Cluster Computing (2019) 22:1085–1106
https://doi.org/10.1007/s10586-018-02888-9

A reliable, TOPSIS-based multi-criteria, and hierarchical load balancing method for computational grid

Aref M. Abdullah[1] · Hesham A. Ali[2] · Amira Y. Haikal[2]

Received: 22 March 2017 / Revised: 25 March 2018 / Accepted: 15 December 2018 / Published online: 1 January 2019

Abstract
Load balancing is a very important and complex problem in computational grids. In load balancing, jobs should be effectively distributed among resources in order to minimize the average completion time and maximize the utilization of all resources even those with low reliabilities and capacities. However, using the less reliable and slow resources implies worse completion time, whereas always selecting the powerful and reliable resources undermines the utilization of other resources. So, it is essential to develop an efficient load balancing method which makes a good tradeoff between these criteria in a way that satisfies the quality of service of jobs and fairly distributes jobs between resources based on their reliabilities and capacities. This paper proposes an efficient multicriteria load balancing method using technique for order preference by similarity to ideal solution which treats load balancing as a multi criteria decision making problem. Also, an effective weighting mechanism is proposed, which adaptively adjusts the weights of the considered criteria according to the system's current state and jobs' characteristics. This mechanism can make an efficient tradeoff between the considered criteria and accurately reflect the importance of each one. By simulation, the proposed method was evaluated and compared with other approaches from the literature. In the range of examined parameters' values, the simulation results show that proposed method minimizes the average completion time by 8.7–15.7%, increases the throughput ratio up to 15.8–19.4%, and maximizes the load balancing level by 7.68–20.1%.

Keywords Computational grid · Grid scheduling · Load balancing · Fault tolerance · Distributed system

1 Introduction

Grid computing is a form of distributed systems which utilizes a large pool of computing resources to provide a geographically distributed powerful platform that caters the need of massively computational applications [21]. In computational grids, uneven arrival rates of jobs and heterogeneity of resources usually lead to a situation when some resources are overloaded with many jobs while others are underutilized or even idle [12, 14, 15, 38]. So, the objective of load balancing is to dynamically balance the workload between resources in order to enhance resource utilization, minimize the average completion time, and increase throughput [30].

The allocation and load balancing algorithms can be classified as centralized, distributed, and hierarchical. Centralized approaches suffer from limited scalability, and are intrinsically characterized by a single point of failure because a single entity is responsible for all allocation and load balancing decisions [23, 29]. Distributed approaches improve scalability and fault tolerance since all resources are involved in decision making activities. However, it is costly to let each resource get and maintain the dynamic information of the whole system [24]. Also, distributed schedulers may take conflicting decisions regarding the use of resources [32]. In the hierarchical approaches, decision making entities are organized in two or more hierarchical levels. Hierarchical approaches give more efficient and

✉ Aref M. Abdullah
eng.aref77@gmail.com

Hesham A. Ali
h_arafat_ali@mans.edu.egp

Amira Y. Haikal
amirayh@gmail.com

[1] Taiz University, Taiz, Yemen

[2] Mansoura University, Al-Daqhaliya, Egypt

conflict-free schedules than do the distributed approaches. However, they may suffer from a bit lack of scalability if both the number of levels and the master entities are not efficiently selected.

Although the resource allocation problem in computational grids has been intensively studied, the majority of the previously proposed studies do not consider the load balancing during the allocation process [13]. They always send jobs to the most prominent resources which are capable of speeding up their executions whereas the idleness and underutilized state of the other resources are ignored. As a result, the fittest resources are often overwhelmed with many jobs when other resources remain idle or lightly loaded. So, frequent job redistributions are required by those approaches to balance the workload among resources and thus improving the utilization of each resource. Although the jobs redistribution process is sometimes necessary in highly dynamic grids even though a good allocation is drawn, making a good placement during the allocation process will reduce the jobs' redistribution rate. As a result, the communication overhead will be minimized and thus the overall performance of the system will be improved.

Furthermore, most of the previously developed load balancing approaches failed to capture the complexity of a truly-open grid which is composed not only of dedicated multisite clusters, but also of a huge set of non-dedicated individual resources which can fail or leave the system at any time. For example, those approaches do not consider diversified reliabilities and intermittent availabilities of resources when balancing the workload between resources. So, applying them in such an environment cannot efficiently utilize the grid resources and may make the participation of intermittently available and fault-prone resources worthless. More importantly, ignoring the reliability of resources may lead to a situation in which the more powerful but unreliable resources are overloaded with jobs; or jobs are also allocated to the resources whose expected survival times less than their time constraints. In fact, the failures of such overloaded resources severely affect the performance of the system because a large number of jobs should be redistributed to other resources (if not replicated), which implies that their completion times will be increased. The case will be worse if those jobs are non-checkpointable since they have to be restarted from scratch losing all the computational work done. All of these may prevent jobs from completing their executions within their specified deadlines.

Unfortunately, the large variety of resources reliabilities and capacities makes it difficult to always find resources which are fast and highly reliable at the same time. So, scheduling only for reliability undermines the performance in term of throughput, resource utilization, and job execution speed-up. Whereas, scheduling only for speeding up the completion time can be detrimental due to the performance ramifications of resource failures [33]. For this reason, grid schedulers must make a good tradeoff between resource reliability, resource utilization, and the computational capacity of resources taking into account the QoS of submitted jobs such as deadlines.

To solve the aforementioned problems, this paper introduces a reliable and efficient allocation and load balancing method which considers the unique characteristics of grid resources and jobs' deadlines, and fairly distributes the workload between resources in a way that ensures the participation of all resources in jobs executions even those that are slow and less reliable. By "fairly", I mean that the workload should be distributed among resources proportionally to their reliabilities and capacities so that more powerful and reliable resources (with larger MTBFs) execute much work while slower and less reliable ones execute less.

In fact, the fair distribution of jobs among the resources of a grid system is beneficial in many aspects. (1) From an economic aspect, it ensures that all the resources in the system are utilized so that all the resource providers can make benefits. (2) It avoids overloading resources with jobs, thus it minimizes the execution losses of the jobs in case of resource failures; and also reduces the communication overhead induced by retransferring the failed jobs to new resources. (3) Allocating small and non-urgent jobs to the slow resources as long as they can be executed there within their time constraints saves the faster resources for longer and urgent jobs (jobs with early deadlines). All of these will maximize resource utilization, increase system throughput, and satisfy the QoS of the submitted jobs.

The proposed method treats the allocation and load balancing as a MCDM problem because of the effect of different criteria of jobs and resources on the decision making, when some of these criteria are conflicting to each other. The main contributions of this paper are:

- Proposing a reliable, adaptive, multi-criteria, and hierarchical load balancing method based on TOPSIS technique, in which the resources are prioritized according to the following weighted attributes (criteria):
  - Resource capacity (which is represented by the estimated completion time of a job on the resource)
  - Resource reliability
  - Resource load index
- Making a good tradeoff between these criteria is also very important to make accurate decisions and obtain a good performance from the system's and user's perspectives. So, this paper also proposes an effective

weighting mechanism which adaptively and transparently adjusts the weights/importance of the criteria according to job's characteristics and the system's current state.
– Providing a redistribution policy which continuously monitors and redistributes jobs between resources to minimize their load differences.

Remainder of this paper is structured as follows: Related work is reviewed in Sect. 2. The system model and the proposed allocation and load balancing method are introduced in detail in Sect. 3. Section 4 is dedicated to fault tolerance and reliability model. Section 5 focuses on the simulation setup, performance criteria, and the evaluation results. Finally, Sect. 6 concludes the paper and presents future directions.

2 Related work

In the literature, there are several dynamic scheduling and load balancing algorithms proposed for computational grids [2, 11, 13, 20, 25, 29, 34, 40, 42]. They differ in the way they tackled the load balancing and also in what issues and characteristics of grids they considered.

Yagoubi and Slimani [42, 43] presented a load balancing strategy for multi-cluster grids in which a cluster consists of several sites, each connects together several heterogeneous resources. The proposed strategy designed to perform at three levels, namely: intra-site level, intra-cluster level, and intra-grid level. The load information is gathered periodically from the children resources only to their associated managers at any level, so the number of messages exchanged for the purpose of load balancing is decreased, and thus the communication overhead. This strategy is scalable and privileges local load balancing first (within a site, then within a cluster and finally on the whole grid). However, it does not provide any job allocation procedure, and does not take into consideration the reliabilities of resources when balancing the workload between resources.

Lu et al. [25, 26] proposed a dynamic load balancing scheme which considers the concerns of scalability, heterogeneity, and communication overheads. In this scheme, a grid scheduler runs on each resource which maintains two sets of resources: $LPSet_i$ and $LNSet_i$. The set $LPSet_i$ contains p resources as its partners and the set $LNSet_i$ contains n resources as its neighbors. The scheduler uses the set $LPSet_i$ to select a partner resource for executing a new arriving job while the set $LNSet_i$ is used to select a neighbor resource for offloading jobs. The selection of the candidate resources for a job allocation and redistribution is restricted to these two sets and depends on some parameters which must be taken carefully to get a good performance. In this scheme, each resource appends the load information of itself and some randomly selected resources from its partners and neighbors to each job transfer request; and it gets the information of other resources with job acknowledges or completion replies. The disadvantage of this information policy is the high probability of sending useless information which may be a potential source of communication overheads. That happens because the randomly selected resources do not always belong to receiver's neighbors or partners. This scheme does not consider network and hardware failures, and there is a high probability that the resources with light load and large capacity may be overwhelmed by their neighbors if simultaneous jobs migrations occurred due to the lack of coordination between resources.

Balasangameshwara and Raju [2] proposed an adaptive and performance-driven load balancing approach which considers heterogeneity, dynamicity, and resource failure. It extended the information policy proposed in [25, 26] by incorporating the resource efficiency and also making the job transfer request simple and small sized. However, the high probability of sending useless information was not addressed. To tolerate resource failures, two techniques are used. The first is the replication in which each job is scheduled on two resources, one is a primary and the second is a backup, where the backup copy is activated only if a fault occurs during its primary execution. In the second technique, historical health indices of resources are considered during the allocation process, and the resource with the highest index is selected for a job execution. In this approach, the execution cost of jobs is not considered when redistributing the load between resources. Also, resources with light loads and large capacities may be overwhelmed by their neighbors drastically if simultaneous jobs migrations occurred due to the lack of coordination.

Golmohammadi and Shahhoseini [13] proposed a centralized, economic based, and dynamic allocation method whose aim was to balance the load between resources and complete jobs within their time constraints. This method treats the allocation process as a multi criteria decision making problem. It uses the analytic hierarchy process (AHP) technique to rank resources and then select the most appropriate ones to execute jobs. The selection criteria used by this method are MIPS rating (rp), resource cost (rc), number of PEs (pe), average waiting time (wt), and resource utilization (ru). However, the proposed method has many drawbacks. For example, it does not consider the communication cost and resource reliability. Also, it does not scale for large-scale grids because of the centralized scheme it follows and due to the many pair-wise comparisons which have to be done between resources with respect to criteria.

Subrata et al. [39] studied the suitability of using a genetic algorithm (GA) and a tabu search algorithm (TS) for solving the load balancing problem in grid environments. The two algorithms take into consideration the heterogeneity and dynamicity. However, they have two drawbacks. The first drawback appears in the extra storage and scheduling overhead. The other drawback is that the fault tolerance is not taken into the account.

El-zoghdy and Aljahdali [11] proposed a two-level load balancing method for computational grids. They studied the load balancing problem only at the steady-state mode in which the number of jobs sent to the grid is sufficiently large and the arrival rate of jobs does not exceed the grid overall processing capacity. This algorithm does not take into consideration the resource failure and the heterogeneity of both communication links and jobs. Also, it does not provide any mechanism for information exchange and load redistribution.

Santiago et al. [34] proposed a centralized, multi criteria, and fuzzy-based meta-scheduler for a computational grid consisting of a set of nodes each with several computational resources. This algorithm gives option to the slow nodes, that are often left unused, to participate in jobs' executions. So, the nodes with more computing power execute fewer jobs than when using the common vision of load balancing, and thus ensuring not overloading them. For this purpose, the scheduler selects a node based on its factor of selection metric which is the output of a proposed fuzzy interference system where five parameters are used as inputs to introduce it. These parameters are: the rate of available resources, rate of idle MIPS, length of executed jobs, execution time of jobs, and the number of executed jobs. Santiago et al. did not consider the transmission time of jobs, and also ignored resource failures. They also did not provide any mechanism for monitoring and redistributing jobs between the overloaded and underloaded resources.

Helmy et al. [15] proposed a dynamic, hierarchical, and fuzzy-based load balancing scheme which allocates jobs upon their arrival in a balanced manner. To accurately estimate the load of a node or a cluster, four parameters are used as input variables for the proposed fuzzy-based scheme (namely; node's ready queue length, burst time, CPU utilization, and available resources needed to accomplish the jobs), while the output represents the current workload state. This scheme does not consider the heterogeneity of resources, communication cost, and resource reliability. It also does not provide any mechanism to continuously redistribute jobs between the overloaded and underloaded resources.

Suresh and Balasubramanie [40] proposed a dynamic hierarchical load balancing algorithm for a computational grid consisting of a finite set of resources; each with several machines. Each machine, in turn, consists of several processing elements (PEs). This algorithm focuses on satisfying the deadlines of jobs and achieving better load balancing. A user submits their jobs to its machine which assigns them to its corresponding PEs according to their current loads and the capability of completing them within their deadlines. If there is no suitable PE found for a given job, the submission machine forwards the job to its resource. If there is no suitable PE found at all machines under that resource, the job is forwarded to the resource broker at the top which then assigns it to a suitable resource from the other resources under its control. To perform load balancing, the resources/machines/PEs are grouped into overloaded, normally loaded, and underloaded lists upon the arrival of a new job. Then, the job is assigned to a node from the underloaded list which satisfies its requirements. The major advantage of this work is that the distribution of job submissions between machines instead of using a central entity which leads to a lower bottleneck. However, its disadvantages are that the resource reliability is not considered, and there is a high probability that some machines/resources may be overwhelmed with jobs when the others are lightly loaded or even remain idle.

Prez-Miguel et al.[31] proposed failure-aware scheduling techniques for high-throughput computing systems. These techniques has the same vision as our method proposed in this paper since they both provide mechanisms which match the expected completion times of jobs with the expected survival time of nodes. In those techniques, several nodes can contend for the execution of the same job, and the owner node, based on a certain score, is chosen to execute the job. The main drawbacks of those techniques are that they do not consider the communication time required to transfer jobs to nodes, and do not provide any model to estimate the completion time.

Table 1 summarizes the features and limitations of the load balancing approaches discussed in this section.

In fact, almost none of the already existing methods consider all the characteristics of grid resources and jobs which directly affect the performance of the load balancing from both the system's and user's perspectives. Such characteristics include: resources reliability, heterogeneity of resources and communication links, scalability, and job deadline, etc. Consequently, they are not capable of effectively balancing the workload between resources in computational grids. Also, in these methods, load balancing is achieved by always distributing the workload to resources proportional to only their capacities while their reliabilities are ignored. So, applying them in grids often leads to a condition where the unreliable resources, which have high capacities, may be overloaded with jobs when slower but reliable resources are left unused or underutilized.

Table 1 Comparison between the load balancing approaches

The approach	Type	Features	Limitations
Yagoubi and Slimani [42, 43]	Tree-based Heuristic	Considers resource heterogeneity, scalability, and communication cost	Does not provide any job allocation procedure. Ignores resource reliability
Lu et al. [25, 26]	Neighbor-based heuristic	Considers scalability and heterogeneity	Does not take into account the reliability of resources when balancing the workload
Balasangameshwara and Raju [2]	Neighbor-based heuristic	Adaptive, and scalable Considers heterogeneity, dynamicity, and failure tendency of resources	Execution cost is not considered
Golmohammadi and Shahhoseini [13]	Multi criteria AHP-based heuristic	Achieves high fairness Considers resource heterogeneity	Not scalable. Does not consider resource reliability and communication cost
Subrata et al. [39]	Meta-heuristic optimization approach	Addresses heterogeneity and dynamicity	Incurs extra storage and processing requirements Considers fault tolerance
El-zoghdy and Aljahdali [11]	Tree-based heuristic	Low decision making overhead and scalable Considers resources heterogeneity	Does not consider resource reliability and heterogeneity
Santiago et al. [34]	Multi criteria fuzzy-based heuristic	Considers heterogeneity, and gives option to all resources to participate in the execution of jobs	Does not consider communication cost and resource failure Does not provide any redistribution policy
Helmy et al. [15]	Multi criteria fuzzy-based heuristic	Gives option to all resources to participate in the execution of jobs	Does not consider heterogeneity, communication cost, and fault tolerance Does not provide any job redistribution policy
Suresh and Balasubramanie [40]	Tree-based heuristic	Considers heterogeneity, scalability, and communication overhead Respects job deadline	The resource reliability is not considered when distributing the load between resources
Prez-Miguel et al. [31]	Pool-based heuristic	Considers heterogeneity and resource failure Respects time constraints of jobs	Does not consider communication cost Does not provide any model to estimate the job completion time

The proposed method in this paper tries to tackle the limitations in the existing approaches. It considers the load balancing issue at allocation process, and it provides a load redistribution policy. It also addresses the concern of heterogeneity, differences in the reliability of resources; and focuses on satisfying both the user's and the system's requirements. In the proposed method, the workload is distributed to resources based on jobs' characteristics, resource reliability, resource capacity, and the system's current load balancing state in a way that ensures that all resources participate in the executions of jobs even those with slow speeds and less reliabilities.

3 Proposed method

3.1 System model

In this section, a grid organization model is firstly introduced. Then, the job model and the scheduling components used by the proposed load balancing method are discussed. Table 2 lists the notations used throughout this paper.

3.1.1 Grid hierarchy architecture

In this paper, we used the grid model proposed in our previous work [1]. In this model, the grid is defined as an enormous set of dynamic, heterogeneous, and geographically-distributed computing resources which are connected by various heterogeneous networks (including internet, WAN, LAN, etc.). These resources are mapped into an n-level dynamic hierarchical structure where the number of levels is dynamically determined based on the number of available resources. This model consists of a root level (denoted by L_n) which includes only the top root, a leaf level at the bottom (denoted by L_1) which contains the leaf resources, and a set of $n-2$ intermediate levels in-between. The architecture of the grid model is depicted in Fig. 1. Every resource R_i from level i along with its direct children resources residing in level $i-1$ form a cluster

Table 2 The notations

Notation	Description
$CoM(i)$	The community i which includes all resources in the subtree rooted by the resource R_i except R_i itself
$Cap(i)$	The capacity of R_i measured as MIPS (mega instructions per seconds)
Jl_j	The computational length of job j measured as MB (mega byte)
D_j	Deadline of the job j
IS_j	The size of the input file of job j in MB
OS_j	The size of the output file of job j in MB
GS_i	The global scheduler of R_i which is responsible for scheduling jobs to remote resources/communities
LQ_i	The local queue of R_i which contains all waiting jobs assigned to be executed on R_i
GQ_i	The global queue of resource R_i which contains all jobs waiting to be scheduled by GS_i
$TD(i, j)$	Link transfer delay between resource R_i and R_j.
$BW(i, j)$	The baud rate between R_i and resource R_j.
$LD(i)$	The current load of R_i
$ALD(i)$	The average load of resources in the cluster rooted by R_i
$Ch_m(i)$	The child-resource m of resource R_i.
$CoMCap(i)$	The estimated total capacity of all resources in $CoM(i)$
$CoMALD(i)$	The average load of $CoM(i)$ which is estimated as average load of time required to execute all unfinished jobs which have been allocated to resources in $CoM(i)$
$CT(j, c)$	The estimated completion time of job j on R_c
$CT(j, CoM(c))$	The estimated completion time of job j on community $CoM(c)$
ACT	The actual average completion time of all jobs

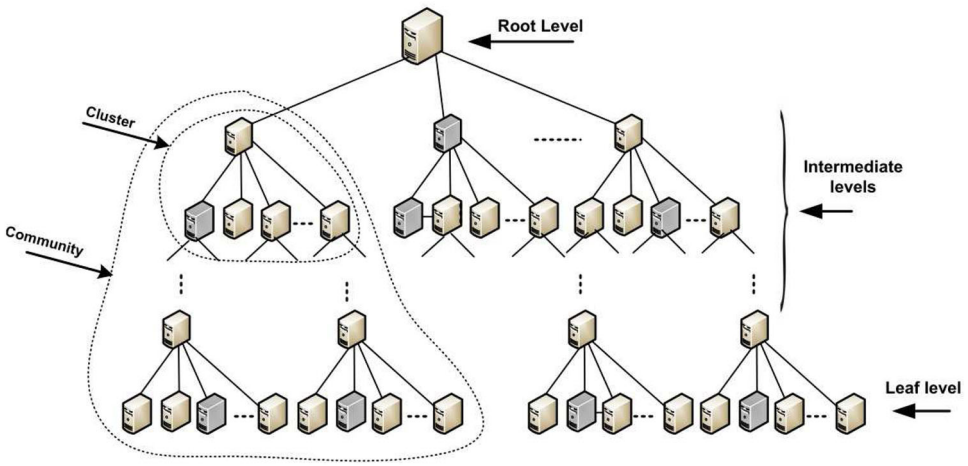

Fig. 1 The proposed dynamic n-level architecture model for computational grid

with R_i acting as its master. Each cluster contains up to k resources where k is a predefined number. Furthermore, each R_i and all resources inside the subtree rooted by itself form a community. The master of each cluster is replicated on one of its children to avoid a single point of failure. This replica will take the master's position when it fails. When a new resource wants to participate in the grid, it queries a predefined host called the entry host (EH) that is aware of the current structure of the tree. EH then responds with some information which is used by the new resource to guide itself joining the closest cluster in term of communication latency.

In the proposed model, each resource can act as both a computing resource (for executing the jobs submitted to it.) and at the same time as a master/manger for the resources in the cluster rooted by itself if any.

To help the scheduler make good decisions, information about resources should be continuously gathered [45]. Here, each resource sends its detailed information (such as speed, load, reliability, and baud rate) to the grid information system (GIS) in its master. Also, it sends some

aggregated information about the resources in the community rooted by itself. Network Weather Service (NWS) [41] and Globus Monitoring and Discovery System (MDS) [8], which are two popular tools of GIS, can be utilized to collect such information.

To be more clear, the proposed work assumes that each computing resource contains only a single processor which operates in a space sharing mode. This mode allows the execution of only a job at a given time, while the other jobs are waiting in the queue. However, the proposed model can be easily extended to accommodate the case when the resource contains several processors and also supports the time sharing mode in which several jobs can be simultaneously executed on a single processor.

This model improves scalability since each resource only manages and maintains the states of a limited number of other resources. The hierarchical structure is also important for localizing the effect of failures [3].

3.1.2 Job model

The jobs are assumed to be independent, computationally intensive, and require no constrained order of their executions. Jobs arrive at global schedulers with different arrival rates, so jobs at different resources can be scheduled in parallel. Jobs differ in their execution and data transmission times. This means that each job requires a different execution time and needs a different time to be transferred to its allocated resource. Each job has three attributes: Job length in MI (million instructions), the size of input and output files expressed in MB (mega byte), and the user deadline (D_j) [2].

3.1.3 Scheduling component

As explained above, each resource in level L_i can act as a child for its upper cluster and at the same time as a master for a cluster at level $L_i - 1$. Thus, it is responsible for both execution and management duties. The scheduling part in each resource consists of the following components as depicted in Fig. 2:

- *Local Scheduler (LS)*: Is responsible for sharing the space of the resource's CPU among all the jobs which have been allocated to execute locally. The jobs waiting for execution are queued in the local queue (*LQ*).
- *Global Scheduler (GS)*: Is responsible for fairly distributing the jobs queued in its global queue (*GQ*) inside the cluster/communities rooted by the resource if they can be executed there within their deadlines. Otherwise, it redirects them to the higher level.
- *Grid Information System (GIS)*: Its role is to maintain the static and dynamic information about resources.

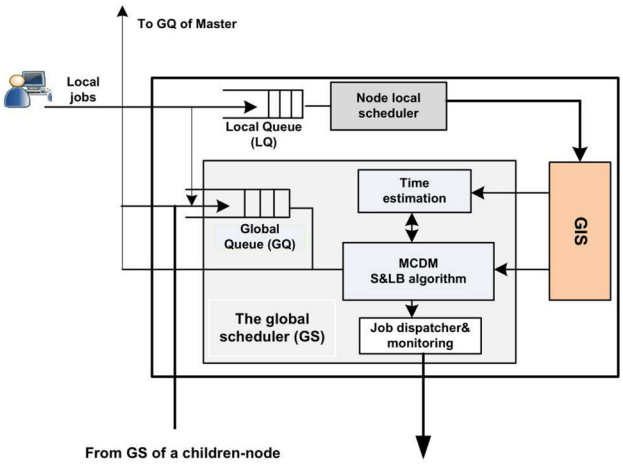

Fig. 2 The scheduling components

3.2 Scheduling policy

Jobs are submitted locally by grid users using their own local resources, and only the jobs that cannot be executed locally within their time constraints are directed to the global scheduler (*GS*) in the same resource, which can be scheduled later as follows:

- Based on a first come first served basis, the GS_i of R_i repeatedly pops jobs from the head of its global queue (GQ_i). For each job, the GS_i first checks if it is possible to execute it within its deadline inside the cluster it manages. If so, the GS_i will utilize the dynamic information of the children resources to create a list of the candidate resources which can execute the job within its deadline provided that the MTBF of each is greater than the estimated completion time. This list will then be sent to the MCDM-based scheduling and load balancing algorithm which uses it to rank the resources and then select the best one to execute the job as discussed later. Finally, the job will be dispatched to the local queue (*LQ*) of the selected resource where it will actually be executed until completion if it is not rescheduled in case of a failure.
- If there is no suitable resource found in the cluster, the GS_i will utilize the aggregated information of the communities rooted by their children to find whether it is possible to execute the job there within its deadline. If so, the least-loaded community will be chosen and the job will then be dispatched to the *GQ* of the child which is the root of the chosen community. The GS there, in turn, will first try to allocate a suitable resource for the job inside the cluster under its control or will forward the job down the hierarchy in the same manner as mentioned above. These steps are iteratively

repeated until finding a suitable resource which actually executes the job.

– If the GS_i at the submission resource found that the job cannot be executed within its deadline either inside its direct cluster or in any community rooted by their children, it will forward the job to the GS of its master. Upon receiving the job, the *GS* at the master acts similarly by firstly trying to allocate a resource for the job from the cluster it manages or forwarding it to the least-loaded community out of those rooted by its children as long as it can be executed there within its deadline. Otherwise, the job will be forwarded again to the higher level. These procedures will be repeated until finding a suitable resource to execute the job or reaching the root level. If the top root is reached and no suitable resource is found, the job will simply be discarded or it can be resubmitted later by the submission resource.

Figure 3 presents the detailed flowchart of the scheduling policy when a job j is submitted.

In contrast to other hierarchical models in which jobs are submitted to a single entity at the top level, the proposed policy distributes the job submission and decision making activities between several entities. This distribution will minimize the communication time incurred by jobs transferring and thus making the proposed model more scalable. The communication overhead will also be decreased since the submitted jobs are tried to be scheduled first to resources nearby to the submission resources.

3.3 Load Index and completion time estimation

Jobs are submitted in the form of gridlets. A gridlet is a package that contains information about the job length expressed in MI, the size of input/output files expressed in MB, and the job deadline D_j. Unlike traditional parallel and distributed systems [22, 37] in which nodes are always connected via a high speed sophisticated communication media, the input and output files in grids need to be transferred through much slower communication links. So, the transfer time should be considered by grid load balancing algorithms in order to select appropriate resources to which jobs are distributed. In addition, information about the candidate resources and communities (such as capacity, current load, baud rate etc.) is required by the global schedulers to calculate the expected completion times which are also used by the allocation and load balancing algorithm to rank and prioritize the potential resources.

In the proposed model, the resource/community capacity and load, which are used to estimate the job completion time, are calculated by the following formulas.

The $LD(c)$ of R_c is estimated as follows:

$$LD(c) = RM + Texec_c \tag{1}$$

where RM is the remaining time of the job being processed on R_c and $Texec_c$ is the total execution time of all jobs waiting in LQ_c which is determined as follows:

$$Texec_c = \frac{\sum_{n=1}^{LQlngth} Jl_n}{Cap(c)} \tag{2}$$

where $LQlngth$ is the number of jobs in LQ_c, Jl_n is the length of the nth job in LQ_c, and cap(c) is the capacity of R_c.

The average load of the resources in the cluster rooted by R_c is calculated by the following formula:

$$ALD(c) = \frac{\sum_{d=1}^{nch(c)} LD(ch_d(c))}{nch(c)} \tag{3}$$

where $LD(ch_d(c))$ is the load of child d of R_c and $nch(c)$ is the number of resources in the cluster.

The average load of community $CoM(c)$ which is rooted by the resource R_c is estimated as follows:

$$CoMALD(c) = \frac{\sum_{n=1}^{GQlngth} Jl_n}{CoMcap(c) + L_c^2 mod2} + Av_c \times \sum_{d=1}^{nch(c)} [LD(ch_d(c)) + CoMALD(ch_d(c))] \tag{4}$$

where $GQlngth$ is the number of jobs in the GQ_c of R_c, L_c is the index of the tree level to which R_c belongs, $CoMALD(ch_d(c))$ is the average estimated load of community rooted by the child d, $CoMCap_c$ is the capacity of the community rooted by R_c, and Av_c is the averaging coefficient of R_c which is equal to:

$$Av_c = \begin{cases} 1 & \text{if} \quad L_c = 1; \\ \dfrac{1}{nch(c)} & \text{if} \quad L_c = 2; \\ \dfrac{1}{2 \times nch(c)} & \text{if} \quad L_c > 2. \end{cases} \tag{5}$$

Or generally as:

$$Av_c = \left| \frac{(2^{2 mod L_c}) - 2}{2^{2 mod L_c} \times [nch(c) + L_c^2 mod2]} \right| \tag{6}$$

The capacity of the community (c) can be determined as:

$$CoMCap(c) = \sum_{d=1}^{nch(c)} [Cap(ch_d(c)) + CoMCap(ch_d(c))] \tag{7}$$

where $Cap(ch_d(c))$ and $CoMCap(ch_d(c))$ are the capacity of the child d of R_c and the capacity of the community rooted by that child respectively.

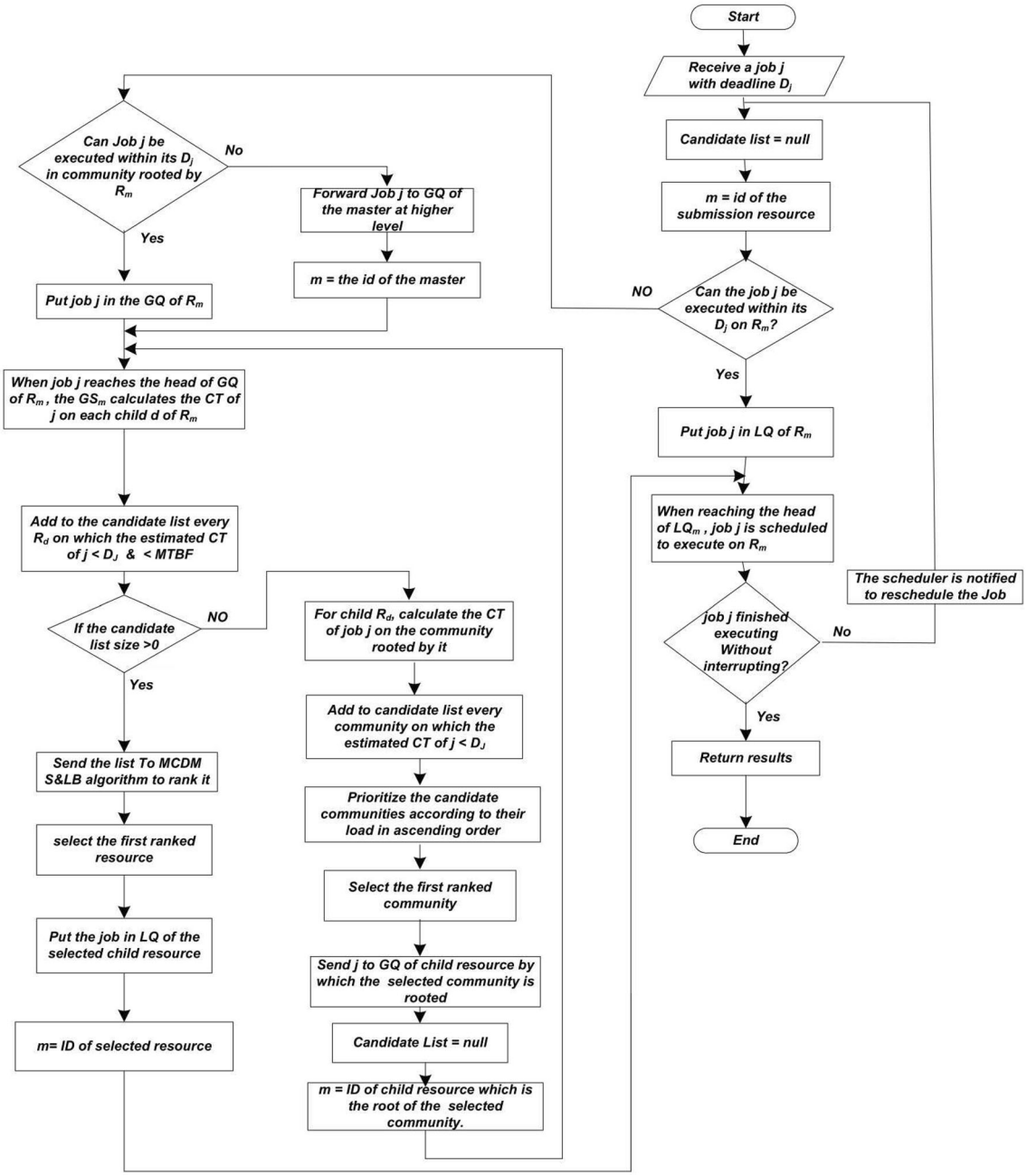

Fig. 3 Flowchart of the scheduling policy

The completion time (CT) of job j can then be estimated on the resource R_c or the community $CoM(c)$ rooted by it using following two functions respectively:

$$CT(j,c) = \frac{Jl_n}{Cap(c)} + \frac{IS_j + OS_j}{BW(c,j)} + LD(c) \tag{8}$$

$$CT(j, CoM(c)) = \frac{Jl_n}{Av_c \times CoMCap(c) + L_c^2 mod2} + \frac{IS_j + OS_j}{BW(c,j)} + CoMCap(c) \tag{9}$$

where the first terms in Eqs. (8) and (9) are the estimated execution times of the job j on R_c and $CoM(c)$ respectively, and the second terms are the estimated time to transfer the job's related files.

It should be noted that the resource load index (LD) is a very important factor which affects the performance of the load balancing algorithms. It is used to measure the load of the resources and also to classify them as overloaded, normally loaded, and underutilized. Several load indices have been used in distributed systems such as: CPU queue

length, average CPU queue length, amount of available memory, etc. However, most of these indices are not good indicators of resource workload in the grid due to the heterogeneity of both jobs and resources. In such an environment, it is essential to adjust the load measures from different resources to make them comparable. For this reason, the resource load index (LD), at a particular instant of time, is defined in this paper (Eq. 1) as the total time required to execute all unfinished jobs which have been allocated to the resource.

3.4 TOPSIS-based allocation with load balancing algorithm

3.4.1 TOPSIS

TOPSIS [16] is a technique for ordering preference by similarity to an ideal solution. It is a kind of MCDM techniques which finds out the best choice among all feasible alternatives in a problem with different and maybe conflicting criteria. The best alternative should have the shortest euclidian distance from the positive ideal solution (PIS) and the farthest distance from the negative ideal solution (NIS). For instance, the PIS maximizes the benefit and minimizes the cost, whereas the NIS minimizes the benefit and maximizes the cost. We chose TOPSIS because of its simplicity, flexibility, its ability to consider a non-limited number of alternatives and criteria, and also for its ability in identifying the best alternative quickly compared with other MCDM techniques. These advantages make it more suitable for the allocation and load balancing problem in grids where several conflicting factors and criteria should be considered when ranking the alternative resources, and then selecting the best ones to which the jobs are distributed. TOPSIS is widely used in many fields, including: financial performance evaluation, supplier selection, location selection, and company evaluation [5, 28].

3.4.2 Criteria selection

It seems preferable to send jobs to resources with more computational capacities, however, always selecting the same eminent resources may lead to a load imbalance since other resources remain underutilized or even idle. Furthermore, considering only the capacities of resources may not always be enough to finish jobs faster or within their deadlines especially in a real grid which contains resources that exhibit different reliabilities and intermittent availabilities. In fact, the completion times of jobs are affected by the reliability status of resources and ignoring reliability may lead to a disastrous situation in which the less reliable resources may get overwhelmed with many jobs. The frequent failures of such overloaded resources will cause many jobs to be rescheduled, thus all CPU cycles spent in their last executions will be lost. Also, a high communication overhead will be caused by resending the failed jobs to new resources. As a result, jobs may not finish their executions within their deadlines.

On the other hand, it is sometimes desired to also send some jobs to the less reliable or slower resources in order to achieve better load balancing and allow all resource providers to make benefits. This also saves the reliable and faster resources for jobs with intensive computational requirements and earlier deadlines.

For these reasons, a good load balancing algorithm must consider and make a good tradeoff between all of these conflicting objectives and factors to ensure that jobs are fairly distributed to resources based on their capacities and reliabilities. In other words, it must ensure that faster and reliable resources execute more jobs, and at the same time it gives option to worse and underutilized resources to participate in execution. To this end, the resources in our TOPSIS-based load balancing algorithm must be prioritized during the decision making for each job according to the following weighted criteria:

- Expected completion time (*CT*): This criterion symbolizes the computational power of each resource and it is highly related with the job deadline. *CT* can be determined through calculating all the time needed to complete the execution of the job based on Eq. 8 (i.e. it includes the transfer time, waiting time, and expected execution time).
- Resource reliability measured by mean time between failures (MTBF).
- Resource load index (*LD*): It is defined, at an instant of time, as the total time required to execute all unfinished jobs which have been allocated to the resource (Eq. 1). This criterion is chosen for ensuring that all the grid resources will participate in the execution of jobs.

3.4.3 Calculation of criteria weights

In MCDM problems, decisions are made based on several criteria of varying importance to decision makers. Each criterion is assigned a weight that usually indicates its importance relative to other criteria under consideration. The derivation of criteria weights is an essential step to obtain effective decisions that meets the preferences of the decision-makers. The higher the weight value, the more dominant the corresponding criterion becomes. The values of weights are often normalized to the sum of 1. For n criteria, a set of weights is defined as $\mathrm{w} = w_1, w_2, w_j, w_n, \sum_{i=1}^{n} w_n = 1$.

There are several weight determination methods found in the scheduling literature such as rating, ranking, and pairwise comparisons [4, 13, 27]; however, they are less efficient in term of memory utilization and time complexity. Also, they do not consider both the user's and system's preferences, dynamicity, and resources reliability. So, they do not suit well for load balancing in highly dynamic grids. Herein, we propose an efficient criteria weighting mechanism in which the weights are adaptively estimated for each job based on its requirements and the system's current state. As a result, it well reflects the importance of each criterion.

This mechanism favors faster and more reliable resources over slower and idler ones for jobs with early deadlines. Under this circumstance, it increases the importance of the *CT* and MTBF criteria while it decreases the importance of the *LD* criterion. In contrast, for those jobs which are short or have late deadlines, the proposed mechanism decreases the importance of the *CT* and MTBF criteria whereas it increases the importance of the *LD* criterion. This adjustment mechanism is expected to maximize the system throughput (i.e. the number of jobs completed within their deadlines) and improve the utilization of each resource since it gives an opportunity to all resources to execute jobs even those with low speeds or less MTBFs. For each job, the proposed mechanism calculates the weights of the used criteria as follows:

1. Calculate the arithmetic mean for the values of every criterion l with respect to all the candidate resources (alternatives) which can finish the job under consideration within its deadline as follows.

$$E_l = \frac{\sum_{i=1}^{n} CV_{il}}{n} \tag{10}$$

where CV_{il} is the value of the criterion l with respect to the alternative resource R_i, and n is the number of the alternative resources in the candidate list. For example, $E_{CT} = \frac{\sum_{i=1}^{n} CT(j,i)}{n}$, where $CT(j, i)$ is the estimated completion time of the job j on R_i and $R_i \in$ the candidate resource list.

2. Find the standard deviation among the values of every criterion with respect to all the alternative resources using the following formula:

$$SD_l = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(CV_{il} - E_l)^2} \tag{11}$$

For example, the *SD* among the values of the *CT* criterion is: $SD_{CT} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(CT_{j,i} - E_{CT})^2}$

3. Finally, the weights of the used criteria, namely the weights of the *CT*, MTBF, and *LD* criteria are determined using the following equations respectively:

$$W_{CT} = \frac{SD_{CT}}{TSD} + \frac{1}{2}\frac{E_{CT}}{D_j} \times \frac{SD_{LD}}{TSD} \tag{12}$$

$$W_{MTBF} = \frac{SD_{MTBF}}{TSD} + \frac{1}{2}\frac{E_{CT}}{D_j} \times \frac{SD_{LD}}{TSD} \tag{13}$$

$$W_{LD} = \frac{SD_{LD}}{TSD} - \frac{E_{CT}}{D_j} \times \frac{SD_{LD}}{TSD} \tag{14}$$

where $TSD = SD_{CT} + SD_{MTBF} + SD_{LD}$ (which is used for normalization) and D_j is the deadline of the job under consideration.

From Eqs. 12, 13 and 14, it is noted that the weight calculation mainly depends on the standard deviation values of the used criteria with respect to the alternative resources. In fact, the standard deviation is a good indicator of how spread out the values of each criterion over the alternative resources is. So, it helps to well estimate the system's current state and then adjust the weights properly. For example, the large standard deviation over the *LD* criterion indicates that there is a high variation in resources loads which certainly implies that the system is in an imbalance state. In this case, the weight of the *LD* criterion will be increased in order to give option to the underutilized resources to participate in the jobs' executions (especially those with low capacities and less reliabilities). The low standard deviation, on the other hand, implies that the system is in a load balanced state. So, the *LD* weight will be decreased in this case. For the *CT* and MTBF criteria, the larger standard deviation means that there is a high variety in the capabilities and reliabilities of the system resources. Hence, the weight of these criteria will be incremented accordingly to increase the probability of selecting a powerful and reliable resource to execute the job under consideration as fast as possible. This ensures that the powerful and reliable resources execute more jobs compared to others.

The average completion time to deadline ratio ($\frac{E_{CT}}{D_j}$) is another important factor used by our mechanism. This ratio is used to adjust a good tradeoff between the importance of the *LD* criterion and those of the other criteria. Specifically, the *LD* weight (W_{LD}) is equal to its normalized standard deviation over the values of the *LD* with respect to all alternative resources decreased by the ratio $\frac{E_{CT}}{D_j}$ of its normalized standard deviation (Eq. 14).

When E_{CT} is very close to the deadline of the job under consideration (i.e. $\frac{E_{CT}}{D_j}$ approaches to 1), which indicates that no resources, on average, can execute the job much earlier to its deadline, the *LD* weight approaches to zero. This means that the importance of the load balancing at that point of time is lessened. At the same time, the weights of *CT* and MTBF are increased to give higher importance

to these criteria in order to increase the probability of allocating the job to a more reliable and faster resource. Decreasing the *LD* weight in this case means that our mechanism favors fast and reliable resources for urgent jobs (jobs with large $\frac{E_{CT}}{D_j}$) over the underutilized resources which may be slow and less reliable. On the contrarily, the decrease in $\frac{E_{CT}}{D_j}$ indicates that the resources, on average, can execute the job much earlier to its deadline (non-urgent job). In this case, there is no need to send the job to a fast and more reliable resource as long as it can be executed within its deadline on a resource with lower speed and less reliability. So, the proposed mechanism increases the *LD* weight in this case to give option to the underutilized resources to be selected; and thus avoiding overwhelming the prominent resources with non-urgent jobs (jobs with $E_{CT} \ll D_j$). This also saves the fast and reliable resources for other urgent jobs. It is worth noting here that a non-urgent job is less influenced if it is allocated to a faulty resource since it can be restarted and finished within its deadlines on a new resource.

3.4.4 Allocation procedure

After calculating the weights of the criteria, they will be sent to our TOPSIS-based allocation and load balancing algorithm along with the list of the alternative resources which can execute the job inside the cluster within its deadline. The TOPSIS-based algorithm will then use them to prioritize the resources as follows.

– *Step 1* Construct the decision matrix A using the values of the alternative resources (the candidate resources) with respect to the selected criteria, namely MTBF, *LD*, and the estimated *CT* of the job on each resource:

$$A = \begin{array}{c} \\ R_1 \\ R_2 \\ \vdots \\ R_m \end{array} \begin{array}{c} \begin{array}{ccc} CT & MTBF & LD \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} \end{bmatrix} \end{array}$$

where $R_1, \ldots, R_m$, are the alternative resources from which the proposed algorithm has to choose the suitable one for the job execution. The *CT*, *MTBF*, and *LD* are the criteria with which the performance of each alternative is measured, and a_{ij} is the rating of the alternative resource R_i with respect to the criterion l_j. For example, a_{11} is the estimated completion time of the job under consideration on R_1 and a_{m3} is the load of R_m.

– *Step 2* Calculate the normalized decision matrix X:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{m3} \end{bmatrix}$$

Each normalized value x_{ij} is calculated as:

$$x_{ij} = \frac{a_{ij}}{\sqrt{\sum_{i=1}^{m} a_{ij}^2}}, \; for \; i \in \{1, 2, \ldots, m\} \text{ and } j \in \{1, 2, 3\}. \tag{15}$$

where m is the number of the alternative resources.

– *Step 3* The weighted normalized decision matrix V is constructed as follows:

$$V = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ v_{m1} & v_{m2} & v_{m3} \end{bmatrix} = \begin{bmatrix} W_{CT}x_{11} & W_{MTBF}x_{12} & W_{LD}x_{13} \\ W_{CT}x_{21} & W_{MTBF}x_{22} & W_{LD}x_{23} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ W_{CT}x_{m1} & W_{MTBF}x_{m2} & W_{LD}x_{m3} \end{bmatrix}$$

where $v_{ij} = w_j \, x_{ij}$, w_j is the weight of the criterion l_j, $i \in \{1, 2, \ldots, \mathrm{m}\}$, and $\mathrm{j} \in \{1, 2, 3\}$.

– *Step 4* Find the positive-ideal solution (R^+) and the negative-ideal solution (R^-):

$$\begin{aligned} R^+ &= \{(v_1^+, v_2^+, v_3^+)\} \\ &= \left\{ \left(\max_i v_{ij} | l_j \in C_B \right), \left(\min_i v_{ij} | l_j \in C_c \right) \right\} \end{aligned} \tag{16}$$

$$\begin{aligned} R^- &= \{(v_1^-, v_2^-, v_3^-)\} \\ &= \left\{ \left(\min_i v_{ij} | l_j \in C_B \right), \left(\max_i v_{ij} | l_j \in C_c \right) \right\} \end{aligned} \tag{17}$$

where C_B denotes the set of the benefit criteria (*MTBF* is the only benefit criterion used in the proposed algorithm) and C_c is associated with the cost criteria (i.e. the *CT* and *LD* criteria).

– *Step 5* Calculate the Euclidean distance measures for each candidate resource, which gauge the separation distances between the resource and each of the positive ideal solution and the negative ideal solution using Eqs. 18 and 19 respectively:

$$S_i^+ = \sqrt{\sum_{j=1}^{3}(v_{ij} - v_j^+)^2} \quad (18)$$

$$S_i^- = \sqrt{\sum_{j=1}^{3}(v_{ij} - v_j^-)^2} \quad (19)$$

where $i \in \{1, 2, \ldots, m\}$.

– *Step 6* Calculate the relative closeness $RC^+{}_i$ to the positive ideal solution for each alternative R_i as follows:

$$RC_i^+ = \frac{S_i^-}{S_i^+ + S_i^-} \quad (20)$$

where $0 \leq RC_i^+ \leq 1$. $RC_i^+ = 1$ if $R_i = R^+$ and $RC_i^+ = 0$ if $R_i = R^-$.

– *Step 7* The alternative resources are ranked according to the descending order of $RC^+{}_i$ and the resource with the maximum value of the relative closeness is chosen for the job allocation.

3.5 Load redistribution policy

Even though a good job placement is drawn during the allocation stage, a load redistribution is sometimes necessary in computational grids to achieve better load balancing. In fact, the dynamicity and uneven job arrival rates of resources may lead to load imbalance. So, the load redistribution policy presented here attempts to minimize the difference in the loads of the neighboring resources. The presented redistribution policy is triggered by any master resource (say R_m) every time it detects the emptiness of its global queue GQ. At that time, R_m first calculates the average load of its children $ALD(m)$ (Eq. 3) and the average load of the community rooted by itself $CoMALD(m)$ (Eq. 4). Then, it creates a list of overloaded resources which includes each child resource whose its load (LD) is greater than $ALD(m)$ and $CoMALD(m)$. After that, the master R_m sends a request to each resource in this list which, upon receiving the request, will migrate the new arriving jobs immediately to the GQ of R_m. Those jobs will then be distributed by R_m according to the scheduling policy introduced in Sect. 3.2. Finally, when R_m receives any job from a higher level, it notifies every resource in the list to stop sending the new arriving jobs as long as they can be executed locally or inside the communities rooted by themselves.

4 Fault tolerance and reliability model

The unique characteristics of truly-open computational grids such as the extreme heterogeneity, dynamicity, and geographically distribution of their resources make the probability of failures much greater than the traditional parallel and distributed systems [10]. Resource failures occur frequently in highly dynamic computational grids and have an adverse effect on jobs executions, and they always violate timing deadlines and service level agreement (SLA). So, the fault tolerance is essential to meet the QoS requirements of submitted jobs [36]. In this paper, a failure is defined to be an event that causes the resource to transition from the uptime state (available state) to the downtime state (unavailable state) making it invisible to the users of the grid. The failures in the sense of this perspective include but are not limited to resource crashes, shutdowns/restarts (announced or sudden), and power outages. Network failures are out of the scope of this paper. Fault tolerance can be categorized into reactive strategy and proactive strategy. In the reactive strategy,the failure is handled after it has occurred. Checkpointing is a typical example of this strategy in which a snapshot of the job's state during the normal execution is periodically saved to a stable storage. The saved snapshot will then be used upon a failure occurrence to restart the execution of the job from its last consistent state instead of restarting it from scratch. In contrast, the proactive fault tolerance strategy takes actions before dispatching jobs to resources [36]. For example, the resource reliability is considered during the allocation and load balancing decision making before actually dispatching the job to the resource. The proactive strategy will be followed in this paper.

Resource reliability, in this context, is quantified as mean time between failures (MTBF). MTBF is a common used measure in reliability engineering to describe the reliability of repairable systems. MTBF of each resource can be estimated using a statistical and historical analysis of the events stored in the data logs of each resource. MTBF can be approximated directly from the failure rate of the resource as the total time during the period of investigation divided by the number of failures encountered. However, this is a rough estimation and is not often ideal because it does not consider the time lost during failures. When the time of failures cannot be neglected, MTBF is best estimated as the accumulative time of uptime intervals (available intervals) divided by the total number of uptime intervals. Several probability distribution models such as the normal distribution, weibull distribution, and log normal distribution are often used to analyze the failure data and then predict the resource reliability. It is pointed out by [6, 9, 17–19, 35, 44] that the uptime/availability

intervals, from which MTBF is often calculated, are best modeled by the weibull distribution on modern distributed systems. On the other hand, the unavailability/down durations (from which the mean time to repair (MTTR) is calculated) are modeled by different probability distributions. Iosup [17] reported that the log normal distributions is a best choice for modeling the unavailability durations while Bouguerra et al. [6] used the hyperexponential distribution to model the unavailability durations.

5 Performance evaluation

We have developed a java-based simulator for assessing the performance of the proposed work under different system parameters in a highly dynamic simulated computational grid. All simulations were conducted on a Toshiba-Satellite computer with intel (R) core (TM) i3 processor M 350@2.27 GHz, 4 GB of RAM, and running windows 7 Home Premium 64-bit version. The proposed algorithm (labeled as OurLB) is compared with the following algorithms:

- Minimum completion time (MCT) [7]: This method selects jobs in an arbitrary order and then assigns each job to the resource that gives minimum expected completion time. MCT considers the resource and job heterogeneity. However, it does not consider the reliability and the idleness/underutilization states of resources, and it always selects the same prominent resources for arriving jobs. So, there is a high probability that some powerful resources, which may be unreliable, are being overwhelmed with many jobs while the other resources remain idles or underutilized. The failures of such overloaded resources often lead to a large loss of computation work, and also cause a high communication overhead. This is because that many jobs are required to be transferred from the failed resources to new ones. We selected this algorithm because it represents a typical algorithm for grid scheduling and allocation methods, and it is often used as a benchmark for grid scheduling evaluation. We also used it to show the performance of considering only the computational capacities of resources while ignoring their reliability and idleness.
- User demand aware grid scheduling model with hierarchical load balancing algorithm (labeled as UDSL) [40]: We selected this algorithm because it represents a typical class of hierarchical approaches, and bears similarity to our work. This algorithm focuses on both satisfying the user deadline and achieving better load balancing as mentioned in Sect. 2. It also considers the scalability and the heterogeneity of both resources and jobs. To perform load balancing in UDSL, resources/machines/PEs are categorized into overloaded, underloaded, and normally loaded lists upon arrival of a new job. Then, the arriving job will be allocated to a node from the underloaded list which satisfies its QoS requirements. However, UDSL does not consider resource reliability when distributing jobs.

5.1 Performance evaluation metrics

For evaluating the performance of the proposed work and comparing it with the above approaches, we used the following performance evaluation metrics:

- Average completion time (ACT): This metric evaluates the ability of the proposed method in minimizing the completion time of jobs. For each job, the CT is the time period from the point at which the job is submitted by its user to the time when the job is completed. ACT is calculated as:

$$ACT_t = \frac{\sum_{j=1}^{M} CT_j}{M} \tag{21}$$

  where t is the simulation time period, M is the number of jobs completed during this period, and CT_j is the completion time of job j.
- Throughput ratio (TR): It is the percentage of jobs that finished their executions within their time constraints (deadlines). This metric is selected as a standard performance criterion for reflecting the ability of the proposed method in satisfying the QoS of submitted jobs. It can be calculated as:

$$TR_t = \frac{S_j}{T_j} \times 100 \tag{22}$$

  where S_i is the number of jobs which are successfully completed within their deadlines and T_j is the total number of submitted jobs.
- Load balancing level (LBL): This metric is selected to reflect the ability of the proposed method in maximizing resource utilization. It can be derived as follows:

$$Let\, X_i = U_i \times \frac{MaxMTBF}{MTBF_i} \tag{23}$$

  where $MTBF_i$ is the reliability of R_i, $MaxMTBF$ is the maximum $MTBF$ of resources, N is the total number of resources in the system, and U_i is the utilization of R_i which is estimated as follows:

$$U_i = \frac{busy\ time}{busy\ time + idle\ time} \tag{24}$$

  where *busy time* is the time during which the resource is kept busy doing useful work.

Table 3 Jobs characteristics, grid characteristics, and simulation parameters

Parameters	Value
Number of resources N	3000
Job mean inter-arrival time of resources	Exponentially distributed in the range {0.05, 20}
Processing capacity	Uniformly distributed between 50 and 440 MIPS
Job length	Uniform distributed between 1000 and 10,000 MB (mega instructions)
Job input size	10–140 MB (uniformly distributed)
Job output size	25–300 MB (uniformly distributed)
Number of submitted jobs by each resource	Offset + $2^{rid \bmod 7}$, where *rid* is the resource *id*
The deadline of job j	It is calculated by this formula: $ar_j + \frac{jobsize_j}{mps} + y$, where ar_j is the arrival time of job j, mps is the mean processing capacity of the resources used, and y is the deadline dispersion parameter which is uniformly distributed between {300, 10,000}
Transfer delay ($TD(i, j)$)	Uniformly distributed in the range {0.005 to 0.06 s}
Bandwidth (baud rate)	Uniformly distributed in the range {1 to 400 MB}
Resource availability	Modeled by the weibull distribution with scale and shape parameters has been adjusted to simulate different MTBF for each resource. Specifically, the scale parameter is randomly sampled between {1–18}, when the shape parameters is uniformly distributed between {0.31–0.85} (the unit is h)
Resource unavailability	Modeled by the hyperexponential distribution with $\mu_i \in \{0.031, 11.566, 1.322\}$ and $p_i \in \{0.398, 0.305, 0.298\}$

$$LBL_t = \left(1 - \frac{SD_X}{AX}\right) \times 100 \tag{25}$$

where AX and SD_X are the average and the standard deviation over the values of X of resources, respectively. SD_X and AX can be calculated according to Eqs. 26 and 27 respectively.

$$SD_X = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - AX)^2} \tag{26}$$

$$AX = \frac{\sum_{i=1}^{N} X_i}{N} \tag{27}$$

5.2 Simulation model

Our developed simulator took into account the issues of scalability, heterogeneity, dynamicity of computing resources, and significant communication overhead. We used the same simulated grid platform for all evaluated approaches. The computational capacities of the simulated grid resources are uniformly distributed in the range [50 to 440 MIPS]. The baud rate (bandwidth) and the link transfer delay between resources are randomly sampled in the range [1 to 400 MB] and the range [0.005 to 0.06 s] respectively. To best simulate a highly dynamic grid system, the availability and unavailability durations of grid resources are modeled using the weibull and hyperexponential distribution models respectively. The shape and scale parameters of the weibull model were adjusted to simulate different MTBF for each resource. Specifically, the shape parameter β is uniformly distributed in the range [0.31–0.85] and the scale parameter η is randomly sampled between [1–18] (the measurement unit is hour). Whereas, the rate parameter μ_i and the probability parameter p_i of the hyperexponential distribution, which are used to model the unavailability durations, are used as suggested in [6], i.e. $\mu_i \in \{0.031, 11.566, 1.322\}$ and $p_i \in \{0.398, 0.305, 0.298\}$. We also generated a different job arrival rate λ_i for every grid resource R_i using a poisson process. The simulation parameters and their corresponding values are summarized in Table 3.

5.3 Simulation results

In this subsection, our simulation results are presented and the performance of the proposed approach is compared with MCT and UDSL which have been discussed above. All experiments were conducted under a system scale of 3000 resources which is large enough to represent the scale of a typical computational grid.

5.3.1 Effect of the maximum size of cluster *k*

In the first experiment, we studied the effect of increasing the maximum number of resources (k) per cluster on the performance of the evaluated approaches. We increased

k from 10 to 50 by adding ten resources at each step and then investigated the effect on *ACT*, *TR*, and *LBL*.

Figure 4 reveals that *ACT* decreases in all different approaches as k increases. This is because the number of jobs that are executed locally inside their clusters increases as k increases. In other words, the number of jobs that traverse longer between levels searching for suitable resources is decreased. This certainly minimizes the communication overhead and thus improving *ACT*. However, it is noted that our approach gives minimum *ACT* compared to MCT and UDSL under all values of k. The improvement obtained by our approach is attributed to the effective distribution of workload between resources which can be achieved by considering resource reliability, resource capacity, and the current load balance state of system. In fact, taking these factors into consideration minimizes jobs' execution losses, which may be caused by resources failures, and reduces the jobs' rescheduling time; thus improving *ACT* noticeably. From Fig. 4, it can also be seen that the performance improvements gained by MCT and UDSL decrease as k increases in comparison with our approach. This is because of the increase in the number of rescheduled jobs in these two approaches. The maximum improvements obtained by our approach over MCT and UDSL are 8.7% and 10.58%, respectively.

Figure 5 reports throughput ratio *TR* for the three evaluated approaches under different values of k. The results show that our approach extremely improves the performance and gives better *TR* for all k compared to MCT and UDSL. The maximum improvement ratios obtained by our approach are 16.44% and 19.4% over UDSL and MCT respectively. The improvement obtained by our approach comes from the fact that the QoS of jobs, resource capacity, and reliability are considered when making the allocation and load balancing decisions. So, in the proposed method, the jobs with early deadlines are likely to be allocated to reliable and fast resources unlike the other approaches in which those jobs may be allocated to less reliable resources which often experience frequent failures. In UDSL and MCT, those jobs often require to be frequently rescheduled (maybe from scratch) and also need much time to be transferred to new resources, so they may fail to complete their executions within their deadlines. From Fig. 5, it is observed that as k increases the performance improvement ratio obtained by our approach increases compared to MCT and UDSL.

Figure 6 shows the result of *LBL* for the three evaluated approaches. It is obvious that our approach gives better results for all values of k especially when comparing with MCT which gives worse *LBL*. This is because MCT often allocates the jobs to the prominent resources while ignoring the idleness/underutilization state of the other resources. UDSL can balance the workload, however, its efficiency is lower than our approach and may result in overutilization or underutilization of some resources because it does not continuously monitor the workload of resources and then redistributes jobs between the overloaded and underloaded ones if necessary.

Table 4 shows the improvement ratios obtained by our load balancing approach compared to the other approaches for the three used metrics.

5.3.2 Effect with jobs deadlines span

In this experiment, we focused on analyzing the performance when varying the distribution range of jobs'

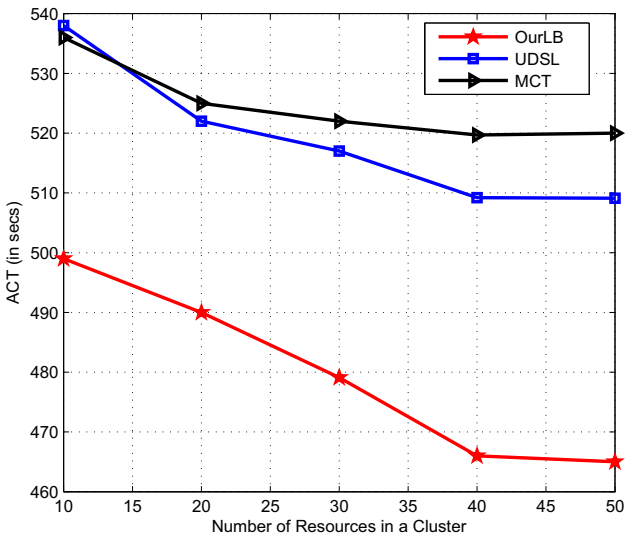

Fig. 4 *ACT* with increasing the maximum number of resources (k) per cluster for MCT, UDSL, and OurLB

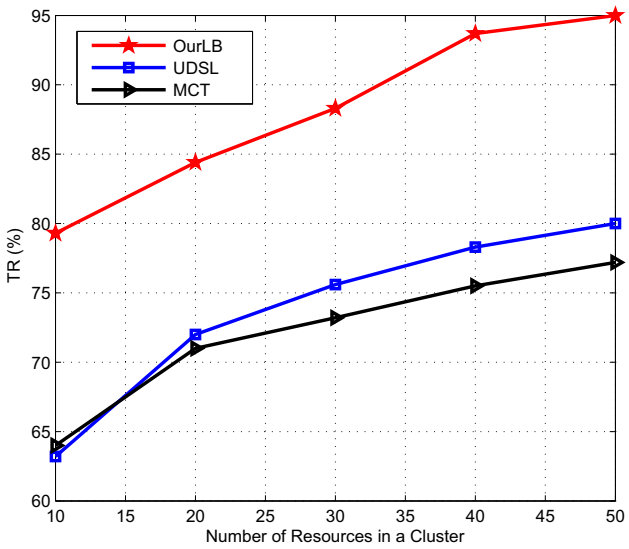

Fig. 5 *TR* with increasing the maximum number of resources (k) per cluster for MCT, UDSL, and OurLB

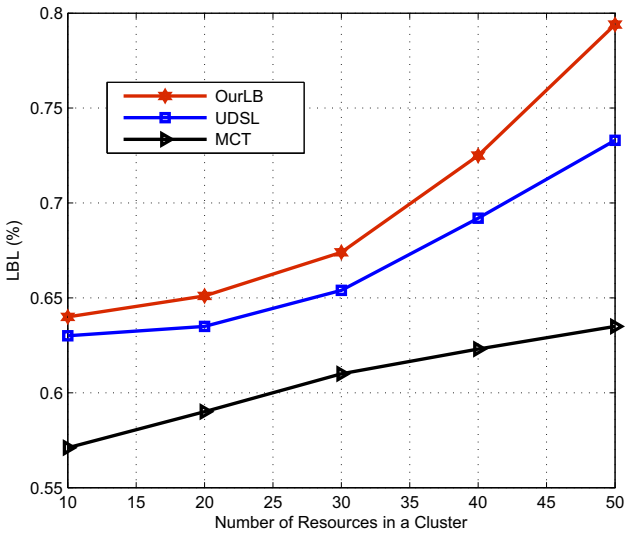

Fig. 6 *LBL* with increasing the maximum number of resources (k) per cluster for MCT, UDSL, and OurLB

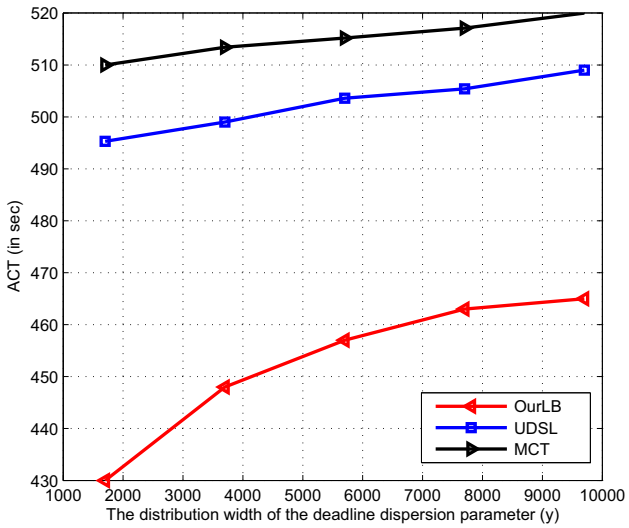

Fig. 7 *ACT* with increasing the uniform distribution width of the deadline dispersion parameter y for MCT, UDSL, and OurLB

Table 4 Improvement ratios obtained by our approach in comparison with UDSL and MCT when varying the cluster size

Metric	OurLB–UDSL (%)	OurLB–MCT (%)
ACT	8.7	10.58
TR	16.44	19.4
LBL	7.68	20.1

deadlines. For this purpose, the deadline dispersion parameter y in the formula $ar_j+\frac{jobsize_j}{mps}+y$ is distributed on different uniform distribution intervals which are: [300, 2000], [300, 4000], [300, 6000], [300, 8000], and [300, 10,000]. More specifically, the upper bound was varied from 2000 to 10,000 by adding 2000 at each step (which means that the distribution width of the deadline dispersion parameter y varies, from 1700 to 9700). The maximum number of resources per cluster k was fixed at 40 in this experiment when the other simulation parameters were kept unchanged as in the former experiment.

Figure 7 illustrates the effect of varying the distribution width of the deadline parameter y on *ACT* for the three evaluated approaches. It is obvious that our approach achieves minimum *ACT* compared to UDSL and MCT under all examined distribution widths. The maximum improvement ratios obtained by our approach are 13.2% and 15.7% over MCT and UDSL respectively. This is because the proposed weighting mechanism in our approach considers the deadlines of jobs and then adaptively adjusts the importance of the considered criteria during allocation and redistribution processes accordingly. For narrow widths, which means that the majority of jobs have early deadlines, the importance of load balancing is often reduced whereas a higher importance is given to the resources' reliability and capacity. This ensures that the jobs with early deadlines are allocated to more reliable and powerful resources. For wider distribution widths, which means that the number of non-urgent jobs (jobs with late deadlines) increases, our approach increases the importance of load balancing to ensure that jobs are allocated to underutilized resources even those with low reliability and capacity. However, this may lead to some increase in *ACT*. From Fig. 7, we can also notice that MCT is not significantly affected by varying the distribution range of the deadline dispersion parameter. It can also be seen that MCT gives the worst results. This is because some fast but unreliable resources may be overwhelmed with jobs since MCT always ignores the load balance state and resource reliability. In the highly dynamic grid, where the failure is the rule, not the exception, the presence of failures in such overloaded resources often leads to a severe effect on *ACT*. This is due to the large executions losses and the additional time spent in rescheduling the affected jobs.

Figure 8 shows the effect of increasing the distribution width of the deadline dispersion parameter y on *TR* in the three evaluated approaches. It can be noted that all approaches improve *TR* under all distribution widths. However, it is clear that our approach gives better results compared to UDSL and MCT. In highly dynamic grids, there is a high probability that UDSL and MCT fail to complete the urgent jobs, which have early deadlines, within their time constraints. This is because these approaches do not consider the QoS of jobs and resource reliability when allocating and distributing jobs to resources. In fact, allocating those jobs to unreliable resources

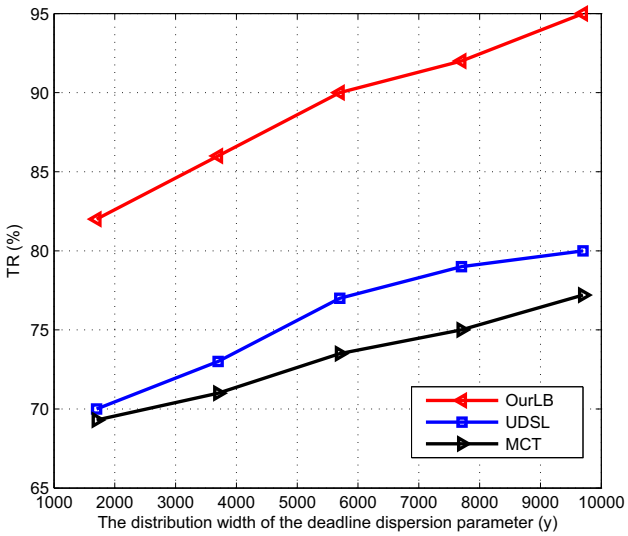

Fig. 8 *TR* with increasing the uniform distribution width of the deadline dispersion parameter *y* for MCT, UDSL, and OurLB

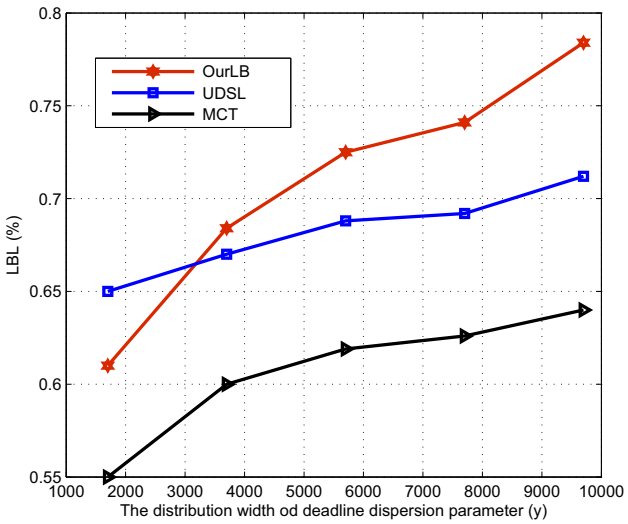

Fig. 9 *LBL* with increasing the uniform distribution width of the deadline dispersion parameter *y* for MCT, UDSL, and OurLB

may lead to frequent failures which cause more execution losses and thus extra delay time is added to *ACT*. This is supported by the worst results obtained by UDSL and MCT at narrow widths where the majority of jobs have early deadlines. On the other hand, the jobs in our approach are distributed to resources according to their reliabilities and capacities, so the reliable resources execute more jobs than do the unreliable resources. Also, the jobs with early deadlines are more likely to be allocated to more reliable and powerful resources. As a result, the number of the jobs which are completed within their defined deadlines are increased. The maximum improvements obtained by our approach are 15.8% and 18.74% over the MCT and UDSL respectively.

In Fig. 9, the effect of varying the distribution width of the deadline parameter *y* on *LBL* is shown. It is noted that our approach gives better results compared to UDSL and MCT except for USDL approach at the distribution width of less than 3200. That is because the importance of load balancing in our approach is adjusted adaptively according to the system state and the jobs' characteristics. For example, at the distribution width of less than 3200, where almost all of the jobs have early deadlines, our approach reduces the importance of load balancing criterion to ensure that the jobs are sent to the faster and more reliable resources so that they can finish their executions within their time constraints. On the other hand, for wider distribution widths, where the number of jobs with late deadlines increases, our approach frequently increases the importance of load balancing specifically when allocating non-urgent jobs. Under this circumstance, our approach gives option to less reliable and slower resources to participate in jobs executions. From Fig. 9, it is obvious that MCT gives the worst results. It can also be seen that the *LBL* obtained by MCT was not significantly affected by increasing the distribution width of the deadline dispersion parameter. This is because MCT does not consider the deadlines of jobs when distributing them to resources.

Table 5 summarizes the performance improvements gained by our load balancing approach in comparison with other approaches for the three used metrics.

With respect to each performance metric and in order to guarantee statistical correctness, each measurement was carried out ten times each with different random seeds, and the average of these runs was used to plot the evaluated performance of the three approaches. The standard deviation over the values of these runs, which are carried at each measurement, are shown in Tables 6 and 7 for the experiments when varying the cluster size and the distribution with of the deadline dispersion parameter respectively.

5.3.3 The evaluation of performance gained by taking the load balancing into consideration at the allocation process

This experiment was carried out to investigate how the consideration of the system load balancing state at the

Table 5 Improvement ratios obtained by our approach in comparison with UDSL and MCT approaches when varying the range of the deadline dispersion parameter

Metric	OurLB–UDSL (%)	OurLB–MCT (%)
ACT	13.2	15.7
TR	15.8	18.74
LBL	9.18	18.4

Table 6 The standard deviation of simulation runs values at each measurement when varying the cluster size (k)

k	10			20			30			40			50		
	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL
OurLB	4.3	5.2	4.3	4.3	3.1	4.8	2.9	3.86	5.1	3.9	3.86	4.2	4.07	3.96	3.6
UDSL	5.2	6.5	5.9	6.1	5.3	6.1	3.88	5.94	3.6	5.2	3.98	4.06	5.02	4.27	3.45
MCT	4.04	5.06	7.0	4.91	6.1	6.5	4.04	3.1	6.6	5.4	5.7	5.84	6.0	4.95	6.02

k is the maximum number of resources per cluster

Table 7 The standard deviation of simulation runs values when varying the distribution with of the deadline parameter (*y*)

U of *y*	1700			3700			5700			7700			9700		
	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL	ACT	TR	LBL
OurLB	2.5	0.9	4.6	1.9	4.6	4.7	3.6	3.6	3.9	3.92	2.6	3.95	4.01	3.51	2.91
UDSL	6.3	5.2	4.5	4.0	4.9	4.02	4.06	4.3	4.0	2.2	2.98	3.73	4.19	3.82	3.92
MCT	5.3	5.7	6.3	4.82	5.7	4.99	6.8	4.1	4.4	2.9	3.3	5.4	7.5	4.14	5.2

U of *y* is the uniform distribution width of *y*

allocation process affects the performance through reducing the communication overhead during the redistribution process. The performance improvement was also analyzed here in terms of *ACT*, *TR*, and *LBL* when varying the maximum number of resources (*k*) from 10 to 50 by adding ten resources at each step. This experiment was conducted under the same simulation parameters which are shown in Table 3. For a fair comparison, two adjustments of our approach were evaluated and compared against each other. The first adjustment (denoted as OurLB_CT) represents the case when the load balancing is not taken into consideration during the allocation process when the other adjustment (denoted as OurLB_CT + LD) represents the case when the load balancing is taken into account. Furthermore, the two adjustments use the same load redistribution policy which was explained in Sect. 3.5, and the reliability criterion is neutralized in the both adjustments through setting its weight to a very small value. The two adjustments are explained as follows:

- OurLB_CT: In this adjustment, the weights of the *LD* and MTBF criteria were statically adjusted to very small values which are close to zero (0.001 for each) while the weight of the *CT* criterion was set to 0.998. This adjustment is to approximately simulate the case when the resources are ranked at allocation or at redistribution process only based on the *CT* criterion. It is worth noting here that the load balancing state and resource reliability are not completely ignored since TOPSIS technique must depend on several criteria to make a decision; however the small weights values of their corresponding criteria (i.e. *LB* and MTBF) marginalize their effects.
- OurLB_CT + LD: This adjustment is to approximately simulate the case when the load balancing state (the *LD* criterion) is considered at the allocation process alongside with the completion time (the *CT* criterion). In this adjustment, the weight of the MTBF criterion was also statically adjusted to a small value (0.001), which means that the resource reliability is also marginalized; whereas the weights of the *CT* and *LD* criteria are adaptively determined according to the following equations respectively:

$$W_{CT} = \frac{SD_{CT}}{NSD} + \frac{E_{CT}}{D_j} \times \frac{SD_{LD}}{NSD} - 0.001 \tag{28}$$

$$W_{LD} = \frac{SD_{LD}}{NSD} - \frac{E_{CT}}{D_j} \times \frac{SD_{LD}}{NSD} \tag{29}$$

where $NSD = SD_{CT} + SD_{LD}$

Figure 10 shows the *ACT* in the two evaluated adjustments when varying the maximum number of resources (*k*) per cluster. Clearly, OurLB_CT + LD gives better results compared to OurLB_CT for all values of *k*. It has an average improvement factor of 3.2% over OurLB_CT. This is because the consideration of load balancing at the allocation process avoids overloading resources with jobs, and thus reduce the number of jobs which have to be redistributed at the redistribution process to get the system balanced. Also, the number of jobs which should be retransferred to new resources in presence of failures will be reduced (unlike OurLB_CT in which the powerful but

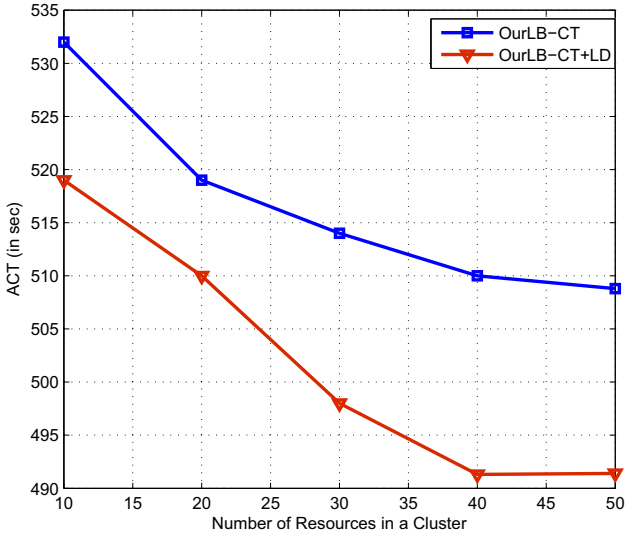

Fig. 10 *ACT* with increasing the maximum number of resources (k) per cluster for OurLB_CT and OurLB_CT + LD

faulty resources may be overloaded with many jobs). In fact, the less the number of jobs required to be retransferred, the less the communication time is needed, and thus the less *ACT*.

Figure 11 shows the throughput ratio *TR* for the two evaluated adjustments under different values of *k*. From the figure, it can be concluded that the *TR* that results from applying OurLB_CT + LD is higher than the *TR* results from applying OurLB_CT across all values of *k*. OurLB_CT + LD has an average improvement factor of 7.3% over OurLB_CT. This improvement came form the fact that the number of jobs which have to be redistributed in OurLB_CT + LD at the distribution process or in case of failures are less than those in OurLB_CT as mentioned above. This means that no additional transfer time has to be added to the *CT* of most jobs, so they finish their executions within their deadlines as expected.

Finally, the *LBL* for the two adjustments OurLB_CT and OurLB_CT + LD are depicted in Fig. 12. From the figure, it can be observed that OurLB_CT + LD and OurLB_CT give comparable *LBL*. This is because they both used the same redistribution policy which redistributed the jobs among resources to get the system balanced.

Table 8 shows the performance improvement ratios gained by OurLB_CT + LD compared to OurLB_CT for the three used metrics.

6 Conclusions and future work

In this paper, we have presented a reliable, adaptive, multi-criteria TOPSIS-based, and hierarchical load balancing method for a truly-open computational grid which is composed of dedicated machines alongside individual and autonomous computing resources which characterized by their intermittent availability. The proposed method uses the estimated completion time, resource reliability, and resource load as criteria upon which the allocation and distribution decisions are made. This enables fair distribution of workload among resources based on their reliability and capacity in a way that ensure that all resources participate in jobs' executions. In this method, a novel weighting mechanism was also proposed in which the weights of the used criteria are determined adaptively according to jobs' characteristics and the system's current state.

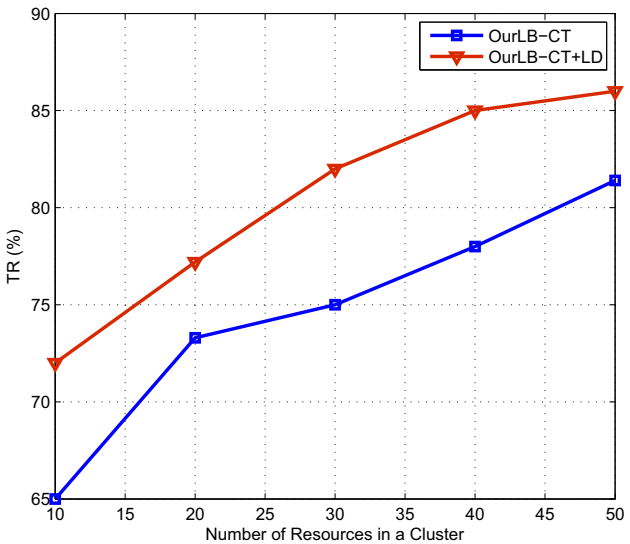

Fig. 11 *TR* with increasing the maximum number of resources (k) per cluster for OurLB_CT and OurLB_CT + LD

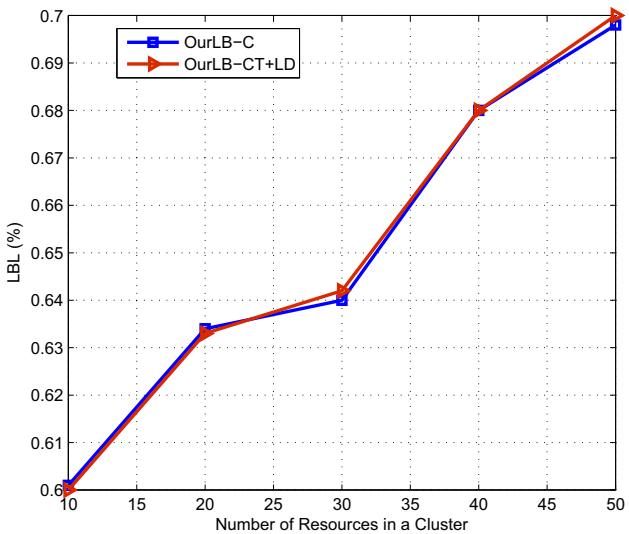

Fig. 12 *LBL* with increasing the maximum number of resources (k) per cluster for OurLB_CT and OurLB_CT + LD

Table 8 Improvement ratios obtained by OurLB_CT + LD in comparison with OurLB_CT when varying cluster size (k)

Metric	OurLB_CT (%)
ACT	3.2
TR	7.3
LBL	0.15

The simulation results proved that the proposed method outperforms other approaches in the range of the examined parameters' values. The results proved that the proposed method speeds up completion time, improves system throughput and resource utilization. In future work, we plan to propose a sophisticated prediction based model for reliability measurement of grid resources. We also consider jobs which have inter-process communication.

References

1. Abdullah, A.M., Ali, H.A., Haikal, A.Y.: Reliable and efficient hierarchical organization model for computational grid. J. Parallel Distrib. Comput. **104**, 191–205 (2017)
2. Balasangameshwara, J., Raju, N.: Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost. IEEE Trans. Comput. **62**(5), 990–1003 (2013)
3. Banerjee, S., Kommareddy, C., Bhattacharjee, B.: Scalable peer finding on the internet. IEEE Glob. Telecommun. Conf. **3**, 2205–2209 (2002)
4. Bansal, S., Hota, C.: Distributed scheduling on utility grids. Romanian J. Inf. **16**(4), 373–392 (2013)
5. Behzadian, M., Khanmohammadi Otaghsara, S., Yazdani, M., Ignatius, J.: A State-of the-art survey of TOPSIS applications. Expert Syst. Appl. **39**(17), 13051–13069 (2012)
6. Bouguerra, M.S., Kondo, D., Martin, M.S., Trystram, D.: On the scheduling of checkpoints in desktop grids. In: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Newport Beach, CA, pp. 305–313 (2011)
7. Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems (2001)
8. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), pp. 181–194. IEEE Press (2001)
9. Dagnew, S.A.: Optimization of periodic maintenance using condition monitoring techniques and operational data. PhD thesis, University of Stavanger, Norway (2012)
10. El-Sayed, G.A., Abdullah, A.M.: Mailbox-based non blocking minimum-process coordinated checkpointing with message logging for hierarchical computational grid (MNMCCP). In: 2012 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA), pp. 86–90 (2012)
11. El-Zoghdy, S.F.: A Two-level load balancing policy for grid computing. In: International Conference on Multimedia Computing and Systems (ICMCS), pp. 617– 622 (2012)
12. El-Zoghdy, S.F.: An intelligent AntNet-based algorithm for load balancing in grid computing. Int. J. Comput. Technol. **11**(9), 2975–2986 (2013)
13. Golmohammadi, R., Shahhoseini, H.S.: Load balancing in local computational grids within resource allocation process. Res. J. Appl. Sci. Eng. Technol. **4**(21), 4546–4551 (2012)
14. Goswami, S., Das, A.: Resource prioritization technique in computational grid environment. In: Proceedings of the Second International Conference on Computer and Communication Technologies, Advances in Intelligent Systems and Computing, pp. 765–772 (2016)
15. Helmy, T., Al-jamimi, H., Ahmed, B., Loqman, H.: Fuzzy logic based scheme for load balancing in grid services. J. Softw. Eng. Appl. **5**, 149–156 (2012)
16. Hwang, C.L., Yoon, K.: Multiple Attribute Decision Making: Methods and Application, vol. 186. Springer, New York (1981)
17. Iosup, A., Jan, M., Sonmez, O., Epema, D.H.J.: On the dynamic resource availability in grids. In: 8th IEEE/ACM International Conference on Grid Computing, Austin, TX, pp. 26–33 (2007)
18. Javadi, B., Kondo, D., Vincent, J.M., Anderson, D.P.: Discovering statistical models of availability in large distributed systems: an empirical study of SETI@home. Measurement **22**(11), 1896–1903 (2011)
19. Javadi, B., Kondo, D., Iosup, A., Epema, D.: The failure trace archive: enabling the comparison of failure measurements and models of distributed systems. J. Parallel Distrib. Comput. **73**(8), 1208–1223 (2013)
20. Kumar, D., Chitaranjan, P.: An improved approach for load balancing among heterogeneous resources in computational grids. Eng. Comput. **31**(4), 825–839 (2014)
21. Li, K.: Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments. J. Syst. Arch. **54**, 111–123 (2008)
22. Li, T., Ren, Y., Yu, D., Jin, S.: Resources-conscious asynchronous high-speed data transfer in multicore systems: design, optimizations, and evaluation. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1097–1106 (2015)
23. Lu, K.: Decentralized load balancing in heterogeneous computational grids. PhD thesis, University of Sydney, Australia (2007)
24. Lu, K., Subrata, R., Zomaya, A.Y.: An efficient load balancing algorithm for heterogeneous grid systems considering desirability of grid sites. In: The 25th IEEE International Performance, Computing, and Communications Conference (IPCCC), pp. 311–320 (2006)
25. Lu, K., Subrata, R., Zomaya, A.Y.: Towards decentralized load balancing in a computational grid environment. In: GPC'06 Proceedings of the First International Conference on Advances in Grid and Pervasive Computing, vol. 3947, pp. 466–477 (2006)
26. Lu, K., Subrata, R., Zomaya, A.Y.: On the performance-driven load distribution for heterogeneous computational grids. J. Comput. Syst. Sci. **73**, 1191–1206 (2007)
27. Mohanty, D.R., Mishra, S.K.: A data-driven approach for option pricing algorithm. In: Proceedings of the Second International Conference on Computer and Communication Technologies, Advances in Intelligent Systems and Computing, vol. 380, pp. 163–170 (2016)
28. Onder, E., Dag, S.: Combining analytical hierarchy process and TOPSIS approaches for supplier selection in a cable company. J. Bus. Econ. Finance **2**(2), 56–74 (2013)
29. Patel, D.K., Tripathy, D., Tripathy, C.: An improved load-balancing mechanism based on deadline failure recovery on gridsim. Eng. Comput. **32**(2), 173–188 (2016)
30. Patel, D.K., Tripathy, D., Tripathy, C.: Survey of load balancing techniques for grid. J. Netw. Comput. Appl. **65**(C), 103–119 (2016)
31. Pérez-Miguel, C., Mendiburu, A., Miguel-Alonso, J.: Competition-based failure-aware scheduling for high-throughput computing systems on peer-to-peer networks. Clust. Comput. **18**(3), 1229–1249 (2015)

32. Righi, R.D.R.: MigBSP: a new approach for processes rescheduling management on bulk synchronous parallel applications. PhD thesis, Universidade Federal Do Rio Grande Do Sul (2009)
33. Rood, B.: Grid resource availability prediction-based scheduling and task replication. PhD thesis, State University of New York at Binghamton, Binghamton (2011)
34. Santiago, A.J.S., Yuste, A.J., Expósito, J.E.M., Galán, S.G., Prado, R.P.D.: A multi-criteria meta-fuzzy-scheduler for independent tasks in grid computing. Comput. Inform. **30**, 1201–1223 (2011)
35. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. IEEE Trans. Dependable Secure Comput. **7**(4), 337–350 (2010)
36. Singh, S., Bawa, R.K.: Proactive fault tolerance algorithm for job scheduling in computational grid. Int. J. Grid Distrib. Comput. **9**(3), 135–144 (2016)
37. Snchez, J.M.: Global behavior modeling: a new approach to grid autonomic management. PhD thesis, Boston, MA (2010)
38. Soundarabai, P.B., A, S.R., Sahai, R.K., J, T., Venugopal, K.R., Patnaik, L.M.: Comparative study on load balancing techniques in distributed systems. Int. J. Inf. Technol. Knowl. Manag. **6**(1), 53–60 (2012)
39. Subrata, R., Zomaya, A.Y., Landfeldt, B.: Artificial life techniques for load balancing in computational grids. Comput. Syst. Sci. **23**(8), 1176–1190 (2007)
40. Suresh, P., Balasubramanie, P.: User demand aware grid scheduling model with hierarchical load balancing. Math. Probl. Eng. **2013**, 8 (2013)
41. Wolski, R., Spring, T., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**(5–6), 757–768 (1999)
42. Yagoubi, B., Slimani, Y.: Dynamic load balancing strategy for grid computing. World Acad. Sci. Eng. Technol. **19**, 90–95 (2006)
43. Yagoubi, B., Slimani, Y.: Task load balancing strategy for grid computing. J. Comput. Sci. **3**(3), 186–194 (2007)
44. Zhang, Y., Mandal, A., Koelbel, C., Cooper, K., Hill, C.: Combined fault tolerance and scheduling techniques for workflow applications on computational grids. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, pp. 244–251 (2009)
45. Zhu, Y., Ni, L.M.: A survey on grid scheduling systems (2013)

Aref M. Abdullah is a Ph.D. student at Mansoura University, Egypt. He received a B.Sc. in Computer Engineering from University of Technology, Iraq, in 2001. He received an M.Sc. in Electrical Engineering from Assiut University, Egypt, in 2013. His areas of interest include networking administration, resource management, job scheduling and load balancing in traditional distributed systems, grid computing and in cloud computing.

Hesham A. Ali is a Professor in Computer Eng. and Sys. and an associate Professor in Info. Sys. and computer Eng. He received a B.Sc. in Electronics Eng., and M.Sc. and Ph.D. in Computer Eng. and Control from the Fac. of Engineering, Mansoura Univ., in 1986,1991 and 1997, respectively. He is a founder member of the IEEE SMC Society Technical Committee on Enterprise Information Systems (EIS). He has many book chapters published by international press and about 150 published papers in international (conf. and journal). He has served as a reviewer for many high quality journals, including Journal of Engineering Mansoura University. His interests are in the areas of network security, mobile agent, network management, search engine, pattern recognition, distributed databases, and performance analysis.

Amira Y. Haikal is an associate Professor in Computers Eng. and Control Sys. Dept. at Faculty of Engineering, Mansoura University. She received a B.Sc. in Electronics Eng., and M.Sc. and Ph.D. in Computers Eng. and Control from the Fac. of Engineering, Mansoura Univ., in 1998, 2001 and 2007, respectively. Her interests are in the areas of machine learning, artificial intelligence, optimization, grid computing and cloud computing.