


# An improved load-balancing mechanism based on deadline failure recovery on GridSim

Deepak Kumar Patel<sup>1</sup>  · Devashree Tripathy<sup>2</sup> · Chitaranjan Tripathy<sup>1</sup>

Received: 25 March 2015 / Accepted: 18 June 2015 / Published online: 30 June 2015  
© Springer-Verlag London 2015

**Abstract** Grid computing has emerged a new field, distinguished from conventional distributed computing. It focuses on large-scale resource sharing, innovative applications and in some cases, high performance orientation. The Grid serves as a comprehensive and complete system for organizations by which the maximum utilization of resources is achieved. The load balancing is a process which involves the resource management and an effective load distribution among the resources. Therefore, it is considered to be very important in Grid systems. For a Grid, a dynamic, distributed load balancing scheme provides deadline control for tasks. Due to the condition of deadline failure, developing, deploying, and executing long running applications over the grid remains a challenge. So, deadline failure recovery is an essential factor for Grid computing. In this paper, we propose a dynamic distributed load-balancing technique called “Enhanced GridSim with Load balancing based on Deadline Failure Recovery” (EGDFR) for computational Grids with heterogeneous resources. The proposed algorithm EGDFR is an improved version of the existing EGDC in which we perform load balancing by providing a scheduling system which includes the mechanism of recovery from deadline

failure of the Gridlets. Extensive simulation experiments are conducted to quantify the performance of the proposed load-balancing strategy on the GridSim platform. Experiments have shown that the proposed system can considerably improve Grid performance in terms of total execution time, percentage gain in execution time, average response time, resubmitted time and throughput. The proposed load-balancing technique gives 7 % better performance than EGDC in case of constant number of resources, whereas in case of constant number of Gridlets, it gives 11 % better performance than EGDC.

**Keywords** Load balancing · GridSim · Gridlet · Response time

## 1 Introduction

A *Grid* is a computing and data management infrastructure that provides the electronic underpinning for a global society in business, government, research, science and entertainment [1]. A computational Grid constitutes the software and hardware infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [2]. The Grid integrates networking, communication, computation and information to provide a virtual platform for computation and data management in the same way that the Internet integrates resources to form a virtual platform for information [1]. The Grid can also be considered as a collection of distributed computing resources over a local or wide area network that appear to an end user as one large virtual computing system [3]. The speedy development in computing resources has enhanced the performance of computing systems with reduction in cost. The availability of low cost, high-speed networks,

---

✉ Deepak Kumar Patel  
patel.deepak42@gmail.com

Devashree Tripathy  
devashree.tripathy@gmail.com

Chitaranjan Tripathy  
crt.vssut@yahoo.com

<sup>1</sup> Department of Computer Science & Engineering,  
Veer Surendra Sai University of Technology, Burla,  
Sambalpur 768018, Odisha, India

<sup>2</sup> CSIR-Central Electronics Engineering Research Institute,  
Pilani 333031, Rajasthan, India

powerful computers coupled with the advances and the popularity of the Internet has led the computing environment to be mapped from the traditional distributed systems to the Grid environments [4].

A *computational Grid* enables the effective access to high-performance computing resources. It supports the sharing and coordinated use of resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal [5]. Grid infrastructure provides us with the ability to dynamically link together resources as an ensemble to support the execution of large-scale, resource-intensive and distributed applications [1]. With its multitude of heterogeneous resources, a proper scheduling and efficient load balancing across the Grid is required for improving the performance of the system [6]. The load-balancing mechanism attempts to improve the response time of the user's submitted applications by ensuring maximal utilization of available resources. The main goal of this type of algorithm is to prevent, if possible, the condition in which some processors are overloaded with a set of tasks while others are lightly loaded or even idle [7].

In general, the load-balancing algorithms are classified as *static* and *dynamic*. In a static algorithm, the information governing load-balancing decisions which include the characteristics of the jobs, the computing nodes and the communication networks are known in advance. The load-balancing decisions are made deterministically or probabilistically at compile time and remain constant during runtime. In contrast, the *dynamic* load-balancing algorithms attempt to use the runtime state information to make more informative load-balancing decisions. Here, the responsibility for making global decisions may lie with one centralized location, or be shared by multiple distributed locations. Undoubtedly, the static approach is easier to implement and has minimal runtime overhead. However, the advantage of dynamic load balancing over static is that the system need not be aware of the runtime behaviour of the application before execution. The adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm and/or the scheduling policy itself is changed based on the global state of the system. According to another classification, the load-scheduling algorithms could be classified as *centralized* or *distributed*. In the *centralized* approach, one node in the system acts as a scheduler and makes all the load-balancing decisions. The information is sent from the other nodes to the scheduler. In the *distributed* approach, all the nodes of the system remain involved in the load-balancing decisions. It therefore, becomes very costly for each node to obtain and maintain the dynamic state information of the whole system. Here, each node obtains and maintains only the partial information locally to make suboptimal decisions. However, the

distributed algorithms suffer from the problem of communication overheads incurred by frequent information exchange between processors. The centralized strategy on the other hand has the advantage of ease of implementation, but it suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck. Therefore, the centralized algorithms are found to be less reliable than the decentralized algorithms [8, 9].

Load balancing has been discussed in traditional distributed systems literature for more than three decades. Although a Grid belongs to the class of distributed systems, the load balancing algorithms which are usually run on homogeneous and dedicated resources in classical distributed systems, cannot work well in the Grid architectures. This is due to the unique characteristics of the Grid computing environment such as heterogeneity, autonomy, scalability, adaptability, dynamic behaviour, application diversity, resource non-dedication, resource selection and computation-data separation. Thus, it is a challenging problem to design an efficient and effective load-balancing scheme for Grid environments which can integrate all the above said factors [7].

In this paper, our basic aim is to develop a load-balancing model for Grids which can be adapted to the heterogeneous Grid computing environment. The method proposed by us is an improvement over the existing enhanced GridSim with deadline control (EGDC) [7]. Our proposed approach called "Enhanced GridSim with Load balancing based on Deadline Failure Recovery" (EGDFR) performs load balancing by providing a scheduling system which includes the mechanism of recovery from deadline failure of the Gridlets. The proposed load-balancing strategy (EGDFR) is simulated on the GridSim platform [10]. The proposed mechanism is shown to reduce the total execution time, average response time (ART), resubmitted time and give better results for throughput in comparison to [7, 11].

The rest of the paper is organized as follows. In the next Section, the related works are discussed. We hold background discussions on GridSim and some important existing load-balancing schemes on GridSim in Sect. 3. In Sect. 4, we present our Grid load-balancing scheme EGDFR. The Sect. 5 presents the simulation results and compares the proposed work with the existing load-balancing schemes LBEGS [11] and EGDC [7]. Finally, the concluding remarks are presented in Sect. 6.

## 2 Related works

In literature, the researchers have proposed several load-balancing strategies in Grid environments [6, 7, 12–27].

In [12], Anand et al. presented a decentralized dynamic load-balancing algorithm called Estimated Load

Information Scheduling Algorithm (ELISA) for general purpose distributed computing systems. The ELISA uses the estimated state information based upon the periodic exchange of exact state information between the neighbouring nodes to perform load scheduling. The primary objective of their algorithm is to cut down the communication and load transfer overheads by minimizing the frequency of status exchange and by restricting the load transfer and status exchange within the buddy set of a processor. It is shown that the resulting algorithm performs almost as well as a perfect information algorithm and is superior to other load-balancing schemes based on the random sharing. In [6], Shah et al. proposed two algorithms, the MELISA (Modified ELISA) and load balancing on arrival (LBA). Their algorithms differ in the way the load balancing is carried out and are shown to be efficient in minimizing the response time on large- and small-scale heterogeneous Grid environments. The MELISA is applicable to large-scale systems with the resource heterogeneity and network heterogeneity. The other algorithm, LBA is applicable to small-scale systems. It performs load balancing by estimating the expected finish time of a job on buddy processors on each job arrival. Both the said algorithms estimate system parameters such as the job arrival rate, CPU processing rate and load on the processor. They quantify the performance of their algorithms using several influencing parameters such as the job size, data transfer rate and the status exchange period.

The researchers in [13] used several well-known artificial life techniques to gauge their suitability for solving Grid load-balancing problems. The artificial life techniques can be used to solve a wide range of complex problems in recent times. The power of these techniques stems from their capability in searching large search spaces, which arise in many combinatorial optimization problems very efficiently. Due to their popularity and robustness, the genetic algorithm (GA) and tabu search (TS) are used to solve the Grid load-balancing problem. The results of the experiment showed that these two methods can be effectively used for Grid load balancing. The GA and TS showed similar performance results, and performed better than the Best-fit, Random, Min–min and Max–min algorithms. Bharadwaj et al. [14] proposed a new paradigm for load scheduling in distributed systems e.g. divisible load theory. The Divisible load theory is a methodology that involves the linear and continuous modelling of partitionable computation and communication loads for parallel processing. It adequately represents an important class of problems with applications in parallel and distributed system scheduling, various types of data processing, scientific and engineering computation and sensor networks. Cao [15] used an ant-like self-organizing mechanism to achieve system-wide Grid load balancing through a collection of

simple local interactions between the Grid nodes. In this model, multiple resource management agents cooperate to achieve automatic load balancing of distributed job queues. Each ant takes two sets of  $m$  steps in succession to determine the least and the most loaded nodes, respectively. The two nodes then redistribute the load between themselves. After a series of successive redistributions, the system-wide uniform load balancing could be achieved.

The authors in [16] presented two new distributed swarm intelligence inspired load-balancing algorithms. One of these algorithms is based on ant colony optimization and the other algorithm is based on particle swarm optimization. The performance of both the algorithms is evaluated using several performance criteria such as make span and load-balancing level. The simulation of these approaches using a Grid simulation toolkit was conducted. The experimental results showed that the proposed algorithms could perform very well in a Grid environment. Especially, the use of particle swarm optimization can yield better performance results in many scenarios than the ant colony approach. Erdil and Lewis [17] described information dissemination protocols to distribute the load in a way without using load rebalancing through job migration. However, it becomes more difficult and cost prohibitive for large-scale heterogeneous Grids. Essentially, in their model, the nodes adjust their advertising rates and aggressiveness to influence where jobs get scheduled. Subrata et al. [18] proposed a game-theoretic solution to the grid load-balancing problem. The algorithm developed combines the inherent efficiency of the centralized approach and the fault-tolerant nature of the distributed, decentralized approach. The grid load-balancing problem is modelled as a noncooperative game, where the objective is to reach the Nash equilibrium. One advantage of this scheme is the relatively low overhead and robust performance against inaccuracies in performance prediction information.

Zikos et al. [19] studied the site allocation scheduling of nonclairvoyant jobs in two-level heterogeneous grid architecture. Three scheduling policies at grid level which utilize site load information are examined. The aim is the reduction of site load information traffic, while at the same time means response time of jobs and fairness in utilization between the heterogeneous sites are of great interest. A simulation model is used to evaluate performance under various conditions. The simulation results showed that considerable decrement in site load information traffic and utilization fairness can be achieved at the expense of a slight increase in response time. Fernandes et al. [20] proposed a route load-balancing algorithm, which is designed to equally distribute the load of tasks for parallel applications. This algorithm uses the message routing concepts to define the computer neighbourhood. If any resource becomes overloaded, then the neighbour's load is

evaluated. If the neighbour node is not overloaded then the tasks are transferred to it. In [21], Balasangameshwara and Raju proposed a fault-tolerant hybrid load-balancing algorithm. Their algorithm is carried out in two phases. In the first phase, a static load-balancing policy selects the desired effective sites to carry out the submitted job. If any of the sites is unable to complete the assigned job, a new site is located using the dynamic load-balancing policy. By this way, the job failure is identified and load is redistributed to the underloaded resources. They claimed their algorithm to perform well in large Grid environments.

Li et al. [22] addressed the load-balancing problem by presenting a hybrid approach to the load balancing of sequential tasks under grid computing environments. The main objective is to arrive at task assignments that could achieve minimum execution time, maximum node utilization and a well-balanced load across all the nodes involved in a grid. A first-come-first-served and a carefully designed GA are selected as representatives of both classes to work together to accomplish the goal. The simulation results showed that the algorithm can achieve a better load-balancing performance as compared to its ‘pure’ counterparts. The authors in [23] introduced a hybrid load-balancing policy to integrate static and dynamic load-balancing technologies. Essentially, a static load-balancing policy is applied to select effective and suitable node sets. This minimizes the unbalanced load probability caused by assigning tasks to ineffective nodes. When a node reveals the possible inability to continue providing resources, the dynamic load-balancing policy determines whether the node in question is ineffective to provide load assignment. Then the system obtains a new replacement node within a short time, to maintain system execution performance. Cao et al. [24] demonstrated various artificial intelligence techniques which can be utilized to achieve effective workload and resource management. A combination of intelligent agents and multi-agent approaches is applied to both local grid resource scheduling and global grid load balancing. Each agent is a representative of a local grid resource and utilizes predictive application performance data with iterative heuristic algorithms to engineer local load balancing across multiple hosts.

Wu et al. [25] used a novel multi-agent reinforcement learning method, called ordinal sharing learning (OSL) method, is proposed for job scheduling problems, especially, for realizing load balancing in Grids. The approach circumvents the scalability problem by using an ordinal distributed learning strategy, and realizes multi-agent coordination based on an information sharing mechanism with limited communication. Zheng et al. [26] addressed the problem of determining which group an arriving job should be allocated to and how its load can be distributed among

computers in the group to optimize the performance. The algorithms guarantee finding a load distribution over computers in a group that leads to the minimum response time or computational cost. The effect of pricing on load distribution is studied by considering a simple pricing function. Three fully distributed algorithms are developed to decide which group the load should be allocated to, taking into account the communication cost among groups. These algorithms use different information exchange methods and a resource estimation technique to improve the accuracy of load balancing. Li [27] considered optimal load distribution in a nondedicated cluster or grid computing system with heterogeneous servers processing both generic and dedicated applications. The goal of load balancing is to find an optimal load distribution strategy for generic tasks on heterogeneous servers preloaded by different amount of dedicated tasks such that the overall ART of generic applications is minimized. This optimization problem is solved for three different queueing disciplines, namely, dedicated applications without priorities, prioritized dedicated applications without preemption, and prioritized dedicated applications with preemption.

In [7], Y. Hao et al. presented a load-balancing mechanism called “Load balancing on Enhanced GridSim with Deadline Control” (EGDC) for Grid environment based on deadline control for tasks. At the outset, first, resources check their states and make a request to the Grid Broker according to the change of state in load. Then, the Grid Broker assigns the Gridlets between the resources and scheduling for load balancing under the deadline request. The EGDC is simulated on the GridSim platform. The EGDC is shown to reduce the response time, improve the finished rate of the Gridlet and reduce the resubmitted time. But the scheduling mechanism of the EGDC suffers from serious drawbacks. Some Gridlets cannot be finished at the first resource scheduling before their deadline. So, again all these unfinished Gridlets can be rescheduled for execution on the other resources. But, if all resources present in the Grid system fail to execute some of these unfinished Gridlets due to deadline failure, then no mechanism of recovery from deadline failure of the Gridlets is present in the scheduling mechanism of the EGDC. This is viewed as a serious limitation of the EGDC and therefore needs research attention for its further improvement.

### 3 Background

In this Section, we present a brief discussion of the Grid simulator (GridSim) and review the existing load-balancing schemes on GridSim which are useful for rest portion of this paper.

### 3.1 GridSim

The GridSim is a very popular Grid simulation tool [10]. The GridSim toolkit supports modelling and simulation of a wide range of resources, such as single or multiprocessors, shared or distributed memory machines such as PCs, workstations, SMPs, and clusters with different capabilities and configurations. The GridSim is built on a general-purpose discrete event simulation package called SimJava [28], which is implemented in Java. The GridSim’s highly specialized grid simulation-based classes are developed extending SimJava’s simulation foundation classes. The various layers of the GridSim are shown in Fig. 1 [7]. A brief discussion on each layer of the GridSim is as follows.

The user dispatches its jobs to the Grid through its own broker called the *Grid Broker*. Each user is connected to an instance of the broker entity. Every job of the user is first submitted to its broker and then the broker schedules the parametric tasks according to the user’s scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global directory entity. Every broker tries to optimize the policy of its user and therefore, the brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market’s demand–supply situation. The *Grid Resource* is next to the Grid Broker in the hierarchy. Each Grid Resource may differ from the rest of resources in terms of the number of processors, cost of processing, speed of processing, internal process scheduling policy, local load factor and the time zone [10]. The Grid Information Service provides resource registration services and keeps track of a list of resources available in the Grid. The brokers can query this for resource contact, configuration and status information. The *Machine* is a processing entity manager. It

is responsible for task scheduling and load balancing of its *Processing Elements* (PEs). The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs to Gridlet jobs [7].

A *Gridlet* is an entity that contains all the information related to a job and its execution management details such as the job length expressed in million instructions per second (MIPS), the disk I/O operations, the size of input and output files and the job originator. These basic parameters in the GridSim determine the execution time of a job (Gridlet), the time required to transport input and output files between the users and the remote resources, and returning the processed Gridlets back to the originator along with the results [10].

### 3.2 Load-balancing schemes on GridSim

There exist many centralized load-balancing schemes for the Grid. However, those as such cannot be used in GridSim due to the differences in their frameworks [7]. The most important load-balancing schemes include the without load balancing (WLB) [10], load balancing on GridSim (LBGS) [29], load balancing on enhanced GridSim (LBEGS) [11] and EGDC [7].

In WLB, the GridSim resource simulator adopts internal events to simulate the execution and allocation of PEs to the Gridlet jobs. When a job arrives, the space shared systems start its execution immediately if there is a free PE. Otherwise, it is queued. During the Gridlet assignment, the job-processing time is determined and the event is scheduled for delivery at the end of the execution time. Whenever a Gridlet finishes, an internal event is delivered to signify the completion of the scheduled Gridlet job. The resource simulator then frees the PE allocated to it and checks if there are any other job waiting in the queue. If there are jobs

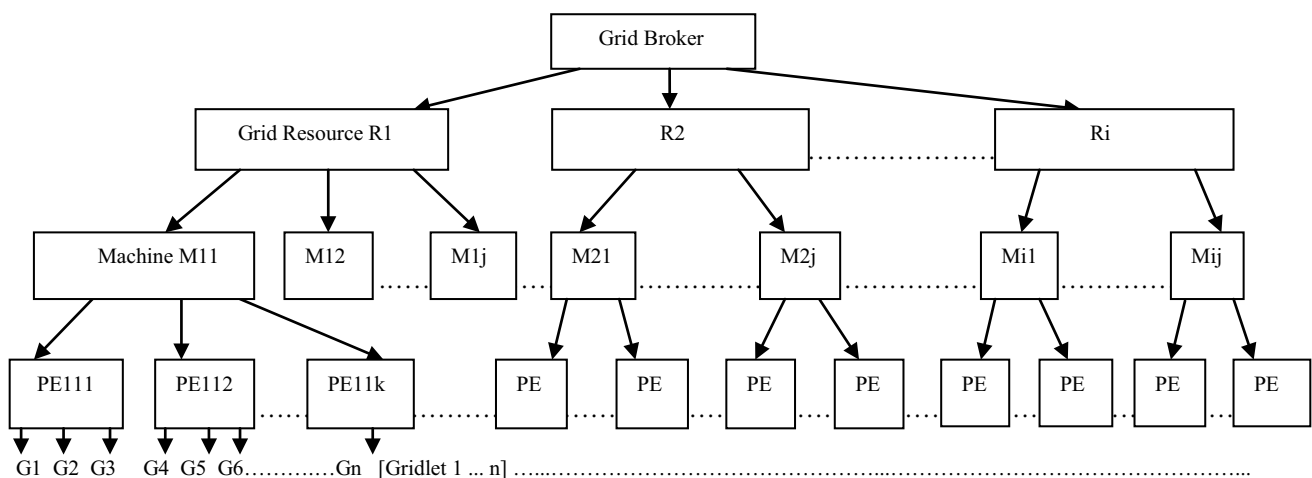


Fig. 1 The Structure of a grid in GridSim

waiting in the queue, then the resource simulator selects a suitable job depending on the selection policy and assigns it to the free PE. If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a Gridlet completion event. If there are Gridlets in the submission queue, then depending on the allocation policy, the GridSim selects a suitable Gridlet from the queue and assigns it to the PE (or a suitable PE if more than one PE is free). After that, the completed Gridlet is sent back to the broker or user and removed from the execution set.

The LBGS is a dynamic tree-based model which represents the Grid architecture. Based on a tree model, the said algorithm possesses the following main features: (i) it is layered, (ii) it supports heterogeneity and scalability and (iii) it is totally independent of any physical architecture of a Grid. Their simulations show a significant improvement in the mean response time with a reduction of communication cost. The load-balancing algorithm LBEGS is an improvement over the LBGS. It gives the details of the load calculation methods of the PE, Machine, and Grid Resource. At the same time, the model also proposes the method of load balancing between PEs, machines, and resources. The above said scheme not only reduces the communication overhead of Grid resources but also cuts down the idle time of the resources during the process of load balancing based on GridSim. The effectiveness in terms of communication overhead and response time reduction is gauged.

In [7], the authors presented a distributed load-balancing scheme called the EGDC for the grid environments that provides deadline control for tasks.

### 3.3 Enhanced GridSim with load balancing based on deadline control (EGDC): a brief discussion

In this Subsection, a brief description of the load-balancing mechanism called EGDC [7] is presented.

Suppose that the load of one resource is the  $l.currentload$  and a Gridlet  $g$  is assigned to resource  $r$  with a deadline in  $t$  seconds. Then the new load of a resource is calculated using the following expression.

$$l.currentload = l.currentload + (g.Gridletlength/t/resource.capacity)$$

Then, the state of every resource is checked periodically. The states are classified into three categories: underloaded, normally loaded, and overloaded. If the load of the resource is more than resource level load, we call the state of the resource “over loaded”. If the load of resource is less than machine level load, we call the state of the resource “underloaded”. If the load of resource is in between resource level load to machine level load, we call the state of resource “normally loaded”. After checking the state of every resource,

the Grid Broker inserts it into the list that it should belong to. When the Grid Broker gets information from a resource about its changed state, it changes the state to new load state in its Grid information. The Grid Broker inserts the resource into the Overload list or the Normally load list or the Underload list depending upon whether the state is overloaded, normally loaded or underloaded, respectively.

If the state of a resource is over loaded, we can transform Gridlets from that resource and if the state of a resource is under loaded, we can assign more Gridlets to the resource. We make a list of Gridlets which we want to transfer from an overloaded resource to an underloaded resource and then insert that list of Gridlets into the Unassignedgridlet list for subsequent scheduling. The Unassignedgridlet list is used to store the unfinished Gridlets coming from the overloaded resource and the new arriving Gridlets. Then, the Grid Broker schedules the Gridlets coming from the Unassignedgridlet list to the underloaded resources using a scheduling mechanism. The EGDC is simulated on the GridSim platform. The EGDC is shown to reduce the response time, improve the finished rate of the Gridlet and reduce the resubmitted time.

However, the scheduling mechanism of the EGDC suffers from serious drawbacks. Some Gridlets cannot be finished at the first resource scheduling before their deadline. So, again all these unfinished Gridlets can be rescheduled for execution on the other resources. But, if all resources present in the Grid system, fail to execute some of these unfinished Gridlets due to deadline failure, then no mechanism of recovery from deadline failure of the Gridlets is present in the scheduling mechanism of the EGDC. This is viewed as serious limitation of the EGDC. Hence, it requires further improvement in the scheduling mechanism. In the next section to follow, we improve the existing EGDC and propose a new method called “Enhanced GridSim with Load balancing based on Deadline Failure Recovery” (EGDFR).

## 4 The proposed work: enhanced GridSim with load balancing based on deadline failure recovery (EGDFR)

In this section, we propose a new load-balancing scheme called “Enhanced GridSim with Load balancing based on Deadline Failure Recovery” (EGDFR). The work proposed in this Section is an improved version of the existing EGDC [7] in which it performs load balancing by providing a scheduling system which includes the mechanism of recovery from deadline failure of the Gridlets. This new mechanism of scheduling reduces the execution time to a large extent. The incorporation of the said features of the proposed method makes it quite attractive in Grid applications. The proposed method is supported with an algorithm and

```

BEGIN
/*Calculate the new load of a resource*/
float newload (Gridlet g, Resource r)
{
factor = (Lg) / (Dg *Cr);
Lr = currentload + (factor);
return Lr;
}

/*Check the state of every resource*/
for all resources do
calculate Lr which is the current load of a resource;
if (Lr < rb)
resource is “under loaded” and add this resource to UnderloadedResourceList;
else if (Lr > rt)
resource is “over loaded” and add this resource to OverloadedResourceList;
else
resource is “normally loaded” and add this resource to NormalResourceList;
end for

/*Transfer Gridlets from overloaded resources to UnassignedGridletlist*/
while (OverloadedResourceList is not empty and UnderloadedResourceList is not empty)
Make a list of Gridlets which we want to transfer from an overloaded Resource to an underloaded Resource;
Insert that list of Gridlets in to UnassignedGridletlist;
end while

Store the new arriving Gridlets in to UnassignedGridletlist;

Arrange the underloaded resources present in the UnderloadedResourceList according to the highest capacity of the resource;
Arrange all the Gridlets present in the UnassignedGridletlist according to the largest size of the Gridlet;

/*Assigning Gridlets from UnassignedGridletlist to underloaded resources */
while (UnderloadedResourceList is not empty and UnassignedGridletlist is not empty)
for every Resource r in UnderloadedResourceList
for every Gridlet g in UnassignedGridletlist
temp = (Lg) / (Dg *Cr);
total = Lr + temp;
if (total <= rt)
Assigning Gridlet g to Resource r;
{
if (Eg <= Dg)
Execution of the Gridlet g on Resource r within the Deadline;
else
Apply the recovery mechanism from Deadline failure of the Gridlet g on Resource r;
}
else
Move to next underloaded resource present in the UnderloadedResourceList for execution of that Gridlet g;
end if
end while
END

```

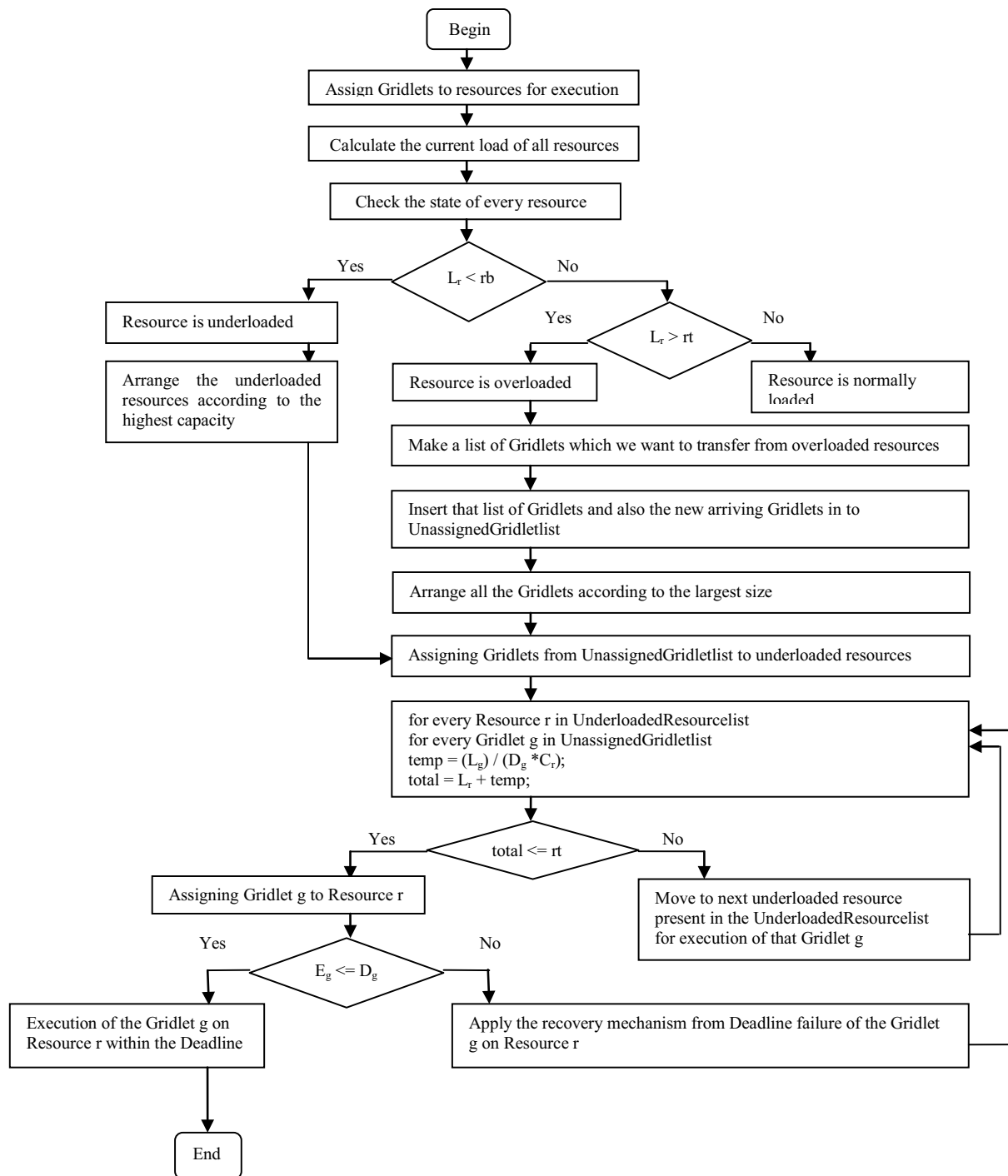
a flowchart followed by a brief description. The following notation and terminologies are used throughout this paper.

#### 4.1 Notation and terminologies

$L_r$  Current load of a resource  
 $C_r$  Capacity of a resource  
PE No. of PEs in a machine  
CPE Capacity of a PE

Underloaded resource list A list of underloaded resources  
OverloadedResourceList A list of overloaded resources  
Normal resource list A list of normally loaded resources

$rt$  Resource load level  
 $rb$  Machine load level  
 $m$  No. of machines in a resource  
 $D_g$  Deadline given to the Gridlet  
 $L_g$  Length of the Gridlet  
 $E_g$  Execution time of the Gridlet on a resource



**Fig. 2** Flowchart of the EGDFR scheme

Next, we present the pseudo code of the proposed algorithm in Subsect. 4.1.

## 4.2 Proposed algorithm

The detailed flowchart illustrating the various steps of the new approach of load balancing is presented in Fig. 2.

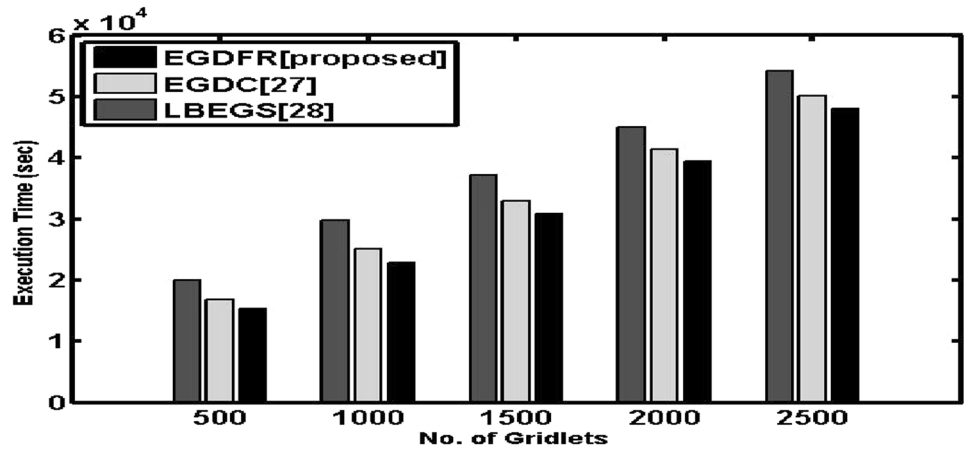
## 4.3 Description of the proposed algorithm

In this Subsection, a brief stepwise description of the proposed algorithm (EGDRF) is presented.

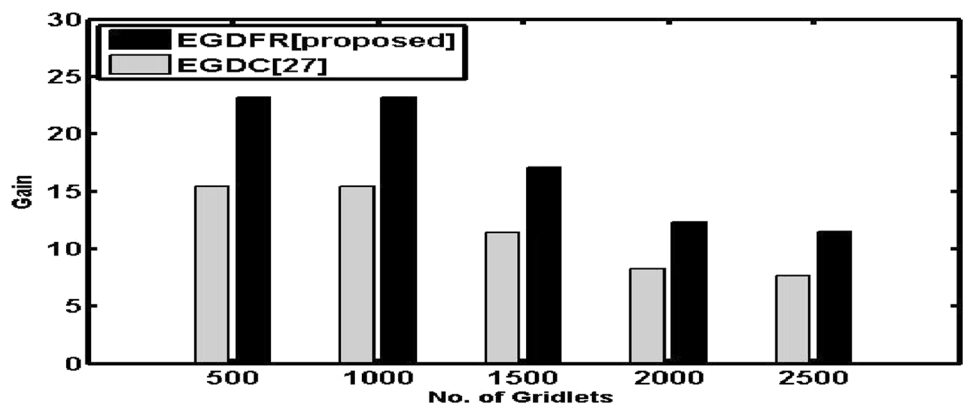
In the proposed model, there is a Grid system which consists of heterogeneous resources which are connected by communication channels. Here, the GridBroker is responsible for



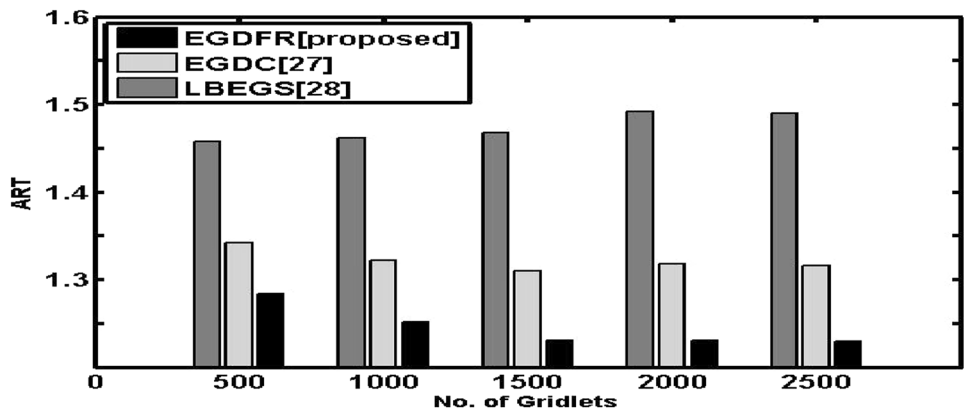
**Fig. 3** Execution time versus number of gridlets



**Fig. 4** Percentage gain in execution time versus number of gridlets



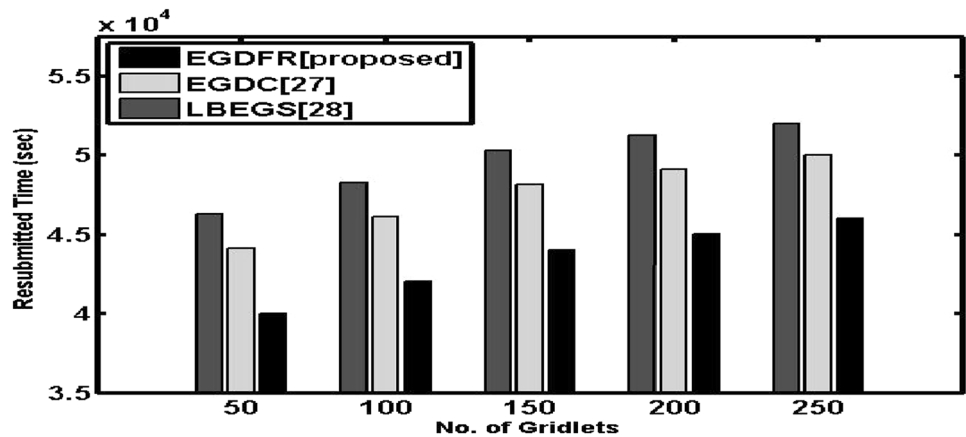
**Fig. 5** ART versus number of gridlets



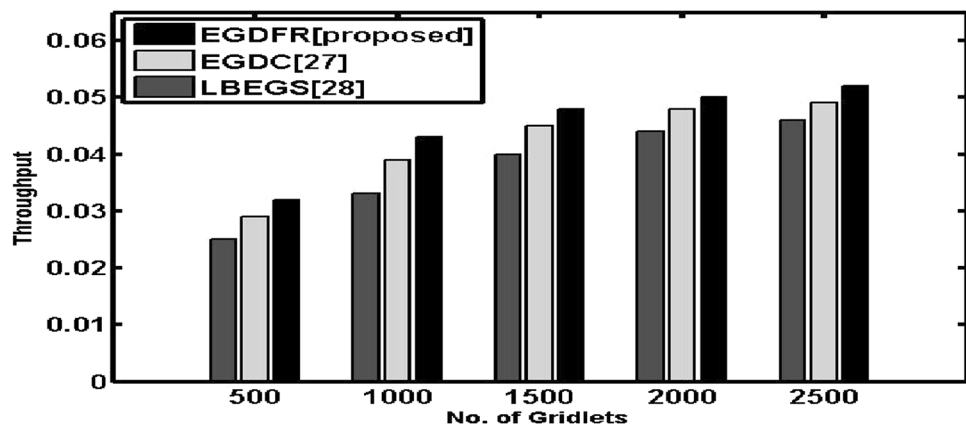
scheduling of Gridlets on the heterogeneous resources present in the Grid system. During the scheduling process, first the GridBroker checks the condition of all the resources present in the Grid system. Then according to the state of all resources, the GridBroker schedules the new Gridlets on those resources by maintaining overall load balance of the total Grid system.

*Step 1* First, for checking the state of all the resources, the GridBroker has to calculate the current load of all resources. Suppose that the current load of a resource  $r$  is  $L_r$ . When a Gridlet  $g$  is assigned to a resource  $r$  with a deadline, the GridBroker estimates the new load on the resource using the following equation [15].

**Fig. 6** Resubmitted time versus number of gridlets



**Fig. 7** Throughput versus number of gridlets



/\*Calculate the new load of a resource\*/

$$\text{factor} = (L_g)/(D_g * C_r); \tag{1}$$

$$L_r = \text{currentload} + (\text{factor});$$

Let us assume that the resource  $r$  has a machine list given as (machine 1, machine 2,..., machine  $i$ ,..., machine  $m$ ). The machine  $i$  can be illustrated as the machine  $i$  (int  $id_i$ , int  $PE_i$ , int  $CPE_i$ ) representing the id of the resource, the number of PEs, and the rating of every PE respectively. The  $CPE_i$  can be expressed in Standard Performance Evaluation Corporation (SPEC) or MIPS. The resource’s capacity ( $C_r$ ) can be calculated as follows

$$C_r = \sum_{i=1}^m (PE_i \times CPE_i) \tag{2}$$

*Step 2* Second, the Grid Broker check the recent state of all the resources. The states are classified into three categories: *underloaded*, *normally loaded*, and *overloaded*. If the load of the resource is more than  $rt$ , we call the state of the resource “overloaded”. If the load of resource is  $<rb$ , we call the state of the resource “underloaded”. If the load of a resource is in between  $rt$  to  $rb$ , we call the state of resource “normally

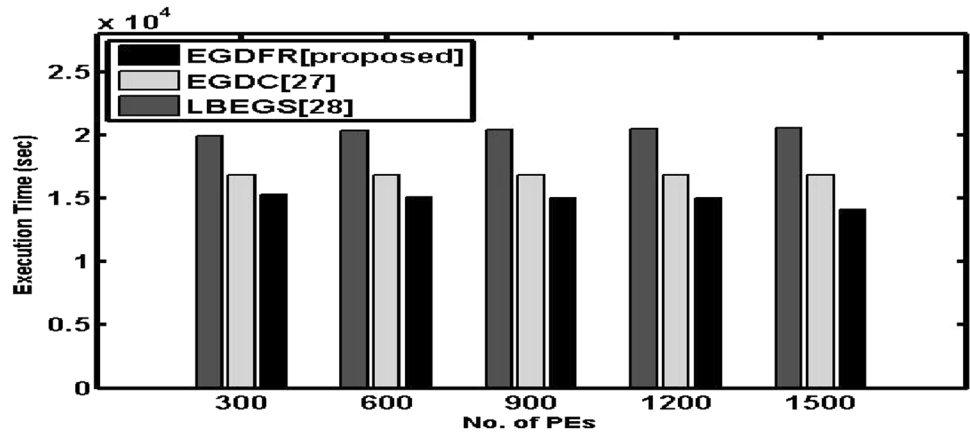
loaded”. Here  $rt$ ,  $rb$  are known as the *resource level load* and the *machine level load*, respectively. After checking the state of the resources, the Grid Broker inserts it into the list that it should belong to. The Grid Broker inserts the resource into the *OverloadedResourcelist* or the *NormalResoucelist* or the *UnderloadedResourcelist* depending upon whether the state is overloaded, normally loaded or underloaded, respectively.

/\*Check the state of every resource\*/

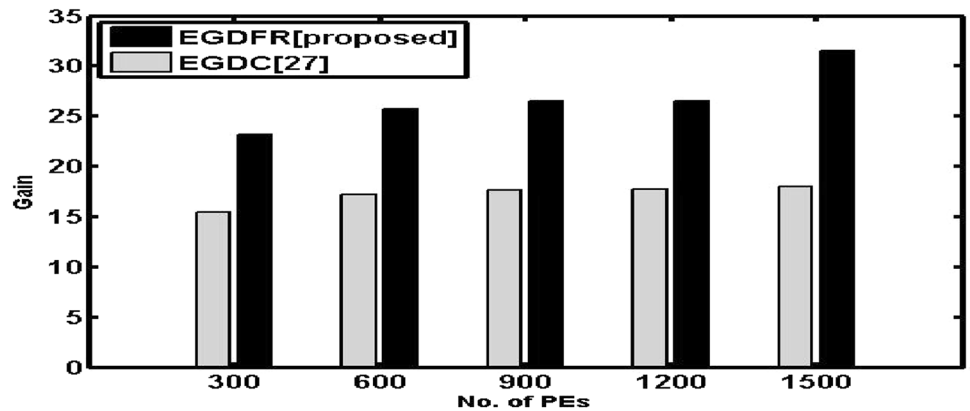
```

for all resources do
  calculate  $L_r$  which is the current load of a resource;
  if ( $L_r < rb$ )
    resource is “under loaded”
    and add this resource to UnderloadedResourcelist;
  else if ( $L_r > rt$ )
    resource is “over
loaded” and add this resource to OverloadedResourcelist;
  else
    resource is “normally loaded”
    and add this resource to NormalResourcelist;
end for
    
```

**Fig. 8** Execution time versus number of PEs



**Fig. 9** Percentage gain in execution time versus number of PEs



*Step 3* In the third step, if the state of any resource is over loaded, then the Grid Broker make a list of Gridlets which need to be transfer from that overloaded resource to one of the underloaded resources present in the UnderloadedResourceList and then insert that list of Gridlets into the UnassignedGridletlist for subsequent scheduling. Now, the GridBroker can also schedule new Gridlets on the underloaded resources present in the UnderloadedResourceList. The UnassignedGridletlist is used to store the unfinished Gridlets coming from the overloaded resource and the new arriving Gridlets.

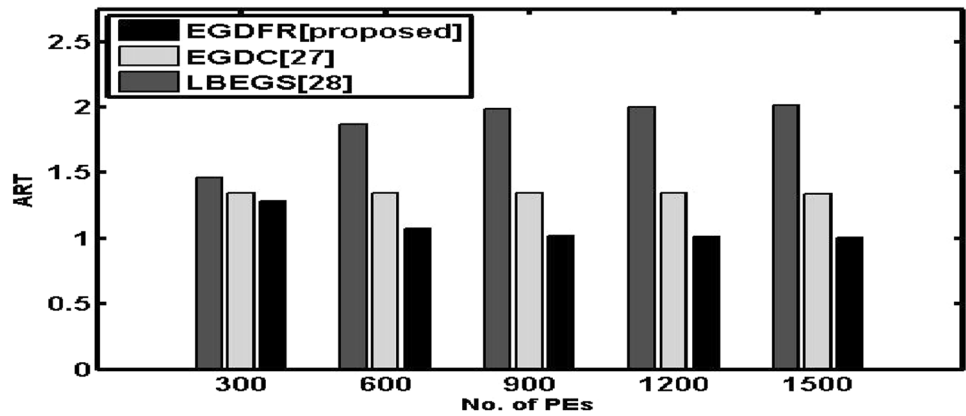
*Step 4* From the beginning of scheduling, our priority always should be assigning the largest Gridlet present in the UnassignedGridletlist to one of the best underloaded resources present in the UnderloadedResourceList. It directs that all the Gridlets should be executed on the heterogeneous resources within the deadline. The Gridlet execution time must be less than the deadline assigned to the Gridlet. So the Gridlet execution time becomes important due to resource heterogeneity which is influenced by the capacity of the resources. The underloaded resource having the highest capacity has the low execution time. Our algorithm considers these facts. Therefore, we arrange the under loaded resources present in the UnderloadedResourceList

according to the highest capacity of the resource. Also, we arrange the Gridlets present in the UnassignedGridletlist according to the largest size of the Gridlet. Now, the Grid-Broker can schedule the largest Gridlet present in the UnassignedGridletlist to the underloaded resource present in the UnderloadedResourceList having highest capacity.

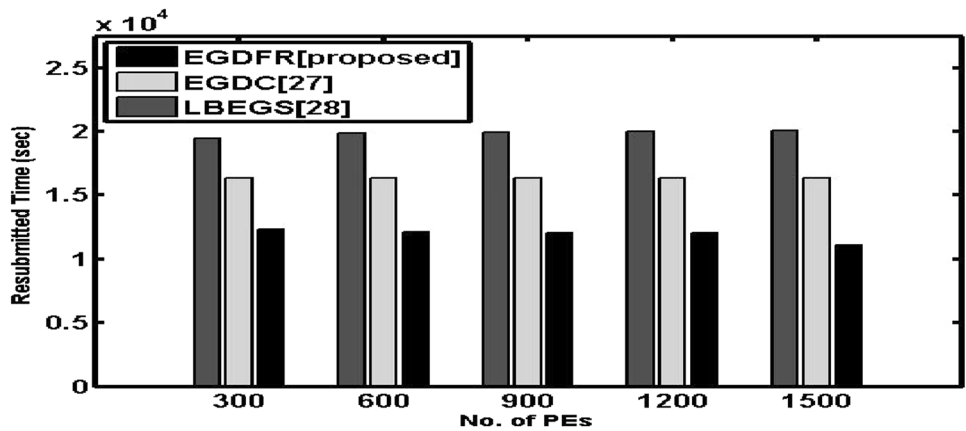
*Step 5* Next, we assign the Gridlets coming from the UnassignedGridletlist to the underloaded resources present in the UnderloadedResourceList. In our scheduling mechanism, the GridBroker can assign Gridlets to a resource until the load of the resource will be less than or equal to  $rt$  means maximum up to the upper bound of the normally loaded range i.e.,  $rb-rt$  and the state of resource will be either normallyloaded or underloaded. When after assigning a Gridlet to the resource, the load of resource will exceed the value of  $rt$ , that means the condition fails and the state of resource will be overloaded. So when the condition fails, then the GridBroker cannot assign that Gridlet to the resource and shall have to move to the next under loaded resource present in the UnderloadedResourceList for execution of that Gridlet.

Some Gridlets cannot be finished at the first scheduling due to deadline failure. These are called unfinished Gridlets and the GridBroker insert that list of Gridlets

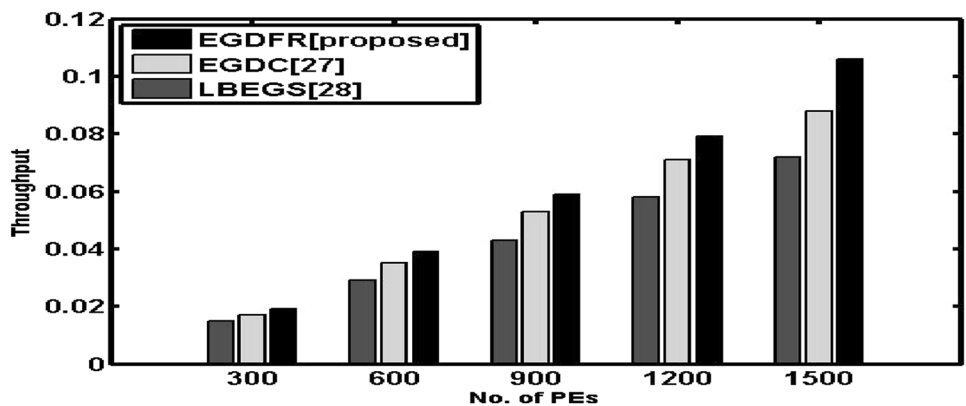
**Fig. 10** ART versus number of PEs



**Fig. 11** Resubmitted time versus number of PEs



**Fig. 12** Throughput versus number of PEs



into the UnfinishedGridletlist. Now, all these unfinished Gridlets present in the UnfinishedGridletlist can be rescheduled for execution on the other underloaded resources present in the UnderloadedResourceList. But after providing the Gridlets to the best resources, if all resources present in the Grid system, fail to execute some of these unfinished Gridlets due to deadline failure, then a recovery mechanism from deadline failure of the Gridlets is used in the new proposed scheduling method.

Here, in the new proposed scheduling mechanism, after assigning a Gridlet to the resource, if the resource cannot be finished at the first scheduling due to deadline failure, then there is no need to store that Gridlet in UnfinishedGridletlist for rescheduling it later. For a Gridlet having the problem of deadline failure, is going to transfer at that present time to the next underloaded resource present in the UnderloadedResourceList for execution. Before transferring the Gridlet to the next underloaded resource, the GridBroker is going to save the state of the

running Gridlet to stable storage means up to which the execution time of the Gridlet is not going to exceed the deadline. This saved state can be used to resume execution of the Gridlet from the point in the computation where the check-point was last taken, instead of restarting the Gridlet from its very beginning. The Gridlet is going to transfer from the present resource to the next underloaded resource present in the UnderloadedResourceList. Once the loading completes in the next underloaded resource, the Gridlet is recovered and resumes execution from the point where the check-point was last taken. The execution time reduces to a large extent by resuming the execution from such intermediate points. This recovery mechanism from deadline failure of the Gridlets is popularly known as check-pointing.

## 5 Simulation and results

The proposed algorithm was implemented using the GridSim5.0 simulator [10]. The details of the experimental setup and results are described below. Finally, the results of the simulation are compared with EGDC [7] and LBEGS [11]. The comparison of the proposed algorithm EGDFR is based on the total execution time, ART, resubmitted time and throughput. We use Windows 7 on an Intel Core (1.73 and 1.73 GHz), with 3 GB of RAM and 1000 GB of hard disk for the simulation purpose.

### 5.1 Performance metrics

In this work, we have considered four performance metrics. The metrics considered here are total execution time, ART, resubmitted time and throughput. First, we consider the total execution time as the performance metric to measure the algorithm's efficiency. It indicates the time at which all Gridlets get executed. The execution time is the total simulation time which is measured from the time the first Gridlet is sent to the Grid until the last Gridlet comes out of the Grid.

Second, we consider the ART of the Gridlets processed in the system as the performance metric. If  $n$  no. of Gridlets is processed by the system, then the ART is given by

$$\text{Average response time (ART)} = \frac{1}{n} \sum_{i=1}^n (\text{Finish}_i - \text{Arrival}_i) \quad (3)$$

where  $\text{Arrival}_i$  is the time at which the  $i$ th Gridlet arrives, and  $\text{Finish}_i$  is the time at which it leaves the system. Some of the Gridlets may not be executed before their deadline. The number of Gridlets that cannot be finished on time called unfinished Gridlets can be rescheduled for execution. Therefore, the resubmitted time is another standard for our

test. Throughput is used to measure the ability of the grid to accommodate jobs. Throughput is defined as

$$\text{Throughput} = \frac{n}{T_n} \quad (4)$$

where  $n$  is the total number of Gridlets submitted and  $T_n$  is the total amount of time necessary to complete  $n$  Gridlets.

### 5.2 Performance evaluation

We conducted our simulations for three different but interesting cases: (i) Case 1: Simulation with constant number of PEs, (ii) Case 2: Simulation with constant number of Gridlets, (iii) Case 3: Simulation with varying Job size.

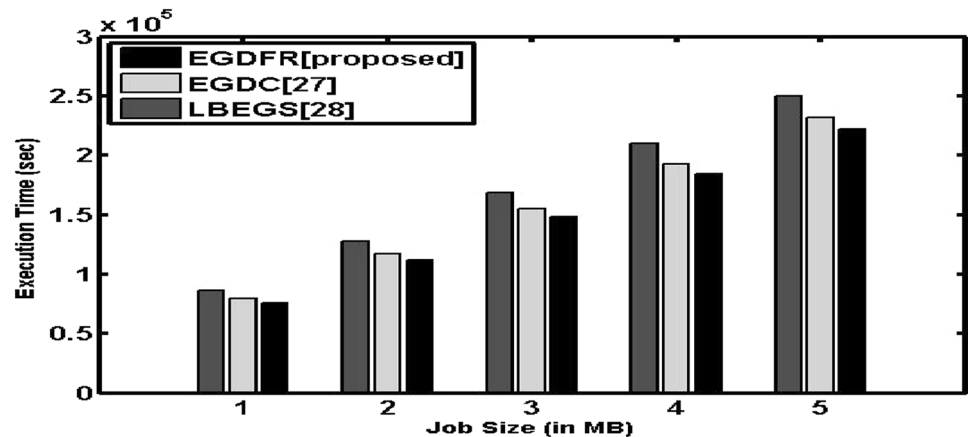
#### 5.2.1 Case 1: simulation with constant number of PEs

In this case, the simulation is conducted assuming the number of PEs to be constant and Gridlets to vary. Without loss of generality, we assume that there are 30 resources in the system and every resource has two machines. Every machine has five PEs. So, the Grid system has total 300 PEs. Due to the resource heterogeneity, every PE has the rating between 1 and 5 MIPS. Here, we express the Gridlet length in PEs (1 million instructions). Every Gridlet length is between 1 and 5 million instructions. Every Grid has a deadline within a range of 1–6 s. In our simulation model, we have considered the heterogeneous resources that are connected by communication channels. The network bandwidth connecting two resources varies from 0.5 to 10 Mbps. According to [7, 11], the resource/machine/PE threshold is 0.8/0.75/0.6. Therefore, we set resource threshold ( $r$ ) = 0.8, machine threshold ( $ml$ ) = 0.75. Figures 3, 4, 5, 6, and 7 show the results of comparison of our proposed method with other for the case 1.

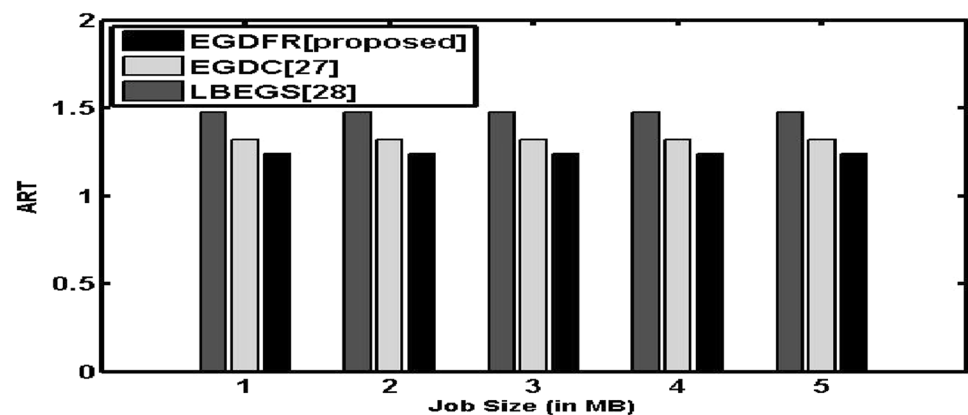
In Figs. 3 and 4, the observations are taken by varying the number of Gridlets starting from 0 and ending at 2500 with a step of 500, keeping the number of PEs constant at 300. Figure 3 presents a comparative look of the total execution time versus number of Gridlets of our proposed algorithm EGDFR with the existing methods. We have executed the simulation a number of times and got the exact results. In all the cases the total execution time of our EGDFR is found to be less than EGDC and LBEGS with the increase of Gridlets. In Fig. 4, the results are presented for EGDFR and EGDC. Figure 4 shows that the gain in execution time is higher in EGDFR as compared to EGDC.

In Fig. 5, the observations are taken by varying the number of Gridlets starting from 50 and ending at 250 with a step of 50 while the number of PEs is kept constant at 300. Some Gridlets cannot be finished at the first resource scheduling before their deadline. So, a Gridlet having deadline failure is going to transfer to the other resource at that present time, not

**Fig. 13** Execution time versus job size



**Fig. 14** ART versus job size



later. Therefore, the sum of resubmitted time is another standard for our test. Here, after providing continuously 2500 Gridlets for execution to 30 resources, from that we have provided 250 unfinished Gridlets for rescheduling to 30 resources. Figure 5 establishes that the resubmitted time of the EGDFR is less than those of the others with the increasing of Gridlets.

In Figs. 6, and 7, the observations are taken by varying the number of Gridlets starting from 0 and ending at 2500 with a step of 500, keeping the number of PEs constant at 300. Figure 5 presents a comparative look of the ART versus number of Gridlets and the results are presented for EGDC, LBEGS and our proposed method EGDFR. Figure 5 shows that the ART is lower in the case of EGDFR as compared to the EGDC and LBEGS. Figure 7 compares the throughput versus number of Gridlets for our proposed method EGDFR with the existing ones. From the results, it is quite clear that the throughput of EGDFR is always more than EGDC and LBEGS with the increase in the number of Gridlets.

### 5.2.2 Case 2: simulation with constant number of Gridlets

Here, we consider another case and conduct simulation assuming the number of Gridlets to be fixed and the PEs to

vary. Here, we assume that each resource has two machines and every machine has five PEs. Due to the resource heterogeneity, every PE has the rating between 1 and 5 MIPS. The Grid is created by varying the number of resources while keeping the number of Gridlets constant. We express the Gridlet length in PEs (1 million instructions). The length of each Gridlet lies between 1 and 5 million instructions. Every Grid has a deadline within a range of 1–6 s. In our simulation model, we have considered the heterogeneous resources that are connected by communication channels. The network bandwidth connecting two resources varies from 0.5 to 10 Mbps. According to [7, 11], the resource/machine/PE threshold is 0.8/0.75/0.6. Therefore, we set resource threshold ( $rl$ ) = 0.8, machine threshold ( $ml$ ) = 0.75. The results plotted in Figs. 8, 9, 10, 11, and 12 compare the proposed method with the existing ones for the case 2.

In Figs. 8, and 9, the observations are taken by varying the number of resources starting from 30 to 150 with a step of 30, while the numbers of Gridlets are kept constant at 1000. Figure 8 presents a comparative look of the total execution time versus number of PEs of our proposed algorithm EGDFR with the existing methods. We have executed

the simulation a number of times and got the exact results. In all the cases the total execution time of our EGDFR is found to be less than EGDC and LBEGS with the increase of resources. In Fig. 9, the results are presented for EGDFR and EGDC. From Fig. 9, it is evident that the gain in execution time is higher in EGDFR as compared to EGDC.

In Fig. 10, the observations are taken by varying the number of resources starting from 30 and ending at 150 with a step of 30 and the number of Gridlets kept constant at 1000. Some Gridlets cannot be finished at the first resource scheduling before their deadline. So, a Gridlet having deadline failure is going to transfer to the other resource at that present time, not later. Therefore, the sum of resubmitted time is another standard for our test. Here, after providing continuously 1000 Gridlets for execution to resources starting from 30 and ending at 150, we have again provided 100 unfinished Gridlets for rescheduling to resources starting from 30 and ending at 150. Figure 10 establishes that the resubmitted time of the EGDFR is less than those of the others with the increase of resources.

In Figs. 11, and 12, the observations are taken by varying the number of resources starting from 30 to 150 with a step of 30, while the numbers of Gridlets are kept constant at 1000. Figure 11 presents a comparative look of the ART versus number of PEs and the results are presented for EGDC, LBEGS and our proposed method EGDFR. From Fig. 11, it is evident that the ART is lower in the case of EGDFR than the EGDC and LBEGS with the increase of resources. Figure 12 compares the throughput versus number of PEs for our proposed method EGDFR with the existing ones. From the results, it is quite clear that the throughput of EGDFR is always more than EGDC and LBEGS with the increase of resources.

### 5.2.3 Case 3: simulation with varying job size

It is very interesting to measure the performance of the algorithms by varying the job size. In this simulation, we have taken the observations by varying the job size from 1 to 5 MB. As we increase the job size, the performance of our proposed method becomes much better than EGDC and LBEGS in terms of the decrease in the total execution time and ART (Figs. 13, 14). Figure 13 establishes the total execution time of the EGDFR to be less than EGDC and LBEGS with the increase in job size. From Fig. 14, it is evident that the ART is lower in the case of EGDFR than the EGDC and LBEGS.

## 6 Conclusions

In this paper, we proposed a dynamic distributed load-balancing technique called “Enhanced GridSim with

Load balancing based on Deadline Failure Recovery” (EGDFR) for the heterogeneous Grid computing environment. We considered three cases for simulation, one with constant number of PEs, second with the constant number of Gridlets and third with the varying Job Size. Extensive simulations are conducted using GridSim5.0 for all these cases. From the simulation results, the proposed method (EGDFR) is observed to have better performance than the EGDC [7] and LBEGS [11] in terms of the total execution time, ART, resubmitted time and throughput. The proposed work with modification can be extended further for scheduling and fault tolerance on GridSim.

## References

1. Berman F, Fox G, Hey AJ (2003) Grid computing: making the global infrastructure a reality. Wiley, New York
2. Foster I, Kesselman C (eds) (1999) The grid: blueprint for a future computing infrastructure. Morgan Kaufmann Publishers, San Francisco
3. Myer T (2003) Grid computing: conceptual flyover for developers. IBM’s Developers work Grid Library, IBM Corporation, New York
4. Rathore N, Channa I (2014) Load balancing and job migration techniques in grid: a survey of recent trends. *Wirel Pers Commun* 79:1–37
5. Rathore N, Channa I (2011) A cogitative analysis of load balancing technique with job migration in grid environment. In: IEEE proceedings of the world congress on information and communication technology (WICT), pp 77–82
6. Shah R, Veeravalli B, Misra M (2007) On the design of adaptive and decentralized load-balancing algorithms with load estimation for computational grid environments. *IEEE Trans Parallel Distrib Syst* 18(12):1675–1686
7. Hao Y, Liu G, Wen N (2012) An enhanced load balancing mechanism based on deadline control on GridSim. *Future Gener Comput Syst* 28:657–665
8. Subrata R, Zomaya AY, Landfeldt B (2008) Game-theoretic approach for load balancing in computational grids. *IEEE Trans Parallel Distrib Syst* 19(1):66–76
9. Yagoubi B, Lilia HT, Maussa HS (2006) Load balancing in grid computing. *Asian J Inf Technol* 5(10):1095–1103
10. Murshed M, Buyya R, Abramson D (2001) GridSim: A toolkit for the modeling and simulation of global grids. Technical Report, Monash, CSSE
11. Qureshi K, Rehman A, Manuel P (2010) Enhanced GridSim architecture with load balancing. *J Supercomput* 57:1–11
12. Anand L, Ghose D, Mani V (1999) ELISA: an estimated load information scheduling algorithm for distributed computing systems. *Comput Math Appl* 37:57–85
13. Subrata R, Zomaya AY, Landfeldt B (2007) Artificial life techniques for load balancing in computational grids. *J Comput Syst Sci* 73:1176–1190
14. Bharadwaj V, Ghose D, Robertazzi TG (2003) Divisible load theory: a new paradigm for load scheduling in distributed systems. *Clust Comput* 6:7–17
15. Cao J (2004) Self-organizing agents for grid load balancing. In: Proceedings of the fifth IEEE/ACM international workshop on grid computing, GRID’04, Pittsburgh
16. Ludwig S, Moallem A (2011) Swarm intelligence approaches for grid load balancing. *J Grid Comput* 9:1–23

17. Erdil D, Lewis M (2012) Dynamic grid load sharing with adaptive dissemination protocols. *J Supercomput* 59:1–28
18. Subrata R, Zomaya AY, Landfeldt B (2008) Game-theoretic approach for load balancing in computational grids. *IEEE Trans Parallel Distrib Syst* 19(1):66–76
19. Zikos S, Karatza HD (2009) Communication cost effective scheduling policies of nonclairvoyant jobs with load balancing in a grid. *J Syst Softw* 82:2103–2116
20. Fernandes de Mello R, Senger LJ, Yang LT (2006) A routing load balancing policy for grid computing environments. In: *Proceedings of the 20th international conference on advanced information networking and applications, Aina'06*, vol. 1, pp 18–20
21. Balasangameshwara J, Raju N (2012) A hybrid policy for fault tolerant load balancing in grid computing environments. *J Netw Comput Appl* 35:412–422
22. Li Y, Yang Y, Ma M, Jhou L (2009) A hybrid load balancing strategy of sequential tasks for grid computing environments. *Future Gener Comput Syst* 25:819–828
23. Yan KQ, Wang SS, Wang SC, Chang CP (2009) Towards a hybrid load balancing policy in grid computing system. *Expert Syst Appl* 36:12054–12064
24. Cao J, Spooner DP, Jarvis SA, Nudd GR (2005) Grid load balancing using intelligent agents. *Future Gener Comput Syst* 21:135–149
25. Wu J, Xu X, Zhang P, Liu C (2011) A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Gener Comput Syst* 27:430–439
26. Zheng Q, Tham CK, Veeravalli B (2008) Dynamic load balancing and pricing in grid computing with communication delay. *J Grid Comput* 6:239–253
27. Li K (2008) Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments. *J Syst Archit* 54:111–123
28. Howell F, McNab R (1998) SimJava: a discrete event simulation package for Java with applications in computer systems modeling. In: *Proceedings of the 1st international conference on web-based modelling and simulation*, society for computer simulation, San Diego
29. Yagoubi B, Slimani Y (2006) Dynamic load balancing strategy for grid computing. *World Acad Sci Eng Technol* 13:90–95