requirement.
- Evaluating the quality of the resources of the Grid by utilizing a lightweight linear strategy. We introduce a method to assess the quality of resources which is designed based on weighted combination of multiple QoS related parameters. The following parameters are taken into account for determining Quality of Resource (QoR): resource availability, score density, communication cost, utilization, idle time slots, and job failure rate. These parameters are crucial to determine QoR of each resource and to find the best possible candidate resource for a specific task.
- Evaluating the Grid clusters status by utilizing a new Fuzzy Qualitative Value Calculation System (FQVCS). As performing good schedules need awareness about environment's dynamic changes, we propose a new method that assesses the quality of each cluster as a combination of multiple QoS related parameters of the resources of the cluster. The cluster's availability, computation power, request rate, and a new parameter which we call the score density, are included in the proposed cluster's quality assessment method. The score density reflects the current and midterm workload of each resource in a more realistic fashion compared with classic workload concept.

The remainder of the paper is organized as follows. Section 2 reviews related works. Section 3 introduces the mathematical system model and Sect. 4 describes the assumption and the related terminology. In Sect. 5 the proposed system is described in detail including all units, the proposed fuzzy evaluation system and all the introduced algorithms. Comparative performance evaluation of the proposed system, which is performed by software simulation is presented in Sect. 6 and in Sect. 7 the research summary and future directions are described.

2 Related work

Scheduling is a process that assigns and monitors the execution of tasks on distributed resources. Various heuristics are proposed to solve the problem of task scheduling in Grid environment. Most of them consider single QoS parameter (for example, the makespan [5–7, 43, 48] of tasks) and few of them consider multiple QoS parameters.

In [5–7] scheduling methods that imply the task's priority concept are proposed. In this concept the priority is assigned to each task. Tasks are selected according to their priority and assigned to the processor which satisfies the tasks predefined requirements. In [5] the priority can be calculated by considering the pair of available processor and ready node without considering the communication costs. In Dynamic Critical Path (DCP) algorithm [6], critical tasks will be selected and assigned to appropriate resources. The lower bound and upper bound of the task's start time are the two parameters which determined the task's priority. The Heterogeneous Earliest Finish Time (HEFT) [7] algorithm calculates the priority of each task regarding two attributes, communication cost and computational cost. These algorithms perform the static scheduling, which assigns the task's priority at the beginning of the scheduling process. They also neglect the dynamic nature of Grid in calculation task's priority.

Clustering algorithms are categorized as one of the most important classes of large-scale application scheduling algorithm [8–12, 18–23] which is categorized in an NP-complete [17] problem. The communication volumes significantly decrease by using this method in Grid environment [16]. This method efficiently schedules large-scale applications according two phases. In the first phase it generates a group of tasks and in the second phase it assigns the tasks of generated groups to the same resources. To minimize the schedule length, Cheng and Zeng [8] proposed a dynamic partitioning approach. To increase the flexibility, Tan and Fan [9] proposed the dynamic workflow partitioning model. A partitioning method that partitions the parallel tasks on homogeneous cluster is introduced in [19]. Pegasus [20–22] statically groups the tasks into clusters to be executed as a single task. This method is not performing well in migrating workflows. Dong and Akl [23] proposed a two-layered workflow scheduling which uses a look-ahead technique. To minimize the completion time Wong and Ahmad [18] proposed a clustering algorithm with trade-off between the concurrency and communication. Although these workflow partitioning methods consider different aspects, they are not efficiently scheduling large-scale workflow applications. All the proposed algorithms create clusters at the beginning of the scheduling process and the status of the resources and tasks does not consider in partitioning tasks of graph. On the other hand, most of the proposed algorithms in this category consider the same resource to schedule the group of tasks, located in the same cluster. This may cause a major problem when most of the important tasks are assigned to unreliable resource (e.g., rescheduling).

Workflow scheduling with QoS is addressed in some research [24, 25, 42, 51]. A two-phased scheduling algorithm proposed by Prodan and Wieczorek [24] is scheduling the tasks according to three parameters: the primary criterion, the secondary criterion and the sliding constraints. All three parameters are determined by the users. A game-theory-based approach introduced in [25] minimizes the overall makespan and overall cost of a workflow, while meeting each workflow's deadline.

In [42] a scheduling model which combines the benefits of both, resource provider and consumer, is proposed. Luo et al. [51] proposed architecture for advance reservation which adapts with dynamic nature of the Grid and guarantees the predefined user's QoS.

Some approaches use meta-heuristic methods to handle task scheduling problem [13–15, 26–28, 50]. An ant colony algorithm, which considers three QoS parameters (i.e., time, cost and reliability), is proposed in [27]. A particle swarm optimization approach proposed by Tao et al. [28] optimizes the weighted sum method for multi-parameter QoS. In [13] a two-phase algorithm which implies the genetic algorithm is introduced to schedule tasks on heterogeneous distributed computing systems. A genetic algorithm approach is introduced by Omara et al. [14] for task scheduling problem. Two genetic algorithms are proposed which the main contribution of the first algorithm is to minimize the total execution time and the second one is to concern the load balance satisfaction. A novel contention-aware task duplication scheduling algorithm proposed in [15]. In order to manage the Grid resources, Cao et al. [53] proposed a multi-agent approach in global and local scheduling. In spite of a good performance of meta-heuristic methods, they are usually more time-consuming than other heuristics.

Workflow scheduling on the Cloud is also addressed in some research [29–34, 41, 49]. In order to execute deadline and budget constrained applications, two scheduling methods, time optimization and cost optimization, are proposed in [29]. The idea behind the time optimization policy is to minimize the schedule length by hiring resources from a Cloud provider. If the cost optimization policy cannot meet the predefined deadline, it hires more resources. However, these proposed algorithms do not consider workflow applications. A particle swarm optimization approach which minimizes the execution cost of a workflow application on the resources of Cloud is introduced in [30]. To schedule the multiple workflows which require multiple QoS constraints, Xu et al. proposed an algorithm in [31, 32]. Byun et al. proposed a scheduling algorithm in [33] which executes an application in a predefined deadline while estimating the minimum number of resources required executing the application. They also extended their algorithm in [34] by considering the pricing policy of the Clouds while meeting the user predefined deadline. The problem of finding an appropriate service satisfying the users' multiple QoS requirements is addressed in [41] by considering three QoS parameters: service's response time, trust degree and monetary cost. Li and Li [49] proposed an optimal Cloud resource-provisioning algorithm by applying two new algorithms which consider the QoS parameters.

3 The large-scale workflow scheduling system model

The proposed large-scale workflow scheduling system model consists of a Cloud application model and a Grid resource model. A Cloud application is modeled by a Directed Acyclic Graph (DAG), $G = (V, E, w, c)$, in which V and E are representing tasks and the communication between them, respectively. Let $V = \{t_i | i = 1, 2, \ldots, m\}$ and E be the set of form $\langle t_i, t_j \rangle$, where t_i is called a prior of t_j, and t_j is a successor of t_i. An edge $e_{ij} \in E$ represents the communication from t_i to t_j. The

non-negative weight $w(t, \mathrm{RT}_x)$ represents the computational cost of task $t \in V$ on resource of type x where $\mathrm{RT}_x \in \mathrm{RT}$ and the weight $c(e_{ij})$ represents the communication cost of edge $e_{ij} \in E$. The set $\{t_x \in V : e_{ix} \in E\}$ off all direct successors of t_i is denoted by succ_{t_i}. If the task is without successors, it is named a sink task ($\mathrm{succ}_{t_{\mathrm{sin}k}}$).

The Grid model denotes the set of available clusters, that is $\mathrm{CL} = \{\mathrm{Cl_{id}}|\mathrm{id} = 1, 2, \ldots, n\}$, where id is the unique identification number of the cluster, and n is the number of clusters. The $\mathrm{GRS} = (\mathrm{gr}_{ik})_{m\times n}$ represents a matrix, where $\mathrm{gr}_{ik} = 1$ expresses that task t_i can be assigned to cluster Cl_k; otherwise, $\mathrm{gr}_{ik} = 0$. Therefore, the preconditions that a workflow can be assigned to the Grid is $\forall t_i \in V, \exists \mathrm{Cl}_k \in \mathrm{Grid} \bullet \mathrm{gr}_{ik} = 1$.

A cluster is a service site defined as $\mathrm{Cl}_i = (\{\overline{\mathrm{TSD}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}\}, \{\overline{\mathrm{AVL}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}\}, \{\overline{\mathrm{CP}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}\}, \{\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}\})_{X\in \mathrm{ResourceTypes}(\mathrm{Cl}_i)}$, where $\overline{\mathrm{TSD}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}$ denotes the average summation of scores density of the tasks which are currently assigned to the resources with type x; $\overline{\mathrm{AVL}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}$, $\overline{\mathrm{CP}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}$ and $\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT}_x}$ denote respectively the average availability, average computation cost and average resources request rate of cluster resources with type x.

The actual start time and actual finish time of task $t_j \in V$ on $R_x \in R$ are denoted by $\mathrm{AST}^{R_x}_{t_j}$ and $\mathrm{AFT}^{R_x}_{t_j}$, respectively. The estimated computation cost of task t_j on resource R_x is shown as $w(t_j, R_x)$ and is calculated as:

$$w(t_j, R_x) = \frac{t_j.\mathrm{Length}}{R_x.\mathrm{CyclePerSecond}} \tag{1}$$

The following conditions for all tasks are mandatory:

- The resource constraint requires that only one task can be executed by a resource at any given time, which means that, for two tasks $t_i, t_j \in V$ [35],

$$\text{If } \left(R(t_i) = R(t_j)\right) \text{ then } \left\{\mathrm{AFT}^{R_x}_{t_i} \leq \mathrm{AST}^{R_x}_{t_j} \text{ or } \mathrm{AFT}^{R_x}_{t_j} \leq \mathrm{AST}^{R_x}_{t_i}\right\} \tag{2}$$

- The precedence constraint is shown in Eq. (3). This constraint requires that, for tasks $t_i, t_j \in V$ the execution of the destination task t_j can only be started after the communication associated with t_i and t_j (shown as e_{ij}) has arrived at t_j's resource [35]:

$$\mathrm{AST}^{R_x}_{t_j} \geq \mathrm{AFT}^{R_y}_{t_i} + \frac{e_{\mathrm{ij}}}{\mathrm{EstimatedBandwidthBetween}(R_x.\mathrm{Cluster}, R_y.\mathrm{Cluster})} \tag{3}$$

4 Assumptions

The following assumptions are considered in proposed system model.

Assumption 1 The cluster-to-cluster bandwidths are bilateral. If there is a connection between cl_i and cl_j, the cl_i can send data to cl_j and vice versa.

Assumption 2 High-speed links used to communicate between different resources within clusters. This means that the communication cost between resources which are both in a certain cluster is considered zero.

Assumption 3 Each task (a node of a DAG) requests one specific service (resource type).

Assumption 4 There is no central terminal submitting Cloud application, and therefore users can submit their tasks to each of the clusters.

Assumption 5 A sub-workflow is a subset of the original DAG. The properties of all tasks of a sub-workflow are similar to the original DAG.

5 The proposed system

Grid users often submit their applications in the form of workflows with certain requirements of QoS. Some workflows are composed of small and compact tasks but are numerous. Some others consist of heavy and long computational tasks which require thousands of hours of a total CPU time. The later type of workflows are usually categorized as the large-scale workflows [47]. The proposed system addresses the scheduling of large-scale applications on the distributed and heterogeneous resources of the Grid. Figure 1 illustrates the general view of the system units. As can be seen in Fig. 1, the proposed system consists of three main units as follows.

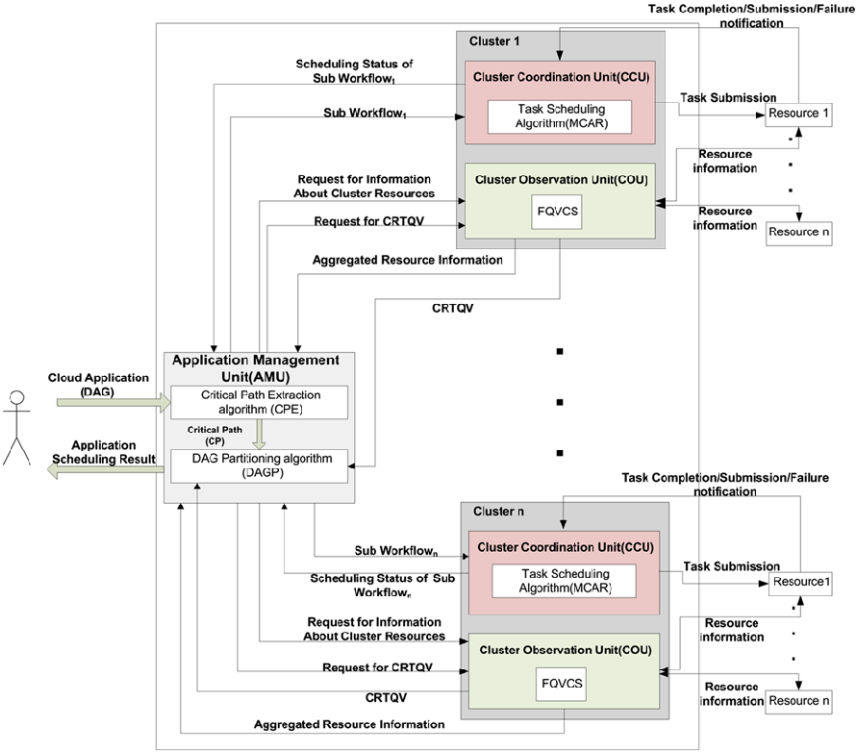


Fig. 1 A general architectural view of the system

- *Cluster Observation Unit*. Each cluster has a Cluster Observation Unit (COU), which investigates the resources' status and calculates the Cluster Resource Type Quality Value (CRTQV) of each resource type of the cluster. The CRTQV is used by AMU to extract sub-workflows from the main application DAG. Calculating the CRTQV value classifies in the family of multi-objective optimization problems, and to transform it to a single-objective problem, a new fuzzy approach is proposed. This unit is described in detail in Sect. 5.1.
- *Application Management Unit.* For each application there is a specific Application Management Unit (AMU) which is responsible for performing Global-level scheduling. Global-level scheduling is the process that extracts sub-workflows from the main workflow based on two novel algorithms: (1) Critical Path Extraction algorithm (CPE) which extracts critical paths, and (2) DAG Partitioning algorithm (DAGP) which creates sub-workflows according to the critical paths generated by CPE. In this unit, the critical path concept extensively is used to reach high degrees of parallelism both in schedule generation and application execution. This unit is described in Sect. 5.2.
- *Cluster Coordination Unit*. Each cluster has a Cluster Coordination Unit (CCU) which is responsible for performing Local-level scheduling. Local-level scheduling is the process that schedules the prioritized tasks on cluster's resources. The CCU uses a new advance reservation mechanism called Multi-Criteria Advance Reservation (MCAR) to select the best resource, which meets the QoS constraints. This unit described in Sect. 5.3.

5.1 Cluster Observation Unit (COU)

Each COU uses a Fuzzy Inference System called Fuzzy Qualitative Value Calculation System (FQVCS) to calculate CRTQV values for its corresponding Grid cluster for a particular task. The remainder of this section describes the FQVCS in detail. The COU hosts the FQVCS_FIS function and responds to the requests of an AMU (specifically DAGP function) about CRTQV value of the cluster. Algorithm 1 describes the details of FQVCS_FIS function.

5.1.1 Fuzzy Qualitative Value Calculation System (FQVCS)

The distinguishing feature of the proposed approach is assigning qualitative values to each Grid's computational cluster. For each resource type in a cluster, the Fuzzy Qualitative Value Calculation System (FQVCS) calculates Cluster Resource Type Quality Value (CRTQV) by using a novel Fuzzy Inference System (FIS). Cluster's Quality Value Array (CQVA) is a series of CRTQV values (calculated by FQVCS) which reflects the worth of the cluster's resources.

Notions about parameters that make numerical value of CRTQV (that is, CRTQV_value) are vague and uncertain to be expressed by clear mathematical models. However, it is often possible to describe the CRTQV_value by means of building fuzzy models. As a common source of information for building fuzzy models is the knowledge of expert, we used this approach for designing and developing the FIS of the COU.

A Fuzzy decision controller [37] is composed of the following parts:

Algorithm 1 FQVCS_FIS function

Input: t as Task
Output: float CRTQV

```
float FQVCS_FIS (Task T){
1:  float TSD,AVL,CP,RR;
2:  TSD=Calculate_TSD(thisCluster, t. ResourceType);      //using Eq. (5)
3:  AVL=Calculate_AVL(thisCluster, t. ResourceType);      //using Eq. (6)
4:  CP=Calculate_CP(thisCluster, t. ResourceType);        //using Eq. (7)
5:  RR=Calculate_RR(thisCluster,T);                       //using Eq. (8)
6:  Float CRTQV=FIS(TSD,AVL,CP,RR);
                    //using Fuzzy Inference System described in Sect. 5.1
7:  Return CRTQV;
}   //End Function
```

(1) Input and output variables, which are usually determined based on knowledge of experts. The input and output variables of the FQVCS are discussed in Sect. 5.1.2.
(2) Fuzzification Interface (FI), which has the effect of transforming clear values of input variables to fuzzy sets. The FI of the FQVCS, which includes the membership functions for the input variables, is illustrated in Sect. 5.1.3.
(3) Fuzzy rule base (RB), in which a set of fuzzy rules is determined. These rules are used for the fuzzy inference process. The RB of the FQVCS is illustrated in Sect. 5.1.4.
(4) Defuzzification interface (DFI) that translates the output of the fuzzy inference process, from fuzzy linguistic values to a clear real number by using a defuzzification method. The DFI of the FQVCS, which includes the membership function for the output variable, is illustrated in Sect. 5.1.3.

In the following, the four parts of the FQVCS are discussed.

5.1.2 Input and output variables of the FQVCS

Output The FQVCS has one output $\mathrm{CRTQV}_{\mathrm{Cl}_i}^{t_x}$ (i.e., degree of suitability of assigning task t_x to the cluster Cl_i). Note that $\mathrm{CRTQV}_{\mathrm{Cl}_i}^{t_x} = \mathrm{CRTQV}_{\mathrm{Cl}_i}^{(\mathrm{RT_A},[T_S,T_E])}$ where $\mathrm{RT_A}$ is the resource type, which is requested by t_x, and $[T_S, T_E]$ is the time slot indicated by start time and finish time of task t_x. Start time and finish time of a task are calculated by the estimation method described in Sect. 5.2.1 (Eqs. (17)–(20)).

Input set Four qualitative criteria that can influence a decision in determining CRTQV of each cluster include: (a) average cluster's assigned tasks score density ($\overline{\mathrm{TSD}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}$), (b) average cluster's availability ($\overline{\mathrm{AVL}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}$), (c) average cluster's computation power ($\overline{\mathrm{CP}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}$), and (d) average cluster's request rate ($\overline{\mathrm{RR}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}$). All parameters mentioned above should be calculated for a specific resource type, which is requested by task t_x.

In the following, the four mentioned qualitative criteria are discussed in detail.

i. *Average cluster's assigned tasks' score density* ($\overline{\text{TSD}}^{\text{Cl}_i}_{\text{RT}_\text{A}}$). In most of previous works, more computationally powerful resources are selected more frequently for important tasks as these resources are able to complete the assigned tasks more quickly. On the other hand, there is a fact that is usually neglected in these approaches: the failure of a resource which hosts too many important tasks will cause a significant loss and may lead to massive application failures as rescheduling of these important tasks (and usually huge and computation-intensive) may not be possible in a timely fashion to prevent application failure. To address this issue a new parameter is introduced under the name tasks' score density. The score density of resource m can be calculated from Eq. (4) and it is simply the summation of the Overall Critically Values (OCV) of all the tasks which are currently assigned to the resource m. Note that the OCV of a task, as calculated by Eq. (22), reflects the importance of that task in its application DAG.

$$\text{TSD}^{\text{Cl}_i}_{R_m} = \sum_{t_i \in k} \text{OCV}(t_i) \tag{4}$$

where k is the set of tasks which are currently scheduled on the resource m of cluster Cl_i and their start time is equal or greater than current time (i.e., scheduled for execution in the future). Having this definition, we can compute $\overline{\text{TSD}}^{\text{Cl}_i}_{\text{RT}_\text{A}}$ using Eq. (5)

$$\overline{\text{TSD}}^{\text{Cl}_i}_{\text{RT}_\text{A}} = \frac{1}{n} \sum_{R_j \in \text{RT}_\text{A}} \text{TSD}^{\text{Cl}_i}_{R_j} \tag{5}$$

where RT_A is the set of all resources of type A in cluster Cl_i, and n is the count of resources in RT_A.

It is obvious that a cluster with a higher TSD is a more failure-sensitive cluster (as it is responsible for more important tasks). The TSD also reflects the current and midterm workload of the clusters in a more realistic fashion compared with classic workload concept [39].

ii. *Average cluster availability* ($\overline{\text{AVL}}^{\text{Cl}_i}_{\text{RT}_\text{A}}$). Dynamic nature of Grid may lead to poor scheduling, as the requested resource may be unavailable when the task is ready to run. Therefore, availability is one of the most effective parameters in efficient scheduling of Grid workflow applications. Availability represents the probability that a cluster's resources can be contacted to be consumed at a given time and is usually defined by Eq. (6):

$$\overline{\text{AVL}}^{\text{Cl}_i}_{\text{RT}_\text{A}} = \frac{1}{n} \left(\sum_{R_j \in \text{RT}_\text{A}} \frac{R_j.\text{Up time}}{L} \right) \tag{6}$$

where R_j. Uptime is the fraction of time in the period L in which the resource j has been up and intact. Note that higher availability value for a resource type of a cluster indicates that the resources of that type are more intact and accessible.

iii. *Average cluster computation power* ($\overline{CP}^{\text{Cl}_i}_{\text{RT}_\text{A}}$). As discussed in [40], when a choice is to be made between two resources, it is better to highly utilize the fast resource rather than the slow resource because the fast resource computes many more operations in the given time than the slow one. According to this logic, the

cluster with higher average computation power value is more suitable for assigning submitted workflow to it. The average computation power for the resources of type A in cluster Cl_i is calculated by Eq. (7)

$$\overline{\mathrm{CP}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}} = \frac{1}{n}\left(\sum_{R_j \in \mathrm{RT_A}} R_j.\mathrm{ComputationPower}\right) \tag{7}$$

iv. *Average cluster resource type request rate* ($\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}$). Grid environment is composed of a number of shared resources which shape a powerful computational system. In most of the time the fraction of provided resources to resource consumers is less than one. Therefore, a competition will form to achieve resources. As the request rate of specific resources with specific resource type (for instance, $\mathrm{RT_A}$) in cluster (for instance, Cl_i) increases, the chance of acceptance of new requests will decrease. A task t_i requires a resource type in a specific time $[T_S, T_E]$, where T_S and T_E are the estimated start time ($\mathrm{EST}(t_i)$) and estimated finish time ($\mathrm{EFT}(t_i)$) of task t_i, respectively. The request rate of resource j with type A in cluster Cl_i in time slot $[T_S, T_E]$ is denoted as $\mathrm{RR}^{\mathrm{Cl}_i}_{R_j}(T_s, T_E)$. Therefore, the average request rate for resource type A in cluster Cl_i based on estimated execution time of task t_i (that is, $\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}(\mathrm{EST}(t_i), \mathrm{EFT}(t_i))$ is calculated by Eq. (8):

$$\begin{aligned}
&\mathrm{RR}^{\mathrm{Cl}_i}_{R_j}(T_s, T_E) = R_j.\mathrm{RequestRate}(T_s < \mathrm{Time} < T_E)\\
&\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}(T_s, T_E) = \frac{1}{n}\sum_{R_j \in \mathrm{RT_A}} \mathrm{RR}^{\mathrm{Cl}_i}_{R_j}(T_s, T_E)\\
&\text{for task } t_i : \overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}(\mathrm{EST}(t_i), \mathrm{EFT}(t_i)) = \overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}(T_s, T_E)
\end{aligned} \tag{8}$$

where T_S and T_E determine the boundaries of the time when the request rate should be calculated. Note that the advanced reservation on Grid resources provides us with sufficient information to calculate request rate of each resource in a given time in the future. This is made possible as each CCU has a request rate table which contains entries for each reservation performed on the resources. This table is updated every time when a new advanced reservation is performed successfully on a resource of the cluster. Although this calculation may have some inaccuracy issues due to further events, those that may happen to the resource in the future (before the real time reaches the beginning of the desired time), it can reflect a good approximation of what the request rate will be in the desired time in the future.

5.1.3 Fuzzification and defuzzification interface

Fuzzy inference is the process of formulating the mapping from a given input set to an output using fuzzy logic [36]. The basic elements of fuzzy logic are fuzzy rules, linguistic variables, and fuzzy sets. The values of the linguistic variables are adjectives like "small," "medium," and so on. The membership degree (a number between 0 and 1) can be obtained by membership function. A curve or linear shape can be used to express a membership function. In the following we discussed the

input and output variables and the membership functions that are used to assign the degree of membership for these variables of the FQVCS.

All membership functions and their graphical representations are illustrated in Table 1 and Fig. 2, respectively. We drew the membership function illustrated in Fig. 2 according to the knowledge of the experts in fuzzy toolbox of Matlab. According to the graphical presentation of membership functions we extracted their corresponding equations as given in Eqs. (9)–(13).

- *Fuzzy values of input variables*:
  i. *Average cluster's Tasks Score Density* ($\overline{\mathrm{TSD}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}$). Three fuzzy sets are defined for this input variable: {*R* (relax), *M* (moderate), *C* (critical)}. Membership functions in Table 1 (Eq. (9)) are used for assigning the membership degree of each clear value of this variable.
  ii. *Average cluster's Availability* ($\overline{\mathrm{AVL}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}$). Three fuzzy sets are defined for this input variable: {*A* (Attainable), *M* (Moderate), *D* (Devastate)}. Membership functions in Table 1 (Eq. (10)) are used for assigning the membership degree of each clear value of this variable.
  iii. *Average cluster's Computation Power* ($\overline{CP}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}$). Three fuzzy sets are defined for this input variable: {*W* (Weak), *M* (Moderate), *S* (Strong)}. Membership functions in Table 1 (Eq. (11)) are used for assigning the membership degree of each clear value of this variable.
  iv. *Average cluster's resource type Request Rate* ($\overline{\mathrm{RR}}^{\mathrm{Cl}_i}_{\mathrm{RT_A}}$). Three fuzzy sets are defined for this input variable: {*S* (Sparse), *N* (Normal), *B* (Busy)}. Membership functions in Table 1 (Eq. (12)) are used for assigning the membership degree of each clear value of this variable.
- *Fuzzy values of output variable*:
  i. *CRTQV*. Five Fuzzy sets are defined for this output variable: {*E* (Excellent), *G* (Good), *M* (Moderate), *B* (Bad), *W* (Worst)}. Membership functions in Table 1 (Eq. (13)) are used for assigning the membership degree of each clear value of this variable.

The outputs of the FQVCS used for calculating CRTQV are plotted in Figs. 3, 4, 5 and 6, against availability, computation power, resource request rate and tasks score density in different input parameters conditions. The weighted average method [36] is used for defuzzification and computing the clear output value of FQVCS_FIS.

5.1.4 Fuzzy rule base (RB)

The available knowledge about the problem is stored in the form of linguistic "*if-then*" rules as shown in Table 2. These rules guide the system behavior. A fuzzy "*if-then*" rule is able to capture the usual decision making of humans by utilizing linguistic labels and membership functions. On the other hand, each fuzzy "*if-then*" rule defines behavioral dynamic of the target system. In the following, the interpretations of two sample rules of Table 2 are presented:

Rule 5: ***if*** (***TSD*** *is* ***Moderate***) *and* (***Avl*** *is* ***Moderate***) *and* (***CP*** *is NOT* ***Weak***) *and* (***RR*** *is NOT* ***Busy***) ***then*** (***CRTQV*** *is* ***Bad***)

Rule 20: ***if*** (***Avl*** *is* ***Devastated***) ***then*** (***CRTQV*** *is* ***Bad***)

Table 1 Equations of the membership functions of the FQVCS_FIS

Input variable	Membership function	Linguistic value
TSD (Eq. (9))	$\mu(x) = \frac{1}{1+\lvert\frac{x-0.06}{0.24}\rvert^{6.8}}$	*R* Relax
	$\mu(x) = e^{\frac{-(x-0.047)^2}{0.02}}$	*M* Moderate
	$\mu(x) = \frac{1}{1+e^{-26\times(x-0.8)}}$	*C* Critical
Avl (Eq. (10))	$\mu(x) = \frac{1}{1+\lvert\frac{x-1.1}{0.22}\rvert^{10}}$	*A* Attainable
	$\mu(x) = \begin{cases} e^{\frac{-(x-0.56)^2}{2\times 0.045^2}} & x \le 0.56 \\ 1 & 0.56 \le x \le 0.73 \\ e^{\frac{-(x-0.73)^2}{2\times 0.07^2}} & x \ge 0.73 \end{cases}$	*M* Moderate
	$\mu(x) = \begin{cases} 1 & 0 \le x \le 0.38 \\ \frac{0.54-x}{0.16} & 0.38 < x \le 0.54 \\ 0 & x > 0.54 \end{cases}$	*D* Devastate
CP (Eq. (11))	$\mu(x) = \frac{1}{1+\lvert\frac{x+0.35}{0.5}\rvert^{10}}$	*W* Weak
	$\mu(x) = \frac{1}{1+\lvert\frac{x-0.5}{0.16}\rvert^{3.4}}$	*M* Moderate
	$\mu(x) = \frac{1}{1+\lvert\frac{x-1.25}{0.48}\rvert^{13.2}}$	*S* Strong
RR (Eq. (12))	$\mu(x) = \frac{1}{1+\lvert\frac{x+0.14}{0.33}\rvert^{5.6}}$	*S* Sparse
	$\mu(x) = \frac{1}{1+\lvert\frac{x-0.52}{0.13}\rvert^{5.8}}$	*N* Normal
	$\mu(x) = \frac{1}{1+\lvert\frac{x-1}{0.28}\rvert^{7}}$	*B* Busy
Output variable	**Membership function**	**Linguistic value**
CRTQV (Eq. (13))	$\mu(x) = \text{Max}(\text{Min}(\frac{x-0.8}{0.95-0.8}, 1), 0)$	*E* Excellent
	$\mu(x) = \text{Max}(\text{Min}(\frac{x-0.53}{0.67-0.53}, 1, \frac{0.87-x}{0.87-0.84}), 0)$	*G* Good
	$\mu(x) = \text{Max}(\text{Min}(\frac{x-0.28}{0.423-0.28}, 1, \frac{0.7-x}{0.7-0.5}), 0)$	*M* Moderate
	$\mu(x) = \text{Max}(\text{Min}(\frac{x-0.14}{0.16-0.14}, 1, \frac{0.39-x}{0.39-0.22}), 0)$	*B* Bad
	$\mu(x) = \text{Max}(\text{Min}(1, \frac{0.2-x}{0.2-0.05}), 0)$	*W* Worse

5.2 Application management unit (AMU)

The AMU unit generally is responsible for the following duties:

- Calculating the criticality of each task by applying the proposed Overall Criticality Value (OCV) method
- Extracting critical paths from the main workflow
- Creating sub-workflows from the extracted critical paths

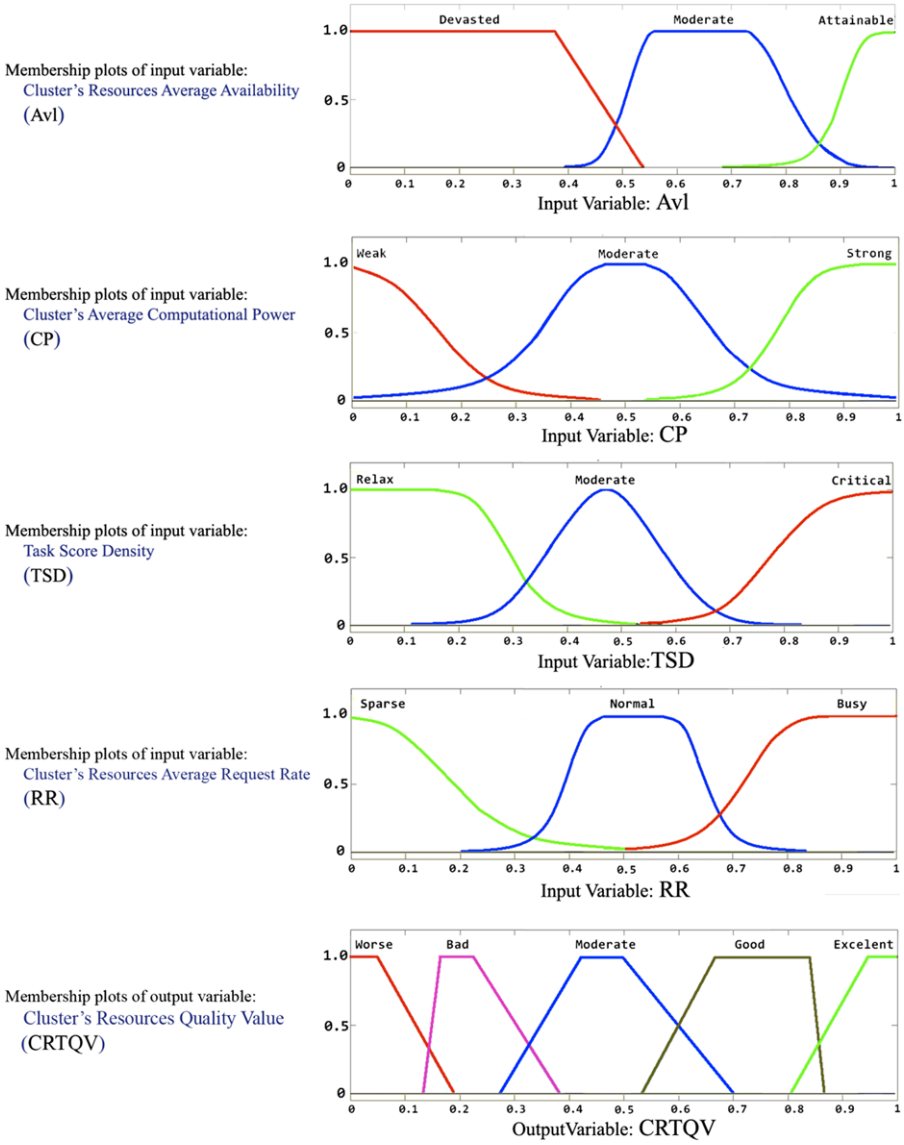


Fig. 2 Graphical presentation of the membership functions of the FQVCS_FIS

- Scheduling the sub-workflows (by sending the sub-workflows to target Cluster Coordination Unit)

This unit uses two proposed algorithms, the Critical Path Extraction algorithm (CPE) and the DAG Partitioning algorithm (DAGP) to accomplish its mission.

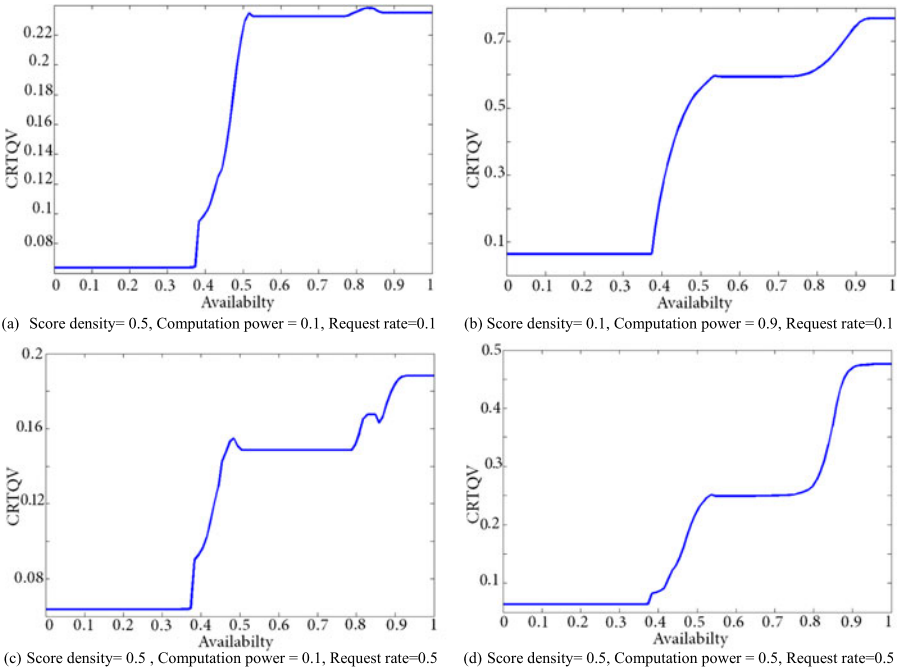
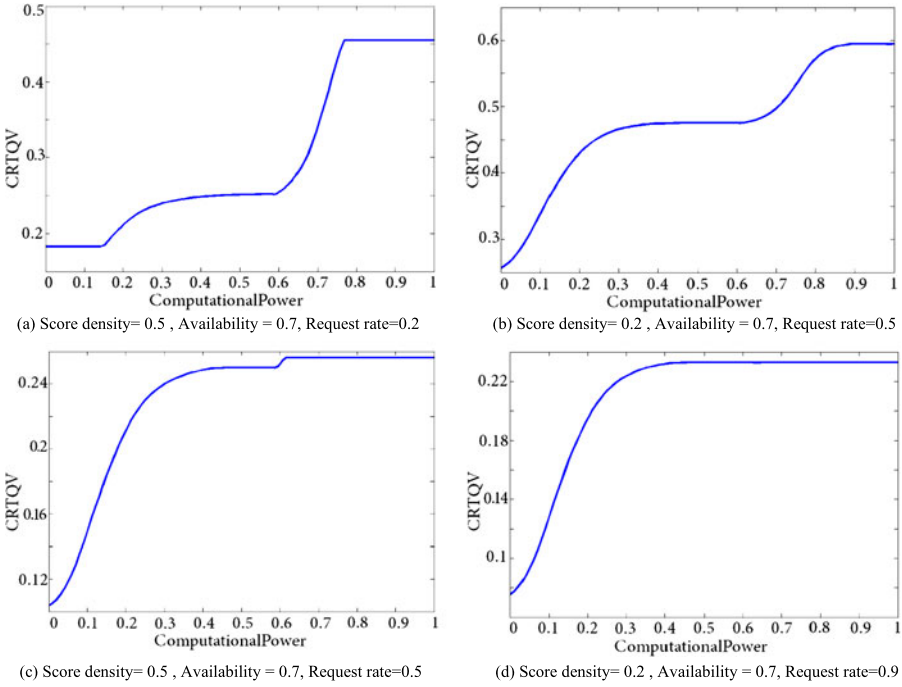


Fig. 3 Outputs of the FQVCS for availability in different score density, computation power and request rate values

5.2.1 Critical path extraction algorithm

Identifying critical tasks is an important key in efficient DAG scheduling. In most literature [6, 23] the critical path is the longest path from entry task to an exit task (or the longest path from current task to an exit task or to an entry task). Note that usually the Critical Path (CP) contains the critical tasks. In most previous literature for identifying the critical tasks only the computation and communication cost of tasks are considered and the dynamic nature of Grid resources is neglected [6, 7]. On the other hand, most of the previous literature used static rank strategy to identify the critical tasks [7, 23]. Static rank strategy calculates the tasks' values at the beginning of the scheduling and applies these critical tasks to identify critical paths.

Instead of using a static rank value computed at the beginning of the scheduling process, the CPE algorithm adopts a dynamic ranking strategy which enables making decisions based on most current status of resources regarding both application graph and Grid resources. The CPE assigns an Overall Criticality Value (OCV) to each task in a given Cloud application DAG as its rank. This value may be updated as the scheduling process proceeds to reflect the most recent state of the application scheduled nodes and Grid resources status. In the calculation of the OCV value, some parameters are in conflict with each other, and according to the definition of the Multi-Criteria Decision Making (MCDM) problems, the OCV calculation is a member of the family of the MCDM problems. Three approaches are introduced for solving the MCD problems: (1) Multiple Attribute Utility Theory (MAUT), (2) out-



(a) Score density= 0.5 , Availability = 0.7, Request rate=0.2

(b) Score density= 0.2 , Availability = 0.7, Request rate=0.5

(c) Score density= 0.5 , Availability = 0.7, Request rate=0.5

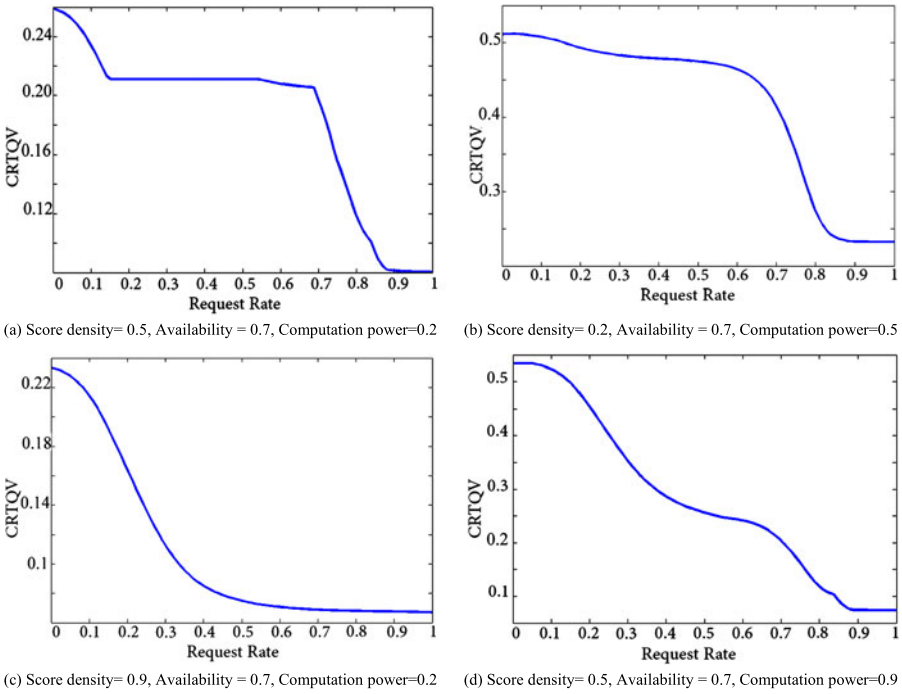
(d) Score density= 0.2 , Availability = 0.7, Request rate=0.9

Fig. 4 Outputs of the *FQVCS* for computation power in different score density, availability and request rate values

ranking [44], and (3) Analytic Hierarchy Process (AHP) [45]. We utilize the MAUT approach which combines the utility function of each parameter while considers the weighting function for each of them. The OCV calculation problem is transformed to a single objective problem by using the MAUT approach. Single objective problems are easy to handle as we have to maximize (or minimize) one specific objective function.

In the following, the parameters used by the CPE algorithm to measure the criticality of the tasks in a Cloud application DAG are mentioned.

- *Communication Cost* ($C(e_{ij})$): The communication cost between two tasks, t_i and t_j, is the amount of data which should be transferred between them. This value can be measured by bytes, kilobytes, etc. As the communication cost between parent and child tasks increases, the tasks become more critical and hence it is desired to schedule these highly dependent tasks on the same cluster or on the clusters with high-speed available communication link.
- *Average Execution Cost of task* t_i *on all available clusters* ($\mathrm{AEC}_{t_i}^{\mathrm{RT_A}}$): This value is based on estimated CPU cycles needed for executing the task t_i and average computation power of resources with type A in all clusters. In most researches estimation for execution cost of a task is calculated as the average of previous run times of the task, but in this paper CPU cycles are used as the unit of measurement. Note that calculating execution cost according to CPU cycles instead of execution time will give us more accuracy to estimate execution time of each task on any



(a) Score density= 0.5, Availability = 0.7, Computation power=0.2

(b) Score density= 0.2, Availability = 0.7, Computation power=0.5

(c) Score density= 0.9, Availability = 0.7, Computation power=0.2

(d) Score density= 0.5, Availability = 0.7, Computation power=0.9

Fig. 5 Outputs of the *FQVCS* for request rate in different score density, availability and computation power values

resources based on resource's computation power. The $\mathrm{AEC}_{t_i}^{\mathrm{RT_A}}$ is calculated by Eq. (14),

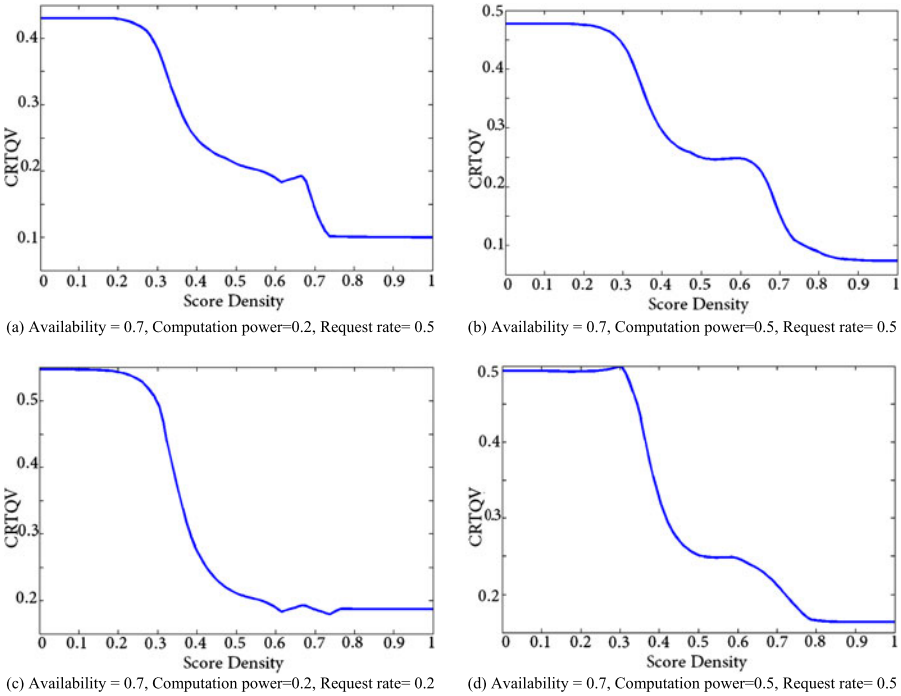
$$\mathrm{AEC}_{t_i}^{\mathrm{RT_A}} = \frac{t_i.\mathrm{Lenght}}{\mathrm{GACP}_{\mathrm{RT_A}}} \tag{14}$$

where $\mathrm{GACP}_{\mathrm{RT_A}}$ is the Grid Average Computation Power for resources of type A, and calculated by Eq. (15),

$$\mathrm{GACP}_{\mathrm{RT_A}} = \frac{1}{k} \sum_{R_j \in \mathrm{Grid}_{\mathrm{RT_A}}} R_j.\mathrm{ComputationPower} \tag{15}$$

where $\mathrm{Grid}_{\mathrm{RT_A}}$ is the set of all resources of type A in the Grid and k is the count of resources in the set $\mathrm{Grid}_{\mathrm{RT_A}}$. The task becomes more critical as its computation cost increases.

- *Count of task successors* ($\mathrm{DAG.Succ}_{t_i}$): A task which has more successors usually depends on more resources for communication, and the count of immediate successors indicates the number of tasks which directly waited for completion of their parent tasks. Therefore, the more successors a task has, the higher importance priority it must have.
- *Time pressure* ($T_{\mathrm{p}}^{t_i}$): Let $T_{\mathrm{p}}^{t_i} \in [0, 1]$ represent task t_i 's time pressure. Let $\mathrm{EST}_{t_i}^{\mathrm{RT_A}}$ be the estimated time at task t_i that can start regardless of the actual resource that will process the task (which will be determined during scheduling). When



(a) Availability = 0.7, Computation power=0.2, Request rate= 0.5

(b) Availability = 0.7, Computation power=0.5, Request rate= 0.5

(c) Availability = 0.7, Computation power=0.2, Request rate= 0.2

(d) Availability = 0.7, Computation power=0.5, Request rate= 0.5

Fig. 6 Outputs of the FQVCS for task score density in different availability, computation power and request rate values

estimated start time $\mathrm{EST}_{t_i}^{\mathrm{RT_A}}$ of task t_i is fast (that is, $T_{\mathrm{p}}^{t_i}$ tends to become one), the Application Management Unit is under more pressure to schedule task t_i. The time pressure is determined by Eq. (16),

$$T_{\mathrm{p}}^{t_i} = 1 - \frac{(\mathrm{EST}_{t_i}^{\mathrm{RT_A}} - \tau)}{\mathrm{Max}(\cup_{t_x \in \mathrm{DAG.Tasks}}(\mathrm{EST}_{t_x}^{\mathrm{RT}_y} - \tau))} \tag{16}$$

where τ is the current time.

As the Grid is a heterogeneous environment and the computation time of tasks varies from resource to resource, it is not possible to compute the exact $\mathrm{EST}_{t_i}^{\mathrm{RT_A}}$. Furthermore, the data transmission time also depends on (1) the bandwidth between selected clusters that can provide the requested resource type, and (2) the amount of data that should be transferred between tasks. Hence, the execution and data transmission times for each unscheduled task should be approximated. If task $\mathrm{t_{entry}}$ is the entry node of the application's DAG, the $\mathrm{EST}_{t_{\mathrm{entry}}}^{\mathrm{RT_A}}$ is computed by Eq. (17):

$$\mathrm{EST}_{t_{\mathrm{entry}}}^{\mathrm{RT_A}} = \tau - \varepsilon \tag{17}$$

where ε is the time which the scheduler needs for schedule task t_i, and τ is the current time.

Table 2 The fuzzy rule base of the FQVCS_FIS

Rule No	TSD	Avl	CP	RR	CRTQV	Rule No	TSD	Avl	CP	RR	CRTQV
1	*R*	*A*	*W*	∼*S*	*B*	17	*R*	*M*	∼*W*	∼*B*	*M*
2	*R*	*M*	∼*W*	*B*	*B*	18	*C*	*A*	∼*W*	*S*	*M*
3	*R*	*M*	*W*	*N*	*B*	19	*R*	*M*	*W*	*B*	*W*
4	*M*	*A*	*S*	*B*	*M*	20	–	*D*	–	–	*W*
5	*M*	*A*	*M*	*N*	*M*	21	*M*	∼*D*	*W*	∼*S*	*W*
6	∼*R*	*A*	*W*	∼*S*	*B*	22	*M*	*M*	∼*W*	*B*	*W*
7	*M*	*M*	∼*W*	∼*B*	*B*	23	*C*	*A*	*M*	*B*	*W*
8	*M*	*M*	*W*	*S*	*B*	24	*C*	*A*	*W*	–	*W*
9	*C*	*A*	*S*	∼*N*	*B*	25	*C*	*M*	*S*	∼*B*	*W*
10	*C*	*A*	*M*	*N*	*B*	26	*C*	*M*	*M*	∼*S*	*W*
11	*C*	*M*	∼*W*	*S*	*B*	27	*C*	*M*	*W*	–	*W*
12	*R*	*A*	*S*	*S*	*E*	28	*C*	*A*	*S*	*B*	*W*
13	*R*	*A*	*S*	∼*S*	*G*	29	*R*	∼*D*	*S*	∼*B*	*G*
14	*R*	*A*	*M*	*S*	*G*	30	*M*	∼*D*	*S*	*S*	*G*
15	*R*	*A*	*M*	∼ *S*	*M*	31	*C*	∼*D*	*M*	*S*	*B*
16	*R*	∼*D*	*W*	*S*	*M*	32	*R*	*M*	∼*W*	*S*	*G*

$$\mathrm{EST}_{t_i}^{\mathrm{RT_A}} = \max_{t_j \in \mathrm{pred}(t_i)} \left\{ \begin{array}{ll} \mathrm{Min}(\mathrm{DL}_{t_j}, \mathrm{EFT}_{t_j}^{\mathrm{RT}_y}) + \frac{c(e_{ij})}{\mathrm{EBW}} & \text{If } t_j \text{ is not scheduled} \\ \mathrm{AFT}_{t_j}^{\mathrm{Cl}_x} + \frac{c(e_{ij})}{\mathrm{EBW}} & \text{otherwise} \end{array} \right\} \tag{18}$$

where EBW is calculated by Eq. (19):

$$\mathrm{EBW} = \frac{1}{n} \sum_{(\mathrm{Cl}_i, \mathrm{Cl}_j) \in \mathrm{CCL} \wedge i \neq j} \left(\mathrm{MBW}^{(\mathrm{Cl}_i, \mathrm{Cl}_j)}\right) - \frac{1}{m} \sum_{k \in \mathrm{IT}} \left(\mathrm{UBW}_k^{(\mathrm{Cl}_i, \mathrm{Cl}_j)}\right)$$
$$\text{where: } \left\{ \begin{array}{l} n = \sum_{(\mathrm{Cl}_i, \mathrm{Cl}_j) \in \mathrm{CCL} \wedge i \neq j} (1) \\ m = \sum_{k \in \mathrm{IT}} (1) \end{array} \right\} \tag{19}$$

where $\mathrm{MBW}^{(\mathrm{Cl}_i, \mathrm{Cl}_j)}$ is the Maximum Band-Width between clusters Cl_i and Cl_j when there is no communication load between Cl_i and Cl_j (i.e., all the bandwidth is unused). The $\mathrm{UBW}_k^{(\mathrm{Cl}_i, \mathrm{Cl}_j)}$ is the Used Band-Width between clusters Cl_i and Cl_j during the time k. The CCL and the IT are candidate cluster list and concerned time slot, respectively.

Let $\mathrm{EFT}_{t_j}^{\mathrm{RT_A}}$ be the estimated finish time at which t_j may finish its computation regardless of the actual resource that will process the task and be calculated by Eq. (20):

$$\mathrm{EFT}_{t_j}^{\mathrm{RT_A}} = \mathrm{EST}_{t_j}^{\mathrm{RT_A}} + \mathrm{AEC}_{t_j}^{\mathrm{RT_A}} \tag{20}$$

After the task is scheduled, the $\mathrm{EST}_{t_j}^{\mathrm{RT_A}}$ and $\mathrm{EFT}_{t_j}^{\mathrm{RT_A}}$ are respectively replaced with actual start time ($\mathrm{AST}_{t_j}^{\mathrm{Cl}_x}$) and actual finish time ($\mathrm{AFT}_{t_j}^{\mathrm{Cl}_x}$).

- *Grid Resource Average Request Rate* ($\overline{\mathrm{GRR}}_{\mathrm{RT_A}}$): As the resources are shared in Grid environment, most of the time the fraction of provided resources to resource consumers is less than one and the Grid resource consumers face with competition in environment. The pressure of competition is varied in different time slots as the resources face different workloads. Assume that task t_i requires resource type A in time slot $[\mathrm{EST}_{t_i}^{\mathrm{RT_A}}, \mathrm{EFT}_{t_i}^{\mathrm{RT_A}}]$. The Grid Resource average Request Rate ($\overline{\mathrm{GRR}}_{\mathrm{RT_A}}$) for requested resource type A is calculated by Eq. (21):

$$\overline{\mathrm{GRR}}_{\mathrm{RT_A}} = \frac{1}{m}\left(\sum_{i=1}^{m} \overline{\mathrm{RR}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}\left(\mathrm{EST}(t_i), \mathrm{EFT}(t_i)\right)\right) \tag{21}$$

where $\overline{\mathrm{RR}}_{\mathrm{RT_A}}^{\mathrm{Cl}_i}(\mathrm{EST}(t_i), \mathrm{EFT}(t_i))$ is calculated as in Eq. (8). The Overall Criticality Value (OCV) of task t_i which belongs to the application with workflow graph indicated by DAG can be calculated by Eq. (22):

$$\begin{aligned}\mathrm{OCV}_{\mathrm{DAG}}^{t_i} = w_1 \times \left(\frac{\sum_{t_k \in \mathrm{DAG.Succ}_{t_i}} (\mathrm{C}(e_{ik})^N + \mathrm{OCV}_{\mathrm{DAG}}^{t_k})}{\mathrm{Max}(2 \times \mathrm{DAG.Succ}_{t_i}.\mathrm{count}, 1)}\right) + w_2 \times T_p^{t_i} \\ + w_3 \times \overset{N}{\mathrm{Succ}}_{t_i} + w_4 \times \mathrm{AEC}_{t_i}^{\overset{N}{\mathrm{RT_A}}} + w_5 \times \overline{\mathrm{GRR}}_{\mathrm{RT_A}}\end{aligned} \tag{22}$$

where the w_i $(i = 1, \ldots, 5)$ is the weight coefficient which determines the impact of the i-th value dimension on the OCV. In Eq. (22) the normalized values of $C(e_{ik})$, Succ_{t_i} and $\mathrm{AEC}_{t_i}^{\mathrm{RT_A}}$ are used (i.e., to be between 0 and 1) which are indicated as $\overset{N}{C}(e_{ik})$, $\overset{N}{\mathrm{Succ}}_{t_i}$ and $\mathrm{AEC}_{t_i}^{\overset{N}{\mathrm{RT_A}}}$, respectively and calculated by Eqs. (23)–(25). The simple formulas used in Eqs. (23)–(25) make the values of $\overset{N}{C}(e_{ik})$, $\overset{N}{\mathrm{Succ}}_{t_i}$ and $\mathrm{AEC}_{t_i}^{\overset{N}{\mathrm{RT_A}}}$ between one and zero, and allow the impact domain of these parameters to be limited by weighting factors w_1, w_3, and w_4 in Eq. (22).

$$\overset{N}{C}(e_{ik}) = \frac{C(e_{ik}) - \mathrm{Min}(\cup_{e_{ij} \in \mathrm{DAG.edges}}(C(e_{ij})))}{\mathrm{Max}(\cup_{e_{ij} \in \mathrm{DAG.edges}}(C(e_{ij}))) - \mathrm{Min}(\cup_{e_{ij} \in \mathrm{DAG.edges}}(C(e_{ij})))} \tag{23}$$

$$\overset{N}{\mathrm{Succ}}_{t_i} = \frac{\mathrm{Succ}_{t_i}.\mathrm{count}}{\mathrm{DAG.tasks.count}} \tag{24}$$

$$\mathrm{AEC}_{t_i}^{\overset{N}{\mathrm{RT_A}}} = \frac{\mathrm{AEC}_{t_i}^{\mathrm{RT_A}} - \mathrm{Min}(\cup_{t_x \in \mathrm{DAG}_{\mathrm{Tasks}}^{\mathrm{RT_A}}}(\mathrm{AEC}_{t_x}^{\mathrm{RT_A}}))}{\mathrm{Max}(\cup_{t_x \in \mathrm{DAG}_{\mathrm{Tasks}}^{\mathrm{RT_A}}}(\mathrm{AEC}_{t_x}^{\mathrm{RT_A}})) - \mathrm{Min}(\cup_{t_x \in \mathrm{DAG}_{\mathrm{Tasks}}^{\mathrm{RT_A}}}(\mathrm{AEC}_{t_x}^{\mathrm{RT_A}}))} \tag{25}$$

The CPE algorithm extracts critical paths according to OCV value of the tasks. In this paper, the critical path is defined as follows:

Definition 1 The critical path of an application DAG is a Depth First Search (DFS) of the DAG from the entry node to exit node with the maximum OCV value of tasks.

The critical path concept is further used by the DAGP algorithm to create sub-workflows as discussed in Sect. 5.2.2. The pseudo-code for the CPE is given in Algorithm 2.

Algorithm 2 CPE function

Type Definition{
TypeDef Task as(int task#,int start_time, int finish_time, int deadline, bool isScheduled,
float OCV, float ComputationWeight
ResourceType RT, float MinAccpetableAvailofResource, Predecessors array of int, Successors array of int);
TypeDef ResourceType as (string ResourceTypeName);
TypeDef Edge as(int task1#, int task2#, float CommunicationWeight);
TypeDef CRTQV_Strcut as(ResourceType ResourceTypes[], float CRTQV);
TypeDef AppDAG as(tasks[0..t] as array of Task, edges as array of Edge);
TypeDef ReservationResult as(int task#, bool Success, int ClusterId, int AFT, int AST);
TypeDef Path as(tasks[0..t] as array of Task, edges as array of Edge, float SumOCV);
TypeDef PairNode as(int task1#, int task2#, int #TargetClusterForFirstNode,
int #TargetClusterForSecondTask, float PartitionRank);
TypeDef Cluster as(int Cluster#, ResourceType AvailableResourceTypes[],
float Average_MIPS_ForEachRT[], CRTQV_Strcut RTQV[]);
TypeDef Schedule as(PN[0..n] as array of PairNode, float ScheduleScore);}

Input:
AppDAG dag // *application DAG*
Output:
boolean (true or false) // *result of advanced reservation for whole application*

boolean **CPE**(AppDAG dag){
1: boolean finished=false;
2: P[] as array of Path;
3: P[] = **GenerateAll_DFS_Paths**(dag);
/∗ The RR variable defined below is a globally shared variable and if in any step one of the partial schedulers MCAR (which are running in parallel) fails to schedule a part of application's DAG, it will change the value of RR variable to false. Therefore if the RR==false then one or scheduler(s) failed. ∗/
4: boolean RR=true;
5: **Thread** Threads_ DAGPCaller[];
6: while(**Count**(dag.tasks.unscheduled==true) > 0){
7: for(int $i = 0$; $i <$ dag.tasks.count; $i++$){
8: if (!dag.tasks [i].isScheduled)
9: $\text{dag.tasks}[i].\text{OCV} = w_1 \times (\frac{\sum_{t_k \in \text{succ}_{t_i}} (C(e_{ik})^N + \text{Score}^{t_k}_{\text{DAG}(j)})}{\max(2 \times \text{Succ}_{t_i}.\text{count}, 1)}) + w_2 \times \overline{\text{Succ}}^N_{t_i} + w_3 \times t_i.\text{Level} + w_4 \times \overline{\text{EC}}^N_{t_i} + w_5 \times \overline{\text{RR}}_{t_i}$
} //Eq. (22)
10: for(int $i = 0$; $i <$ P.count; $i++$){
11: P[i].SumOcv = 0;
12: for(int $j = 0$; $j < P[i]$.count; j++){
13: P[i].SumOcv= P[i].SumOcv +P[i].tasks[j].OCV;} }
14: P[] =**Sort**(P, SumOCV, descending);
15: Path CP=P[0]; // *Selecting the first element of the P[] array. This element has the highest sumOCV value.*
16: int $i = 1$;

Algorithm 2 (*Contnued*)

```
17:                 while((CP.tasks.unscheduled==true) < η){
18:                         CP + = P[i];
19:                         i + +; }
                        // For each DAGP call, a new thread must be created to allow parallel
                           execution of the process
20:                 Threads_ DAGPCaller.Add(new Thread(new ThreadStart(DAGP(CP)));
21:                 int LastAddedThreadIndex= Threads_ DAGPCaller.count - 1;
22:                 Threads_ DAGPCaller[LastAddedThreadIndex].Start();}
23:                 For(int i = 0; i < Threads_ DAGPCaller.count; i + +)
                        { Threads_ DAGPCaller[i].Join(); }
24:     if (!RR){ RollBackUpdates(); // Canceling all successful reservations
                                        for application's DAG
25:                 Return (false); }
26:     Return(true)} //End function
```

When the AMU receives a workflow (DAG), the following steps will be performed by CPE for all unscheduled tasks of the DAG. The CPE first traverses the DAG by utilizing a Depth First Search (DFS) algorithm to find all possible paths of the DAG (line 3). For each unscheduled task the OCV value is calculated according to Eq. (22) (lines 6–9). After calculating the OCV value, the CPE tries to identify the critical path in each step. The critical path is extracted from the summation of tasks' OCV values which belong to a specific path and the CPE selects the path with the highest OCV summation as the critical path (lines 11–17). We consider the parameter η as the threshold of the minimum number of tasks in a critical path. The CPE adds more tasks to the recently extracted critical path, until the count of tasks in it reaches η or all unscheduled tasks of the DAG join the critical path (lines 17–19). Then, the CPE generates a specific thread to call the DAGP algorithm for the recently extracted critical path in parallel with the main thread (lines 20–22). Note that in line 23, the main thread, which runs the CPE algorithm, is joined with all of its child threads (created in line 22), and the main thread will be suspended until all threads in Threads_DAGPCaller array finish. Therefore, lines 24–26 will execute after all DAGPs finish their work (in a hierarchical view the DAGP threads bound to their MCAR schedulers and, therefore, a DAGP thread will end when all of its MCAR scheduler threads end). In this step if RR has the false value, it can be concluded that scheduling of at least one path of the DAG failed, thus the application does not successfully schedule which will require to roll back all effects of scheduling related to application's DAG (as mentioned in line 25).

5.2.2 DAG Partitioning algorithm

The DAGP algorithm dynamically creates partitions according to resource status in each cluster. Creating partitions (sub-workflows) of application DAG usually leads to higher parallelism in both scheduling process and execution of the application. The DAGP tries to fulfill the following needs:

(1) Higher degree of parallelism which leads to better application completion time.
(2) More localized communications between partitions (i.e., decreasing inter-cluster communication).

In most previous works the static DAG partitioning strategy is used, meaning that the whole DAG is partitioned into the sub-workflows before DAG scheduling and the partitions will be unchanged to the end of the scheduling process. As we know, the Grid environment has dynamic nature and in such environment the status of the resources continuously changes through time (i.e., the resource which was the best resource in time t may become a bad resource in time $t + x$ or even it may leave the environment in time $t + x$). Performing the scheduling without noting this dynamism in Grid environment may affect the execution of tasks and may cause the task failure or the need to reschedule due to QoS violation (both task failure and task rescheduling imply extra costs to the system). The actual start time ($\mathrm{AST}_{t_i}^{\mathrm{Cl}_x}$) and actual finish time ($\mathrm{AFT}_{t_i}^{\mathrm{Cl}_x}$) of task t_i is determined after the task is scheduled successfully. As the scheduling proceeds the actual start/finish time determined and new estimations with more accuracy are calculated for remaining unscheduled tasks. It is also worth to note that if the available resources in each cluster are fewer than the number of tasks which are ready to be executed in parallel and the current task requires the same resource in the same cluster, this cluster is eliminated from the candidate cluster list of the current task (while there are available clusters which fulfill this need).

Meeting the predefined deadline and providing some QoS parameters (e.g., reliability, availability, etc.) are the major objectives in this paper. As the parent–child tasks have data dependencies, they usually share a lot of data and therefore it seems that to avoid extra data transfer it is reasonable to put these tasks into the same partitions. But we have to mention that assigning many tasks to the same partition increased some undesirable results (e.g., task failure). Big partition size also means that many tasks should be allocated to the same cluster and therefore the cluster will become more critical. To face this challenge the partition rank parameter, $\mathrm{PartitionRank}_{t_i,t_j}^{\mathrm{Cl}_x,\mathrm{Cl}_y}$, is introduced which indicates the worth of each candidate partition. In each iteration we consider the pair nodes of parent–child tasks in the DAG. Assume that the pair tasks (t_i, t_j) are considered at each step and all possible clusters' combination are studied regarding the requested resource type and clusters' available resource types. If cluster x and cluster y are selected for assigning to them tasks t_i and t_j, respectively, the partition rank value $\mathrm{PartitionRank}_{t_i,t_j}^{\mathrm{Cl}_x,\mathrm{Cl}_y}$ is calculated by Eq. (26).

$$\mathrm{PartitionRank}_{t_i,t_j}^{\mathrm{Cl}_x,\mathrm{Cl}_y} = \mathrm{CRTQV}_{\mathrm{Cl}_x}^{t_i} + \mathrm{CRTQV}_{\mathrm{Cl}_y}^{t_j} + \mathrm{PartitionCriticality}_{i,j}^{x,y} \quad (26)$$

The parameters and equations required for calculating partition rank are presented in Eqs. (27)–(35).

- ParitionCriticality: This value indicates how the selected partitions fit the expectations of the pair tasks. The $\mathrm{PartitionCriticality}_{i,j}^{x,y}$ is calculated by Eq. (27):

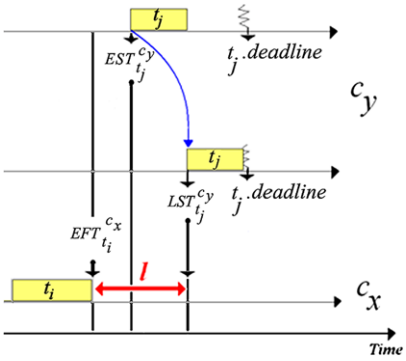


Fig. 7 An example of LST value

$$\text{PartitionCriticality}_{i,j}^{x,y} = \begin{cases} \left\{ \begin{array}{ll} +\beta + \sigma L_{i,j}^{x,y} & \text{IF } (L_{i,j}^{x,y} \geq 0) \\ -\beta + \sigma L_{i,j}^{x,y} & \text{IF } (L_{i,j}^{x,y} < 0) \end{array} \right\} & \text{if } (x == y) \\ \left\{ \begin{array}{ll} +\beta - \sigma \text{CC}_{i,j}^{x,y} + \sigma L_{i,j}^{x,y} & \text{IF } (L_{i,j}^{x,y} \geq 0 \text{ and } L_{i,j}^{x,y} \geq \text{CC}_{i,j}^{x,y}) \\ -\sigma \text{CC}_{i,j}^{x,y} + \sigma L_{i,j}^{x,y} & \text{IF } (L_{i,j}^{x,y} \geq 0 \text{ and } L_{i,j}^{x,y} < \text{CC}_{i,j}^{x,y}) \\ -\beta - \sigma \text{CC}_{i,j}^{x,y} + \sigma L_{i,j}^{x,y} & \text{IF } (L_{i,j}^{x,y} < 0) \end{array} \right\} \\ \text{if} (x <> y) \end{cases} \quad (27)$$

- $\text{CC}_{i,j}^{x,y}$: Communication Cost between clusters x and y for tasks i and j is calculated by Eq. (28).

$$\text{CC}_{i,\text{j}}^{x,y} = \frac{C(e_{ij})}{\text{BW}_{xy}} \quad (28)$$

- $L_{i,j}^{x,y}$: As mentioned in Eq. (29), it is the amount of time between estimated finish time of the task t_j and latest possible start time of task t_i, if the task t_i is scheduled on the cluster x and the task t_j is scheduled on the cluster y. Figure 7 illustrates a sample for the concept of the LST value

$$L_{i,j}^{x,y} = \text{LST}_{t_j}^{\text{Cl}_y} - \text{EFT}_{t_i}^{\text{Cl}_x} \quad (29)$$

- β: This coefficient is used to make the influence of the PartitionCriticality value close to the CRTQV values in the PartitionRank formula (Eq. (26)). The β is the maximum of two CRTQV values of two target partitions in the PartitionRank formula (as mentioned in Eq. (30)), and it helps make the PartitionCriticality value a number in the range of the biggest CRTQV value in Eq. (26).

$$\beta = \text{Max}\left(\text{CRTQV}_{\text{Cl}_x}^{t_i}, \text{CRTQV}_{\text{Cl}_y}^{t_j}\right) \quad (30)$$

- BW: A matrix which represents the maximum bandwidth between clusters in the Grid and is obtained by Eq. (31):

$$\text{BW} = \begin{bmatrix} 0 & \text{bw}_{0,1} & \text{bw}_{0,2} & \text{bw}_{0,m} \\ 0 & 0 & \text{bw}_{1,2} & \cdots \\ 0 & 0 & \cdots & \text{bw}_{m-1,m} \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad (31)$$

- bw: An element of the BW matrix.

$$\mathrm{bw}_{ij} = \begin{cases} 0 & \text{If } i \geq j \\ \text{Maximum bandwidth between } \mathrm{cl}_i \text{ and } \mathrm{cl}_j & \text{If } i < j \\ \text{null} \quad \text{if there is no direct link between } \mathrm{cl}_i \text{ and } \mathrm{cl}_j & \end{cases} \tag{32}$$

- $\mathrm{LST}_{t_j}^{\mathrm{Cl}_y}$: The latest time that task t_i can start its execution if it scheduled on cluster Cl_y and calculated by Eq. (33):

$$\mathrm{LST}_{t_j}^{\mathrm{Cl}_y} = \mathrm{DL}_{t_j} - \mathrm{AEC}_{t_j}^{\mathrm{Cl}_y} \tag{33}$$

- $\sigma L_{i,j}^{x,y}$: The normalized value of the $L_{i,j}^{x,y}$ and it can be calculated by Eq. (34).

$$\sigma L_{i,j}^{x,y} = \frac{L_{i,j}^{x,y} - \mathrm{Min}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(L_{i,j}^{m,n}))}{\mathrm{Max}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(L_{i,j}^{m,n})) - \mathrm{Min}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(L_{i,j}^{m,n}))} \tag{34}$$

- $\sigma \mathrm{CC}_{i,j}^{x,y}$: The normalized value of $\mathrm{CC}_{i,j}^{x,y}$ and can be calculated by Eq. (35).

$$\sigma \mathrm{CC}_{i,j}^{x,y} = \frac{\mathrm{CC}_{i,j}^{x,y} - \mathrm{Min}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(\mathrm{CC}_{i,j}^{m,n}))}{\mathrm{Max}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(\mathrm{CC}_{i,j}^{m,n})) - \mathrm{Min}(\cup_{(\mathrm{cl}_m \wedge c\,\mathrm{l}_n \in \mathrm{clusters})}(\mathrm{CC}_{i,j}^{m,n}))} \tag{35}$$

In the following, some notes are mentioned regarding Eqs. (26)–(35):

- If the candidate clusters for both of the tasks are the same, then the communication cost ($\mathrm{CC}_{i,j}^{x,y}$) is considered equal to zero. In this case, the $\mathrm{L}_{i,j}^{x,y}$ value determines how much critical is the partition.
- If the difference between $\mathrm{LST}_{t_j}^{\mathrm{Cl}_y}$ and $\mathrm{EFT}_{t_i}^{\mathrm{Cl}_x}$ is greater than zero it means that task t_j has sufficient time to start its execution in cluster Cl_y if task t_i is scheduled in cluster Cl_x.
- If the following Boolean expression is true, t_i and t_j can execute in parallel (by neglecting the communication time between t_i and t_j): $\{(\mathrm{EST}_{t_j} \leq \mathrm{EST}_{t_i} \leq \mathrm{EFT}_{t_j}) \vee (\mathrm{EST}_{t_j} \leq \mathrm{EFT}_{t_i} \leq \mathrm{EFT}_{t_j}) \vee (\mathrm{EST}_{t_i} \leq \mathrm{EST}_{t_j} \leq \mathrm{EFT}_{t_i}) \vee (\mathrm{EST}_{t_i} \leq \mathrm{EFT}_{t_j} \leq \mathrm{EFT}_{t_i})\}^{\mathrm{RT_A}}$

The pseudo-code of the DAGP algorithm is mentioned in Algorithm 3. The DAGP starts with receiving a critical path. It generates the pair nodes of the received critical path (line 2). In lines 5–7, for each task in the critical path, the DAGP selects N clusters from the available clusters (which can provide the resource type requested by the task). In lines 11–16, the LST values are calculated regarding the candidate clusters, according to Eq. (33). Lines 17–27 involve calculating the PartitionRank, which requires the calculation of CRTQV and PartitionCriticality (see Eq. (26)). The DAGP requests the CRTQV of each candidate cluster from its Cluster Observation Unit and stores it in the AvailableClusters array (lines 17–24). In lines 24–27 the PartitionCriticality is calculated for each candidate cluster according to Eq. (27).

In lines 28–30, all possible schedules are generated for the pair nodes and the schedule with the highest score is selected as the best schedule. In lines 31–43, a specific execution thread is created for scheduling each task of each pair node in the

Algorithm 3 DAGP function

Input: CP as Path
Output: – (affects globally shared variables) // *result of advanced reservations*
void **DAGP**(Task CP[0..n], Edge E[0..m]){
1: Cluster AvailableClusters[] =**Grid**.**GetAvailableClusters**();
2: PairNode PN[] = **GeneratrPairNodes**(CP);
3: Cluster TargetClusters[0..CP**.**tasks.count-1];
4: For (int $i = 0$; $i <$ CP.count; i++){
5: For (int $j = 0$; $j <$ AvailableClusters.count; j++){
6: If (CP[i].RT **in** AvailableClusters[j].AvailableResourceTypes){
7: TargetClusters[i].Add(AvailableClusters[j]);
8: }
9: }
10: }
11: float LST_Table[0..CP.count][0.. AvailableClusters.count];
12: For (int $i = 0$; $i <$ CP.count; i++){
13: For (int $j = 0$; $j <$ TargetClusters[i].count; j++){
14: LST_Table[i][j]=CP[i].deadline-(CP[i]. ComputationWeight / TargetClusters[j].Average_MIPS_ForEachRT); // *Eq. (33)*
15: }
16: }
17: For(int $j = 0$; $j <$ CP.count; j++){
18: On Each Cluster with index **i** from AvailableClusters do Parallel{
19: For (int $j = 0$; $j <$ AvailableClusters[i].CRTQV.count; j++){
20: AvailableClusters[i].CRTQV[j].CRTQV=, AvailableClusters[i].Cluster_observationUnit.**FQVCS_FIS**(CP[j]); // *defined in Sect. 5.1*
21: }
22: }
23: }
24: For (int $i = 0$; $i <$ PN.count; i++){
25: float PartitionCriticality=**Calculate_PartitionCriticality(PN[i],** PN[i].task1.TargetClusterForFirstNode,PN[i].task2. TargetClusterForSecondNode); //*using Eq. (27)*
26: PN[i].PartitionRank=PN[i].task1.TargetClusterForFirstNode.CRTQV + PN[i].task2.TargetClusterForSecondNode+ PartitionCriticality; //*using Eq. (26)*
27: }
28: Schedule S[] =**Generate_Schedules(PN)**; //*described in Sect. 5.2.2*
29: int i = getmax(S[],ScheduleScore);
30: Schedule BestSchedule=S[i];
31: ReservationResult ResRes[S.count∗2];
32: int ri = −1;
33: **Thread** Threads_ MCARCaller[];
34: For (int $i = 0$; $i <$ S.PN.count; i++){
35: ri++;
36: Threads_MCARCaller.Add(new **Thread**(new **ThreadStart**(TargetClusterForFirstNode.Cluster_cordinationUnit.**MCAR**(S.PN[i]. task1, S.PN[i].TargetClusterForFirstNode, ResRes[ri]);

Algorithm 3 (*Contnued*)

```
37:     Int LastAddedThreadIndex= Threads_ MCARCaller.count − 1;
38:     Threads_ MCARCaller[LastAddedThreadIndex].Start();
39:     ri++;
40:     Threads_ MCARCaller.Add(new Thread(new ThreadStart(
                    TargetClusterForSecondNode.Cluster_cordinationUnit.MCAR(
                    S.PN[i].task2, S.PN[i].TargetClusterForSecondNode,ResRes[ri]);
41:     int LastAddedThreadIndex= Threads_ MCARCaller.count − 1;
42:     Threads_ MCARCaller[LastAddedThreadIndex].Start();
43:     }
44:     For(int i = 0; i < Threads_MCARCaller.count; i ++){
45:         Threads_ MCARCaller [i].Join();
46:         }
47: For(int i = 0; i < ResRev.count; i++){
48:       If (ResRev[i].Success==false) {
49:         RR= false;
50:         Return;
51:       }
52:       Update( dag, ResRev[i].task#, ResRev[i].ClusterId, ResRev[i].AFT, ResRev[i].
                  AST_ );
53:       }
}         //End Function
```

selected schedule. Each of these partial scheduler threads calls the MCAR function, which is "responsible for scheduling a certain task in a certain Grid cluster." The main thread of the algorithm joins with all newly generated worker threads (lines 44–46) which means the main thread will wait until all these threads finish their works and successfully end. Thus, lines 47–53 will not execute until all the MCAR functions, which are called by main thread, will finish successfully. In lines 47–53 the results of the calling of the MCAR function for each task are examined. If there is a task which failed to be scheduled, the function sets the reservation result of the whole schedule to false and exits from the DAGP function. Otherwise, the DAG table is updated with Actual Start Time (AST_) and Actual Finish Time (AFT).

5.3 Cluster Coordination Unit (CCU)

Each cluster has a Cluster Coordination Unit (CCU) which is responsible for scheduling tasks on cluster's resources. The CCU receives a task and uses the proposed Multi-Criteria Advanced Reservation algorithm (MCAR) which schedules a given task on the resources of a Grid cluster while meets the reliability and the QoS expectations simultaneously.

Most of the previous workflow management systems are using scheduling methods with focus on the minimizing the execution time of the workflows [6, 7, 46]. Although these approaches seem promising, they suffer from common issues. First, some approaches [7, 46] may fall into local minimum as they use the execution time parameter in a greedy fashion. Second, many of these approaches try to schedule the tasks on the fastest resources, and it is not wise to schedule a group of related

(i.e., parent–child) tasks to the same resource. If the resource fails by any reason, we have to reschedule all tasks which are assigned to it and this is too costly. Also, as the resource has the highest speed among the others, it may be frequently selected to execute the important tasks (tasks with higher OCV value). Although selecting the fastest resource reduces the execution time, it significantly increases the cost of failure.

In addition to the execution time, there are other potential QoS attributes like reliability, availability and failure tolerance which may be desired by applications' owners in a Cloud environment. Therefore, instead of using the earliest execution time strategy, the MCAR algorithm is introduced as a new QoS-based heuristic to avoid biased schedules.

The MCAR algorithm uses a new parameter called Quality of Resource (QoR) which determines the appropriateness of each resource. We reapply the Multiple Attribute Utility Theory (MAUT) for finding the QoR. In this approach, for a task $t_i \in V$, a QoR value should be calculated for each resource r in the set of resources of the cluster Cl_x.resources, assuming the t_i is scheduled on resource r. The QoR value of resource r is defined based on following parameters:

- $\mathrm{AVL}_r^{\mathrm{Cl}_i}$ (Resource's Availability): The probability that the resource r can be contacted to be consumed at a given time.
- $\mathrm{Util}_r^{\mathrm{Cl}_i}$ (Resource's Utilization): This parameter is the percentage that resource is used. Let L denote the amount of time that the scheduler can look "into the future." On the other word, a task may requested to schedule at most L units of time in the future. Utilization is calculated by Eq. (36).

$$\mathrm{Util}_r^{\mathrm{Cl}_i} = \sum_{t_j \in k} \frac{\min(\mathrm{EFT}_{t_j}^r, C + L) - \max(\mathrm{EST}_{t_j}^r, C)}{L} \tag{36}$$

  where C is the current time.
- $\mathrm{TSD}_r^{\mathrm{Cl}_i}$ (Resource's Task Score Density): This parameter is described in Eq. (4) (in Sect. 5.1.2).
- $\mathrm{STG}_r^{\mathrm{Cl}_i}$ (Resource's Sum of Task Gaps): The MCAR algorithm computes resource's sum of task's gaps ($\mathrm{STG}_r^{\mathrm{Cl}_i}$) value for each resource as a sum of imposed left-gap and right-gap if the task scheduled on that resource. In most of the task scheduling algorithms, the earliest available time of a processor P for a task execution is the time when P completes the execution of its last assigned task [3]. However, the MCAR's advanced reservation phase uses the insertion-based policy [28] which considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a resource. The insertion-based policy is discussed as follows:
- *Insertion-based policy*: Let $[\mathrm{Id}_i, \mathrm{Id}_j], \mathrm{Id}_i, \mathrm{Id}_j \in [0, \infty]$, be an idle time interval (i.e., an interval in which no task is executed) on resource R. A task t can be scheduled on resource R within timeslot $[\mathrm{Id}_i, \mathrm{Id}_j]$ only if R.ResourceType = t.ResourceType with type A and the condition mentioned in Eq. (37) is true.

$$\mathrm{Max}(\mathrm{Id}_i, \mathrm{t.StartTime}) + \mathrm{t.Weight} \leq \mathrm{Id}_j \tag{37}$$

This condition allows task t to be scheduled between already scheduled tasks (insertion–based policy) [38]. Searching for the suitable idle time slot continues until finding the idle time slots that are capable of accomplishing the task t.

- MCCP(t_i): Maximum Communication Cost with Predecessors for task t_i can be defined by Eq. (38):

$$\text{MCCP}(t_i) = \text{Max}_{t_j \in t_i.\text{predecessors}(c_{ij})} \tag{38}$$

Note that if a task and all of its predecessors are assigned to the same resource, then the communication cost between the task and its predecessors will be eliminated. It is preferred allocating related tasks to the resources with the faster interlinks (which decreases the general finish time).

- $\text{Fail}_r^{\text{Cl}_i}$: This parameter indicates the ratio of failed tasks to all assigned tasks on resource r and it is calculated by Eq. (39):

$$\text{Fail}_r^{\text{Cl}_i} = \frac{U^r}{n} \tag{39}$$

where n is the number of all assigned tasks on resource r, and U^r is the unit penalty function which calculated by Eq. (40).

$$U^r = \sum_{j=1}^{n} U_j \tag{40}$$

where $U_j = 1$ if the task t_j has failed and otherwise $U_j = 0$. The lower this ratio is, the better QoR the resource will have.

In order to evaluate the QoR of each resource, the linear approach is given by Eq. (41).

$$\begin{aligned}\text{QoR}_{t_i}^r = {} & a \times \text{AVL}_r^{\text{Cl}_i} + b \times \text{Util}_r^{\text{Cl}_i} - c \times \text{TSD}_r^{\text{Cl}_i} - d \times \text{STG}_r^{\text{Cl}_i} - e \times \text{MCCP}(t_i) \\ & - f \times \text{Fail}_r^{\text{Cl}_i}\end{aligned} \tag{41}$$

where a, b, c, d, e and f are weighing factors which indicate the impact of each parameter in Eq. (41).

Algorithms 4, 5, 6 illustrate the main functions of the MCAR method. The description of each algorithm is mentioned in the following.

The Main Routine function (Algorithm 4) is the main function which receives an application's DAG as an array of tasks and controls the reservation. In line 6 the MCAR_Preprocess function is called to set up W[][] and ST[][] arrays. In line 7 a sorted version of Task[] array stores in STL[] array. Lines 8–17 form a loop in which, by using MCAR_DoReservation function, all tasks will be scheduled. In each iteration of the loop an unscheduled task which has the highest OCV is selected. If DoReservation, returns a true result (which means that the current task is successfully scheduled on a resource), the function proceeds with next task in STL[] array. If DoReservation fails to schedule a given task in each step, the whole process fails and the function returns a Boolean false result (line 11), and if all tasks in the Tasks array scheduled successfully, the function returns a Boolean true result (line 18) indicating that the whole sub-workflow is successfully scheduled. In line 13 and after each successful reservation the deviation of request rate is compared with predefined

Algorithm 4 The MCAR algorithm—*MainRoutine* function

MCAR_MainRoutine function
Type definition:
1: TypeDef Task as (int task#,int start_time, int finish_time, int deadline, bool isScheduled, int score);
2: TypeDef AdvResScore as (int resource#, float QoR)
3: TypeDef Resourceinfo as (int resource#, float availability, float utilization, int Hss, Array of Task assignedtasks);
Input:
1. CW[0..t][0..t] array of float // *communication weight between tasks*
2. Tasks[0..t] array of Task // *application tasks*
3. Resource[0..n] array of Resourceinfo // *Cluster Resources*
4. B[0..n][0..n] array of float // *bandwidth between all resources*
note: B[i][i] $= \infty$
Output:
bool (true or false) // *result of advanced reservation for whole application* (*bag of tasks*)
1: **PUBLIC DECLARATIONS**{
2: var W as array[0..t][0..n] of float; //*computation weights of tasks on resources*
3: var ST as array[0..t][0..n] of float; //*start times of tasks on resources*
}
4: **bool MCAR_MainRoutine** (CW[0..t][0..t] array of float, tasks[0..t] array of Task, Resource[0..n] array of Resource, B[0..n][0..n] array of float)**{**
5: var STL[0..t] array of Task;
6: MCAR_preprocess();
7: STL[] =**Sort**(Tasks[], Score, descending); // *Sorting the Task list array based on Score field in descending order*
8: for(int $i = 0$; $i < t$; i++){
9: if (STL[i].isSheduled==false){
10: bool result= **DoReservation**(W[i][∗], ST[i][∗], STL[i]);
11: if (result ==false) return false; // *if advanced reservation of a single task fails, the scheduling of the whole application fails*
12: STL[i].isScheduled=true;
13: If (Δ RequestRate >= δ){
14: for(int $i = 0$; $i <$Tasks.count; i++){
15: if (!dag.tasks [i].isScheduled)

$$\mathrm{STL}[i].\mathrm{OCV} = w_1 \times \left(\frac{\sum_{t_k \in \mathrm{DAG.Succ}_{t_i}} (C(\mathrm{e}_{ik})^N + \mathrm{OCV}_{\mathrm{DAG}}^{t_k})}{\mathrm{Max}(2 \times \mathrm{DAG.Succ}_{t_i}.\mathrm{count}, 1)} \right) + w_2 \times T_{\mathrm{p}}^{t_i} + w_3 \times \mathrm{Succ}_{t_i}^{N} + w_4 \times \mathrm{AEC}_{t_i}^{\mathrm{RT_A}} + w_5 \times \overline{\mathrm{GRR}}_{\mathrm{RT_A}}$$

}
16: STL[] =**Sort**(Tasks[], Score,descending); // *Sorting the Task list array based on Score field in descending order*
17: $i = 0$; // *restarting loop, because now the tasks' scores are changed and so the STL array.*
// *Note that the scheduled nodes will not be processed again* (*the if-condition in line 10 prevents this.*)
}
}
}
18: return (true);
}

Algorithm 5 The MCAR algorithm—the *MCAR_Preprocess* function

MCAR_Preprocess() function
Input:
The following arrays will be accessed in this function**: CW[][]**, **Task[]**, **Resource[]** and **B[][]**
Note that these arrays are input parameters of MCAR_MainRoutine function
Affected Variables:
1. W[0..t][0..t] array of float *//computation weight of tasks on resources*
2. ST[0..t][0..t] array of float *//start times of tasks on resources*
Output: No regular outputs, all changes reflected on following public arrays: W[][] and ST[][]

1: **MCAR_Preprocess()**{
2: for (int $i = 0$; $i < t$; i++){
i. for ($j = 0$; $j < n$; j++){

$$w[i][j] = \frac{T[i].\text{Lenght}}{H[j].\text{cyclepersecond}}; \} \}$$

3: for (int $j = 0$; $j < n$; j++) {ST[0][j]=CurrentTime; }
4: for (int $i = 0$; $i < t$; i++){
i. var CC as array[0..Task[i].Predecessors.count][0..n] of float;
ii. for (int p = 0; p < tasks[i].Predecessors.count;p++) {
a. for(int $j = 0$; $j < n$; j++){
i. CC[p][j]= CW[tasks[p]][tasks[i]] /
B[tasks[p].Reservedresource][j]; }}
iii. for(int $j = 0$; $j < n$; j++){
iv. $ST[i][j] = \text{Max}_{x \in task[i].predecessors}((\text{Task}[x].FinishTime) + \text{CC}[\text{Task}[x]][j])$;
}
5: return (); *// Noting that W[][] and ST[][] are declared publicly in main routine, thus this function does not need to pass these arrays for further uses.*
}

threshold, δ, and if it exceeds the δ then the scores of unscheduled tasks are recalculated.

The MCAR_Preprocess function (Algorithm 5) prepares the values of two arrays, W[][] and ST[][], which store computational weights and start times, respectively. The W[][] and ST[][] arrays are declared as public and general arrays which make them accessible in all functions. In line 2.i, the W [][] array is filled with computation cost (run-time duration) of each task on each resource by using nested loops. The computation cost is estimated according to resource's processing power (cycle per second) and estimated task length (CPU cycles). In line 3 the start time of first task is set to current time for all resources. The communication cost between two tasks can be calculated according to communication weight (the amount of data that should be transferred) between them and the available bandwidth among the resources in which the tasks are located (e.g., CC = CW/B). According to this fact in line 4.ii.a.i and in a nested loop the communication cost between each task with its related neighbors is calculated.

In line 4.iv the communication costs (CC) are used to estimate the Start Time (ST) of the task on each resource. As can be seen in line 4.iv, the start time of each task can be calculated regarding the finish times and communication costs of its predecessor tasks. It can be said that the start time of a task is the time when all of its predecessors finished and completed their communication with the task. In other words, the task

Algorithm 6 The MCAR algorithm—*MCAR_DoReservation* function

MCAR_DoReservation function

Input:

1. $T_x W$ array of float *//running weight of Task$_x$ on each cluster resource*
2. T_xST array of float *//start time of Task$_x$ on each cluster resource*
3. T_x Task *// the task for which the algorithm must schedule*

Output: true or false *//according to the result of advanced reservation for the given task* (T_x)

1: bool **MCAR_DoReservation**($T_x W$ array of float, T_xST array of float, T_x Task)**{**

2: var **hil** as array of Resourceinfo

3: var Reservation_QoR_List as array of float;

4: Fill((**hil**, "select ∗ from ResourceinfoRepository");

5: for each Resourceinfo r in **hil**

{

i. var does_request_fit as Boolean= True;

ii. $\mathrm{EST}^r_{t_x} = \mathrm{T}_x\mathrm{ST(r)}$;

iii. $\mathrm{EFT}^r_{t_x} = \mathrm{EST}^r_{t_x} + T_x W(r)$;

iv. for each Task t_i In this_resource.assignedtasks

{ if!(($\mathrm{EST}^r_{t_x} > \mathrm{EFT}^r_{t_i}$)||($\mathrm{EFT}^r_{t_x} < \mathrm{EST}^r_{t_i}$)){

if ((($\mathrm{EST}^r_{t_x} \geq \mathrm{EST}^r_{t_i}$)&&($\mathrm{EFT}^r_{t_x} \leq \mathrm{EFT}^r_{t_i}$))||(($\mathrm{EST}^r_{t_x} < \mathrm{EFT}^r_{t_i}$)&&($\mathrm{EFT}^r_{t_x} \geq \mathrm{EFT}^r_{t_i}$))||(($\mathrm{EST}^r_{t_x} \leq \mathrm{EST}^r_{t_i}$)&&($\mathrm{EFT}^r_{t_x} > \mathrm{EST}^r_{t_i}$))) {does_request_fit = false; Exit Loop; } }

}

v. if (does_request_fit) {

a) var STG = [($\mathrm{EST}^r_{t_x} - \mathrm{EFT}^r_{t_j}$) + ($\mathrm{EST}^r_{t_j} - \mathrm{EFT}^r_{t_x}$)];

b) STG= STG MOD Job_Mean_size;

c) var CC as array of float;

d) for each **Task** pt In T_x.Predecessors **{** CC[pt] = CW[pt][T_x]/ B[pt.resource][r]; **}**

e) var A = Min($\mathrm{TSD}_r^{\mathrm{Cl}_i}$); var B = Max($\mathrm{TSD}_r^{\mathrm{Cl}_i}$);

A. For each Resourceinfo h in hil{ $\mathrm{TSD}_h^{\mathrm{Cl}_i}$_Norm = $1 + (\mathrm{TSD}_h^{\mathrm{Cl}_i} - A) \times (10-1)(B-A)$; }

f) var $\mathrm{QoR}^r_{t_i} = a \times \mathrm{AVL}_r^{\mathrm{Cl}_i} + b \times \mathrm{Util}_r^{\mathrm{Cl}_i} - c \times \mathrm{TSD}_r^{\mathrm{Cl}_i} - d \times \mathrm{STG}_r^{\mathrm{Cl}_i} - e \times \mathrm{MCCP}(t_i) - f \times \mathrm{Fail}_r^{\mathrm{Cl}_i}$;

g) AddItem(Reservation_QoR_List, c_resource.resource#, $\mathrm{QoR}^r_{t_i}$); **} }**

6: var best_target_resource_id as int;

7: var advanced_reservation_result as Boolean=false;

8: while (advanced_reservation_result ==false)

{

i. target_resource_id = FindMax(Reservation_QoR_List.QoR);

ii. advanced_reservation_result = PerfromAdvancedReservation(target_resource_id, t_x);

iii. if (advanced_reservation_result==true) exit loop;

else RemoveItem(Reservation_QoR_List, target_resource_id);

}

9: return (advanced_reservation_result); **}**

cannot start until all of its predecessors finished and sent required data to it; thus, the MAX operator is used in line 4.iv.

The MCAR_DoReservation function (Algorithm 6) receives a task T_x and calculates QoR of each resource for T_x. As soon as the MCAR_DoReservation function calculates the QoR, it tries to perform the advanced reservation on the resource with the highest QoR value. If the best resource could not accept the requested reservation, then the next best resource will be chosen and the MCAR_DoReservation function will try to perform the advanced reservation on it. This process will continue until the task T_x is successfully scheduled on a resource (the function returns a true value) or all candidate resources refuse the reservation request (the function returns a false value). In line 4 of MCAR_DoReservation function, most current information about resources is retrieved from Resource_info_Repository database and stored in a temporary variable named hil. According to insertion based policy the DoReservation finds candidate resources in which the task can be scheduled on them (lines 5.i–5.iv). In lines 5.v.a–5.v–f the DoReservation calculates each resource's QoR according to Eq. (41). After all candidate resources are found and their QoR for task t_x are calculated, DoReservation chooses the resource with the highest QoR (line 8.i) and tries to schedule the task on the target resource (line 8.ii); if the reservation is performed successfully the result will be passed to MCAR_MainRoutine (the caller algorithm), otherwise the next best candidate resource (the next resource with maximum QoR) will be selected and line 8.ii is repeated for it. If all candidate resources tried and none of them allows the reservation, DoReservation informs MCAR_MainRoutine passing a Boolean False to it.

6 Performance evaluation

Software simulation is used extensively for modeling and evaluation of real world systems. Consequently, modeling-and-simulation has emerged as an important discipline around which many standard and application-specific tools and technologies have been built. To evaluate the performance of the proposed system we used software simulation and compared it with the DLS [5], the HEFT [7], the QRS [42] and the AWS [16] which are well-known methods for DAG scheduling (these algorithms are described in Sect. 2).

We assume that requests arrive as a Poisson process with rate λ. The tasks are generated with random size by a Pareto distribution. The minimum and maximum task sizes are set to 1 and 50 unit(s) of time, respectively. Let L denote the amount of time that a task may request to be scheduled. Deadline is the latest time by which the task processing should be completed. Tasks have the hard deadline by which they should be processed or they fail. If D_j is the deadline of task t_j then it is uniformly distributed as is given in Eq. (42):

$$t_j.\text{StartTime} + t_j.\text{ExecutionTime} \leq t_j.\text{Deadline} \leq t_j.\text{StartTime} + t_j.\text{ExecutionTime} + q \times (t_j.\text{ExecutionTime}) \quad (42)$$

where q is a parameter to control the tightness of task deadline and it is between 0 and 1. In our simulation, q and L are set to 0.1 and 200 seconds, respectively. The

system workload [39] ρ is calculated using the expression from queuing theory as in Eq. (43):

$$\rho = \frac{\lambda . x}{n} \tag{43}$$

where ρ is the arrival rate of the requests, x is mean task size and n is the number of count resources which are available to respond to the requests.

6.1 Random graph generator

We used random graph generation with the count of nodes in each workflow graph set between 25 and 100 nodes. Each graph has a single entry and a single exit node; all other nodes are divided into levels. Each level is created progressively and has a random number of nodes, which varies from two to half the number of the remaining (to be generated) nodes. We used the random graph generator discussed in [7]. This random graph generator requires following input parameters:

- v: The number of nodes in the DAG
- Out degree: The ratio of maximum out edges of a node to total nodes of the DAG
- Communication to Computation Ratio (CCR). Note that the higher *CCR* value indicates the more computation-intensive applications [16]
- β: The computational heterogeneity factor of resources. Assume that $\overline{\text{CompCost}}$ is the average computation cost of all tasks in a DAG and $\overline{t_i.\text{CompCost}}$ is the average computation cost of task t_i and randomly selected by the uniform distribution within the range $[0.2 \times \overline{\text{CompCost}}]$. The computation cost of task t_i on resource r, $\text{CompCost}(t_i, r)$, is a random number from the range: $\overline{t_i.\text{CompCost}} \times (1 - \frac{\beta}{2}) \leq \text{CompCost}(t_i, r) \leq \overline{t_i.\text{CompCost}} \times (1 + \frac{\beta}{2})$
- α: The depth parameter of the DAG. This parameter indicates the depth of a DAG by using the uniform distribution with the mean value equal to $\frac{\sqrt{v}}{\alpha}$. If $\alpha \gg 1.0$ then short graphs with high parallelism are generated, and if $\alpha \ll 1.0$ then long graphs with low parallelism degree are generated. The values for the input parameters are shown in Table 3.

6.2 Comparison metrics

The following four metrics are used to evaluate the performance of the proposed system:

Table 3 Parameter values of random generated DAGs [3]

Parameter	Value
v	20, 40, 60, 80, 100
out degree	0.1, 0.2, 0.3, 0.4, 1.0
CCR	0.1, 0.5, 1.0, 5.0, 10.0
β	0.1, 0.25, 0.5, 0.75, 1.0
α	0.5, 1.0, 2.0

- *Request Rejection Ratio (RRR)*: This metric is the fraction of requests that are rejected due to inability to provide their required resources, and it is calculated thus:

$$\text{RRR} = \frac{\text{Rejected Request Count}}{\text{Total Request Count}} \times 100 \tag{44}$$

- *Application Failure Ratio (AFR)*: This metric is the percentage of applications which failed to complete due to failure of one or more of its jobs. An application consists of a number of jobs and therefore a job failure may cause an application failure. The AFR value is calculated thus:

$$\text{AFR} = \frac{\text{Count of Failed Apps}}{\text{Count of Apps}} \times 100 \tag{45}$$

- *System Utilization (SU)*: This metric is the fraction of time the resources are busy serving the requests and it is calculated thus:

$$\text{SU} = \frac{\text{Busy Time}}{\text{Total UpTime}} \times 100 \tag{46}$$

- *The running time of the algorithms*: This metric is the execution time of an algorithm for obtaining the output schedule of a given task graph. This value gives the average cost of each algorithm. The simulation configuration is defined with following characteristics:
  - Simulation period: 24 hours (86,400 seconds)
  - Count of Grid hosts: 50
  - Total count of Cloud application owner users: 110
  - Total count of submitted applications: 180
  - Total count of submitted tasks: 15,000

Figures 8(a)–(c) plots the request rejection rate for the four scheduling strategies against the average job size in workloads $\rho = 0.2$, $\rho = 1.2$, $\rho = 2$ and $\rho = 4$. The request rejection rate increases as requests' sizes increase for all four scheduling algorithms in different system workloads. The DLS does not consider the resource status and idle periods; thus, it gives worse results. The HEFT algorithm gives lower request rejection rate compared with the DLS, but performs slightly worse than the QRS and the AWS. The main cause which leads to this result is that in the HEFT algorithm tasks are scheduled on the resource which provides the minimum finish time which will create small idle (and usually unusable) gaps. These small gaps are usually smaller to be used for incoming tasks; hence, they increase the request rejection rate in the system. The AWS performs better than the HEFT and the QRS as it utilizes look-forward technique. The proposed system considers the CRTQV factor in partitioning phase, and it significantly decreases the request rejection rate as the CRTQV is the combination of multiple factors including computation power (the cluster with more powerful resources is generally capable of responding to more requests compared with a similar cluster with lower computation power) and request rate of resources (the cluster with higher request rate is busier, and selecting it as candidate cluster for scheduling a task increases the probability of request rejection). The proposed system also considers the QoR parameter in local-scheduling which includes two major factors: R^h_{TSD} which determines the density of assigned tasks in

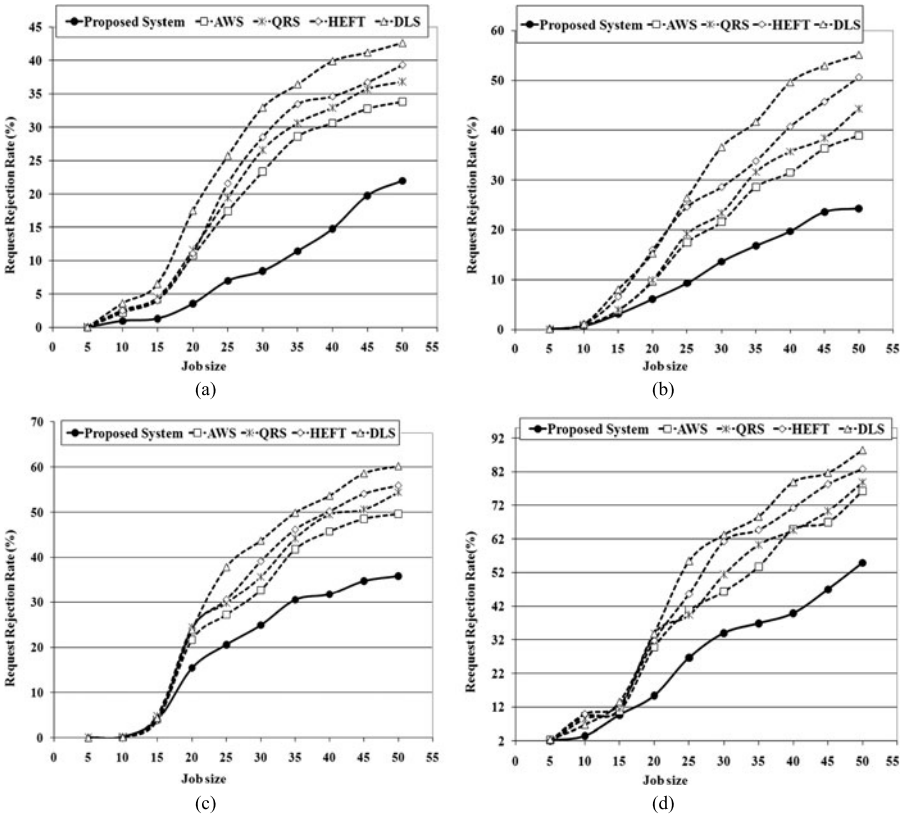


Fig. 8 Request rejection rate against average job size in: (**a**) $\rho = 0.2$, (**b**) $\rho = 1.2$, (**c**) $\rho = 2$, (**d**) $\rho = 4$

each resource and R^h_{STG} which determines the summation of idle gaps (which are generated by each task). The proposed system considers R^h_{TSD} and R^h_{STG} which causes assigning higher priorities to the resources with lower task density and lower imposed trailing gaps.

In Figs. 9(a)–(d) comparison results of the resource utilization versus the average job size are plotted in constant workloads $\rho = 0.2$, $\rho = 1.2$, $\rho = 2$ and $\rho = 4$. Initially, utilization increases with the system load. The DLS shows the lowest utilization as a result of the high request rejection rates (as can be seen in Figs. 8(a)–(d)). The proposed system surpasses the others followed by the AWS, the QRS and the HEFT. This result confirms that the consideration of the request rate factor by the proposed system decreases the probability of dropping high priority jobs. Moreover, the consideration of the request rate factor significantly decreases the probability of selecting resources with higher request rate which will result in more balanced distribution of tasks on the resources. The MCAR also considers the idle gaps imposed by assigning a task to a resource, and hence, the creation of trailing idle gaps is one of the most important criteria used to evaluate the "goodness" of an assignment (i.e., a schedule). Using the MCAR algorithm leads to more utilized resources and, as a consequence, fewer request rejections (which happens due to lack of sufficient free resources). This

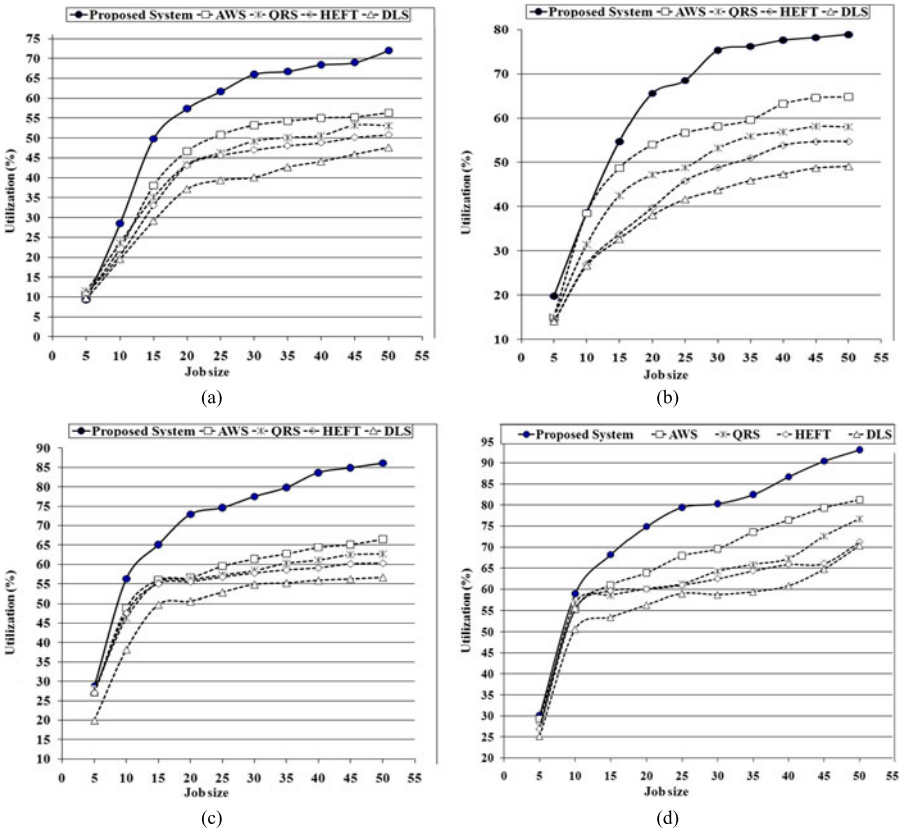


Fig. 9 Resource utilization against average job size in: (**a**) $\rho = 0.2$, (**b**) $\rho = 1.2$, (**c**) $\rho = 2$, (**d**) $\rho = 4$

result proves affirmatively the effect of considering system fragmentation on improving utilization. Note that the difference in utilization between the HEFT and the other algorithms (the LSWF, the AWS and the QRS) can be explained by the fact that the HEFT tends to drop larger jobs as it is more difficult to find feasible idle time slots.

Figure 10 plots the algorithm run time against task count, and the run time of the algorithms increases with the tasks' count for all five scheduling algorithms. In Fig. 10, as the count of tasks increases, the proposed system outperforms the other algorithms as its hierarchical distributed scheme benefits from the parallel execution, thus the algorithm run time decreases.

In Figs. 11(a)–(d), test results for application failure rate versus mean time between failures are illustrated and the proposed system surpasses the other four algorithms. The consideration of the resource idle time slots, resource utilization, resource's density, and resource availability in resource selection decreases the probability of job failures in the proposed system. Distributing the workloads according to the priority of the tasks and density of tasks on the resources also significantly decreases the probability of job failures by the proposed system. Considering the LST value by the proposed system in partitioning phase offers more time space for

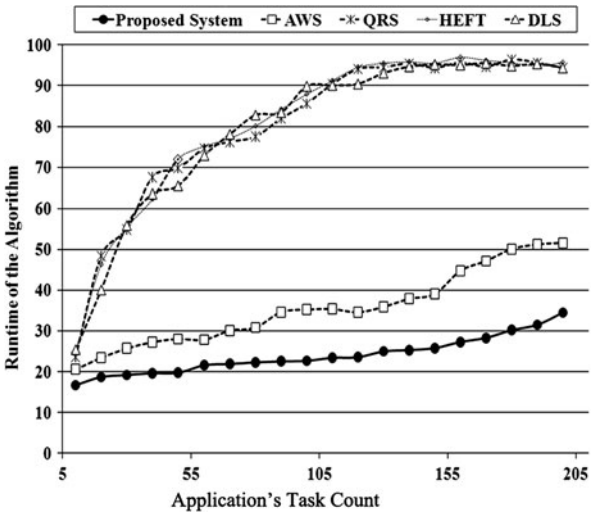


Fig. 10 Run time of the algorithms against application's task count

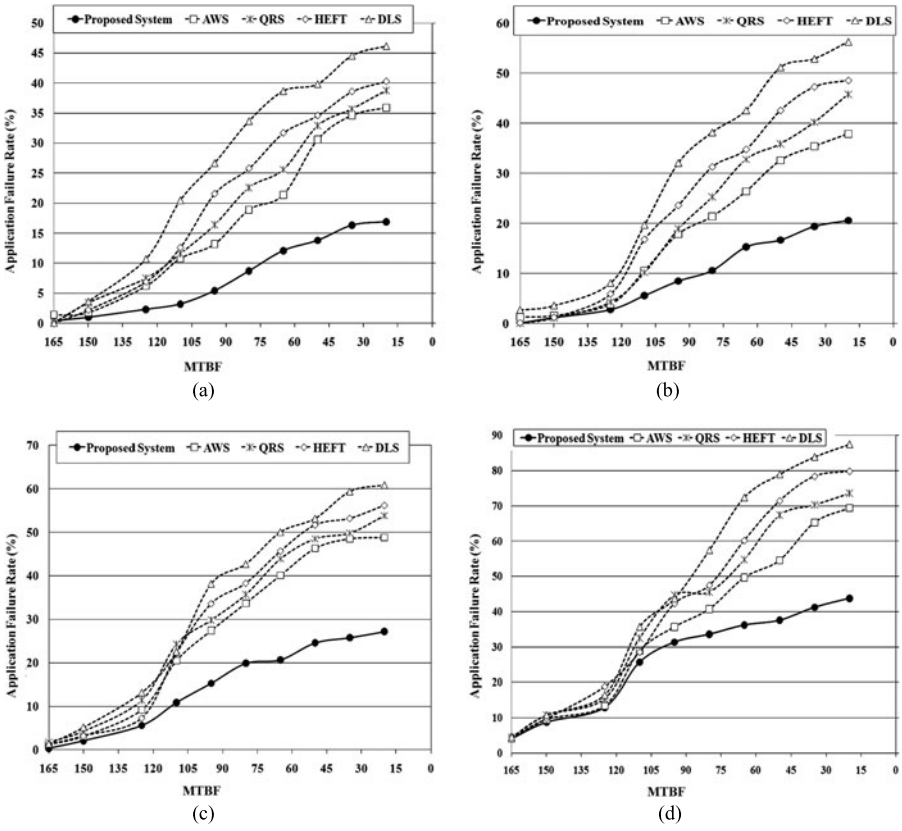


Fig. 11 Application failure rate vs. resources MTBF in (**a**) $\rho = 0.2$, (**b**) $\rho = 1.2$, (**c**) $\rho = 2$, (**d**) $\rho = 4$

the scheduler to reschedule or for the system administrator to recover the failed resource, thus increase the chance of completing the failed task before a failure causes cascading task failure in the successor tasks and eventually the failure of the whole application. The more time space available for rescheduling gives the higher probability of successful reschedule of failed task before its deadline and consequently increases the probability of avoiding application failure.

7 Conclusion and future work

The main contribution of this paper is addressing the problem of scheduling workflow Cloud applications on multi-cluster Grid environments regarding the QoS constraints declared by application's owner. To obtain this objective, we introduced a bi-level scheduling strategy that consists of Global-level scheduling and Local-level scheduling. The heterogeneity and a variety of the available resource types (service types) in each Grid cluster are an issue which dramatically affects the complexity of scheduling workflows. On the other hand, a Cloud application workflow consists of different tasks with the need for different resource types (service types) to complete, which we called heterogeneity in workflow. The main idea of all proposed algorithms and techniques introduced in this paper is to make a good match between the heterogeneity in Cloud application's workflow and the heterogeneity in Grid clusters regarding the requested QoS expectation. The proposed algorithms mainly focused on the using of the idea of matching heterogeneity between clusters and dynamically extracted sub-workflows to maximize the possibility of local scheduling of sub-workflows (i.e., increasing the chance of scheduling each sub-workflow in a specific Grid cluster with aim to minimize inter-cluster communications while following the disciplines of workload distribution). A notable fuzzy-based assessment method is also proposed to evaluate the appropriateness of a Grid cluster for a computational job.

The proposed system consists of three units: (1) the Cluster Observation Unit (COU) which uses a novel Fuzzy FQVCS for evaluating the Grid environment by taking into account some important QoS measures (e.g. availability, workload and so forth); (2) the Application Management Unit (AMU) which is responsible for performing Global-level scheduling by partitioning the large-scale Cloud workflows into sub-workflows; and (3) the Cluster Coordination Unit (CCU) which performs Local-level scheduling by using a new advance reservation mechanism which uses a resource quality determination technique to prioritize candidate resources according to a multi-criteria decision-making problem space which requires balancing different QoS related parameters. The software simulation has been used to evaluate the performance of the proposed system. The results were compared with the DLS, the HEFT, the QRS and the AWS algorithms as some of the most common methods for DAG scheduling. The simulation results confirmed the performance supremacy of the proposed system in different areas of concern. The future directions for further researches related to the area of this paper are mentioned in the following:

- The using of multi-agent framework to overcome the complexity of managing QoS constrained applications. As the agents can independently work in the heterogeneous environments with the ability to re-act, pro-act, migrate, evolve, and cooperate, they are a perfect match for the mentioned problem.

- Using the prediction mechanisms to foresee the critically of the environment before the crisis happens. Extending this ability will widely improve the reliability.
- Extending rescheduling techniques based on predicted status which will generally result in lower application failure and QoS violations and increase the satisfaction level of Cloud application owners.

Acknowledgements This research is supported by Iran Telecommunication Research Center (ITRC). Our thanks go to Dr. Ali Rezaee who has contributed in this research.

Appendix

For the benefit of readers, the authors summarize in Table 4 the key symbols and their definitions used in this paper.

Table 4 Notation and basic terms used in the paper

Parameter	Description
$\overline{\text{TSD}}_{\text{RT}_\text{A}}^{\text{Cl}_i}$	Average cluster's tasks score density
$\overline{\text{AVL}}_{\text{RT}_\text{A}}^{\text{Cl}_i}$	Average cluster's availability
$\overline{\text{CP}}_{\text{RT}_\text{A}}^{\text{Cl}_i}$	Average cluster's computation power
$\overline{\text{RR}}_{\text{RT}_\text{A}}^{\text{Cl}_i}$	Average cluster's resource type request rate
CRTQV	Cluster resource type quality value
$C(e_{ij})$	Communication cost between two tasks, t_i and t_j
$\text{AEC}_{t_i}^{\text{RT}_\text{A}}$	Average execution cost of task t_i on all available clusters
$\text{GACP}_{\text{RT}_\text{A}}$	Grid average computation power for resources of type A
DAG.Succ_{t_i}	Count of task successors
$T_\text{p}^{t_i}$	Time pressure of task t_i
$\text{EST}_{t_i}^{\text{RT}_\text{A}}$	Estimated start time of task t_i
$\text{MBW}^{(\text{Cl}_i,\text{Cl}_j)}$	Maximum bandwidth between clusters Cl_i and Cl_j when there is no communication load between them
$\text{UBW}_k^{(\text{Cl}_i,\text{Cl}_j)}$	Used bandwidth between clusters Cl_i and Cl_j during the time k
CCL	Candidate cluster list
$\text{EFT}_{t_j}^{\text{RT}_\text{A}}$	Estimated finish time of task t_j
$\text{AST}_{t_j}^{\text{Cl}_x}$	Actual start time of task t_j in cluster Cl_x
$\text{AFT}_{t_j}^{\text{Cl}_x}$	Actual finish time of task t_j in cluster Cl_x
$\overline{\text{GRR}}_{\text{RT}_\text{A}}$	Grid resource average request rate
$\text{OCV}_{\text{DAG}}^{t_i}$	Overall criticality value of task t_i
$\overset{N}{C}(e_{ik})$	Normalized value of $C(e_{ij})$
$\overset{N}{\text{Succ}}_{t_i}$	Normalized value of Succ_{t_i}
$\overset{N}{\text{AEC}}_{t_i}^{\text{RT}_\text{A}}$	Normalized value of $\text{AEC}_{t_i}^{\text{RT}_\text{A}}$
$\text{PartitionRank}_{t_i,t_j}^{\text{Cl}_x,\text{Cl}_y}$	Partition rank value for tasks t_i and t_j, if cluster x and cluster y are selected for assigning task t_i and task t_j to them, respectively
$\text{PartitionCriticality}_{i,j}^{x,y}$	Partition criticality value, if cluster x and cluster y are selected for assigning task t_i and task t_j to them, respectively
$\text{CC}_{i,j}^{x,y}$	Communication cost between cluster x and cluster y for task t_i and task t_j
$L_{i,j}^{x,y}$	The amount of time between estimated finish time of task t_j and latest possible start time of task t_i, if task t_i is scheduled on cluster x and task t_j is scheduled on cluster y
β	The maximum of two CRTQV values
$\text{LST}_{t_j}^{\text{Cl}_y}$	Latest time that task t_i can start its execution if it is scheduled on cluster Cl_y
$\sigma L_{i,j}^{x,y}$	Normalized value of $L_{i,j}^{x,y}$
$\sigma\text{CC}_{i,j}^{x,y}$	Normalized value of $\text{CC}_{i,j}^{x,y}$
$\text{Util}_r^{\text{Cl}_i}$	Resource's utilization
$\text{STG}_r^{\text{Cl}_i}$	Resource's sum of task gap
$\text{MCCP}(t_i)$	Maximum communication cost with predecessors for task t_i
$\text{Fail}_r^{\text{Cl}_i}$	Ratio of failed tasks to all assigned tasks on resource r

References

1. Castillo C, Rouskas GN, Harfoush K (2007) On the design of online scheduling algorithms for advance reservations and QoS in grids. In: IEEE international parallel and distributed processing symposium, IPDPS
2. Wieczorek M, Prodan R, Fahringer T (2005) Scheduling of scientific workflows in the ASKALON grid environment. ACM SIGMOD Rec 34(3):56–62
3. Ramakrishnan A, Singh G, Zhao H, Deelman E, Sakellariou R, Vahi K, Blackburn K, Meyers D, Samidi M (2007) Scheduling data intensive workflows onto storage-constrained distributed resources. In: Proceedings of the 7th IEEE symposium on cluster computing and the grid (CCGrid'07)
4. Yu J, Buyya R (2005) A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec 34(3)
5. Sih GC, Lee EA (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans Parallel Distrib Syst 4(2):75–87
6. Kwok W et al (1996) Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans Parallel Distrib Syst 7(5):506–521
7. Topcuoglu H et al (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
8. Cheng J, Zeng G (2011) A two-phase approach to process partitioning for execution optimization migrating workflow. J Comput Interdiscip Sci 7:3478–3490
9. Tan W, Fan YS (2007) Dynamic workflow model fragmentation for distributed execution. Comput Ind 58(5):381–391
10. Maurino A, Modafferi S (2005) Partitioning rules for orchestrating mobile information systems. Pers Ubiquitous Comput 9(5):291–300
11. Baresi L, Maurino A, Modafferi S (2005) Workflow partitioning in mobile information systems. Int Fed Inf Process 158:93–106
12. Liu B, Wang Y, Jia Y, Wu QY (2005) A role-based approach for decentralized dynamic service composition. China J Softw 16(11):1859–1867
13. Daoud MI et al (2011) A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. J Parallel Distrib Comput 71(11):1518–1531
14. Omara FA et al (2010) Genetic algorithms for task scheduling problem. J Parallel Distrib Comput 70(1):13–22
15. Sinnen O et al (2011) Contention-aware scheduling with task duplication. J Parallel Distrib Comput, 77–86
16. Dong F (2009) Workflow scheduling algorithm in grid. PhD thesis
17. El-Rewini H, Lewis T, Ali H (1994) Task scheduling in parallel and distributed systems. PTR Prentice Hall, New York. ISBN:0130992356
18. Wong K, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv 31(4):406–471
19. Casanova H et al (2010) On cluster resource allocation for multiple parallel task graphs. J Parallel Distrib Comput 70(12):1193–1203
20. Deelman E, Mehta G, Singh G, Su M-H, Vahi K (2007) Pegasus: mapping large-scale workflows to distributed resources. In: Taylor I, Deelman E, Gannon DB, Shields M (eds) Workflows for e-science: scientific workflows for grids. Springer, Berlin
21. Deelman E, Singh G, Su M-H, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program 13:219–237
22. Pegasus. http://pegasus.isi.edu
23. Dong F, Akl SG (2007) Distributed double-level workflow scheduling algorithms for grid computing. J Inf Technol Appl 1(4):261–273
24. Prodan R, Wieczorek M (2010) Bi-criteria scheduling of scientific grid workflows. IEEE Trans Autom Sci Eng 7(2):364–376
25. Duan R, Prodan R, Fahringer T (2007) Performance and cost optimization for multiple large-scale grid workflow applications. In: Proc of the 2007 ACM/IEEE conference on supercomputing, pp 1–12
26. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program 14(3, 4):217–230
27. Chen WN, Zhang J (2009) An ant colony optimization approach to grid workflow scheduling problem with various QoS requirements. IEEE Trans Syst Man Cybern 39(1):29–43

28. Tao Q, Chang H, Yi Y, Gu C, Yu Y (2009) QoS constrained grid workflow scheduling optimization based on a novel PSO algorithm. In: Eighth international conference on grid and cooperative computing, pp 153–159
29. Salehi MA, Buyya R (2010) Adapting market-oriented scheduling policies for cloud computing. In: Proceedings of the 10th int'l conference on algorithms and architectures for parallel processing, ICA3PP 2010, pp 351–362
30. Pandey S, Wu L, Guru S, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 24th IEEE international conference on advanced information networking and applications, AINA, pp 400–407
31. Xu M, Cui L, Wang H, Bi Y (2009) A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In: IEEE international symposium on parallel and distributed processing with applications, pp 629–634
32. Ostermann S, Prodan R, Fahringer T (2010) Dynamic cloud provisioning for scientific grid workflows. In: 11th IEEE/ACM international conference on grid computing, GRID, October 2010, pp 97–104
33. Byun E-K, Kee Y-S, Kim J-S, Deelman E, Maeng S (2011) BTS: resource capacity estimate for time-targeted science workflows. J Parallel Distrib Comput 71(6):848–862
34. Byun E-K, Kee Y-S, Kim J-S, Maeng S (2011) Cost optimized provisioning of elastic resources for application workflows. Future Gener Comput Syst 27(8):1011–1026. [Online]. Available http://www.sciencedirect.com/science/article/pii/S0167739X11000744
35. Chen WN et al (2009) An ant colony optimization approach to grid workflow scheduling problem with various QoS requirements. IEEE Trans Syst Man Cybern 39(1):29–43
36. Klir GJ (1995) Fuzzy set and fuzzy logic: theory and application. Prentice-Hall, Englewood Cliffs
37. Ross TJ (1995) Fuzzy logic with engineering applications. McGraw-Hill, New York
38. Kruatrachue B (1987) Static Task Scheduling and Grain Packing in Parallel Processing Systems. PhD thesis, Oregon State University
39. Castillo C et al (2011) Online algorithms for advance resource reservations. J Parallel Distrib Comput, 963–973
40. Tang X et al (2010) List scheduling with duplication for heterogeneous computing systems. J Parallel Distrib Comput 70(4):323–329
41. Zhao L, Ren Y, Li M, Sakurai K (2012) Flexible service selection with user-specific QoS support in service-oriented architecture. J Netw Comput Appl 35(3):962–973
42. Chunlin L, Xiu ZJ, Layuan L (2009) Resource scheduling with conflicting objectives in grid environments: model and evaluation. J Netw Comput Appl 32(3):760–769
43. Abawajy JH (2009) Adaptive hierarchical scheduling policy for enterprise grid computing systems. J Netw Comput Appl 32(3):770–779
44. Kangas J, Kangas A, Leskinen P, Pykalainen J (2001) MCDM methods in strategic planning of forestry on state-owned lands in Finland: applications and experiences. J Multi-Criteria Decision Anal, 257–271
45. Saaty TL (1994) Fundamentals of decision making and priority theory with the analytic hierarch process. The analytic hierarch process series, vol VI. RWS, Pittsburgh
46. Daoud MI et al (2008) A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J Parallel Distrib Comput 68(4):399–409
47. Taylor I, Deelman E, Gannon D, Shields M (2006) Workflows in e-science. Springer, Berlin
48. Afgan E, Bangalore P, Skala T (2012) Scheduling and planning job execution of loosely coupled applications. J Supercomput 59(3):1431–1454
49. Li C, Li LY (2012) Optimal resource provisioning for cloud computing environment. J Supercomput 62(2):989–1022
50. Falzon G, Li M (2012) Enhancing genetic algorithms for dependent job scheduling in grid computing environments. J Supercomput 62(1):290–314
51. Luo J, Wu Z, Cao J, Tian T (2012) Dynamic multi-resource advance reservation in grid environment. J Supercomput 60(3):420–436
52. Bradley A, Curran K, Parr G (2006) Discovering resources in computational grid environments. J Supercomput 35(1):27–49
53. Cao J, Spooner DP, Jarvis SA, Nudd GR (2005) Grid load balancing using intelligent agents. Future Gener Comput Syst 21(1):135–149. Special issue on intelligent grid environment: principles and applications