Journal of Computer Science 10 (5): 763-773, 2014
ISSN: 1549-3636
© 2014 Science Publications
doi:10.3844/jcssp.2014.763.773 Published Online 10 (5) 2014 (http://www.thescipub.com/jcs.toc)

IMPROVING FAULT TOLERANT RESOURCE OPTIMIZED AWARE JOB SCHEDULING FOR GRID COMPUTING

K. Nirmala Devi and A. Tamilarasi

Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India

Received 2013-09-28; Revised 2013-11-20; Accepted 2014-01-04

ABSTRACT

Workflow brokers of existing Grid Scheduling Systems are lack of cooperation mechanism which causes inefficient schedules of application distributed resources and it also worsens the utilization of various resources including network bandwidth and computational cycles. Furthermore considering the literature, all of these existing brokering systems primarily evolved around models of centralized hierarchical or client/server. In such models, vital responsibility such as resource discovery is delegated to the centralized server machines, thus they are associated with well-known disadvantages regarding single point of failure, scalability and network congestion at links that are leading to the server. In order to overcome these issues, we implement a new approach for decentralized cooperative workflow scheduling in a dynamically distributed resource sharing environment of Grids. The various actors in the system namely the users who belong to multiple control domains, workflow brokers and resources work together enabling a single cooperative resource sharing environment. But this approach ignored the fact that each grid site may have its own fault-tolerance strategy because each site is itself an autonomous domain. For instance, if a grid site handles the job check-pointing mechanism, each computation node must have the ability of periodical transmission of transient state of the job execution by computational node to the server. When there is a failure of job, it will migrate to another computational node and resume from the last stored checkpoint. A Glow worm Swarm Optimization (GSO) for job scheduling is used to address the issue of heterogeneity in fault-tolerance of computational grid but Weighted GSO that overcomes the position update imperfections of general GSO in a more efficient manner shown during comparison analysis. This system supports four kinds of fault-tolerance mechanisms, including the job migration, job retry, check-pointing and the job replication mechanisms also considering risk nature of Grid computing environment. The risk relationship between jobs and nodes are defined by the security demand and the trust level. Our evaluation based simulation results show that our algorithm has shorter makespan and more efficient. We also analyze the efficiency of the proposed approach against a centralized coordinated workflow scheduling technique and show that our approach is more efficient than the centralized technique with respect to achieving highly coordinated schedules.

Keywords: Grid Scheduling, Single Point of Failure, Scalability and Network Congestion, GSO Overcomes the Position Update Imperfections, Centralized Technique Achieve Highly Coordinated Schedule

1. INTRODUCTION

The traditional approach to resource access in grid environments is based on a queuing model that provides best-effort quality of service. In this model jobs are queued until they can be matched with appropriate resources for execution. This approach ensures that access to resources is shared equally and fairly among all

Corresponding Author: K. Nirmala Devi, Department of Computer Application, Kongu Engineering College, Perundurai, Erode, Tamilnadu, India

users of the system, but can result in long delays when competition between users forces jobs to wait for resources to become available. For applications with only one job, or with a few jobs that can be submitted in parallel, these delays are encountered only once. For workflow applications with complex job hierarchies and interdependencies the delays are encountered many times. One way to improve quality of service for workflow applications is to use a model for resource allocation based on provisioning. With a provisioning model, for a given period of time resources are allocated for the exclusive use. It minimizes delays for queuing because the user's jobs no longer compete with other jobs for resource access. Also, in counterpoint to the model of queuing where resource allocation and scheduling occur on a per-job basis, the provisioning model allows resources to be allocated once and used for multiple jobs. Provisioning is slightly more complex than queuing in that it requires users to make more sophisticated resource allocation decisions.

There are two policies that can be used to guide these decisions. In static provisioning the application allocates all resources required for the computation before any other jobs being submitted and releases the resources only after all the jobs have finished. This method assumes that the number of resources required is known or can be predicted in advance. In dynamic provisioning resources are allocated by the system at runtime. This allows the pool of available resources to grow and shrink according to the changing needs of the application. This Dynamic provisioning does not require advanced knowledge of resource needs, but it does require policies for acquiring and releasing resources. It also relies on the ability of the provisioning system to acquire resources on-demand when they are needed, which may not be possible if the resources are shared with other users.

Advance reservation is a resource provisioning mechanism supported by many batch schedulers. Users create advance reservations by requesting slots from the batch scheduler that specify the number of resources to reserve and the start and end times of the reservation. During the reservation period the scheduler only runs jobs that belong to the user on the reserved resources. Although batch schedulers used by many resource providers have advance reservation features, few providers support the use of reservations. In a survey of advance reservation capabilities at several grid sites it is inferred that 50% of the sites which are surveyed did not support reservations at all and that most of the sites that supports reservations required administrator assistance in order to create them. As per the above, only a few sites allowed users to create their own reservations. This kind of advance reservations support is time-consuming and cumbersome. Scheduler-based advance reservations also increase resource usage costs. In many grid environments these costs are measured in service units. Users of advance reservations are typically charged a premium for dedicated access to resources. These premiums can be 20 to 100% above normal costs. Furthermore, users are often forced to pay for the complete reservation, though they are not able to use it all (e.g., if there is a failure that causes the application to abort, or if the actual runtime of the application is shorter than predicted).

An alternative to scheduler-based advance reservations is the use of probabilistic advance reservations. In this method reservations are made based on statistical estimates of queue times which allow jobs to be submitted with a high probability of starting some time before the desired reservation begins. This allows "virtual reservations" to be created by adjusting the runtime of the job to cover both the time between the submission of the job and the desired reservation start time and the duration of the reservation itself. Unlike scheduler-based reservations, probabilistic reservations do not require special support from resource providers. However, probabilistic reservations are not guaranteed because the actual queue delay may exceed the predicted delay and the final cost of a probabilistic reservation is difficult to predict because the actual runtime of the reservation job may exceed the desired reservation time.

1.1. Related Work

Jobs A scheduling strategy on load balancing of VM resources based on genetic algorithm has been proposed (Gu *et al*., 2012). Based on historical data and current system state using genetic algorithm, this strategy computes further on the influence it will have on the system after the deployment of the required VM resources and then selects the least-affective solution, by which it obtains the best load balancing and reduces or avoids dynamic migration. Simultaneously, this system also brings in variation rate to describe the load variation of system VMs and it also bring in average load distance to measure the overall load balancing effect of the algorithm. The disadvantages of the proposed system are wastage of resource when the resources are not distributed properly and Subscribers holds huge dynamic heterogeneity and platform irrelevance whereas the advantages are efficiently and dynamic management of resources so as to meet the requirements of subscriber's problems getting solved with full utilization of service in Cloud computing dynamic environment.

Computer system performance depends on load balancing which should concerns about grid topology, communication delay, negotiation protocol and workload. The interactions and interdependences between these above factors and their relationship with the selected load balancing algorithms are analyzed over here (Sharma and Sharma, 2012). Necessary issues are considered and thoroughly examined through the systematic self-examination and the comparison of two load balancing algorithms, a static and a dynamic one. The static load balancing algorithm is the well-known deterministic Round-Robin, whereas the dynamic load balancing algorithm has been developed for the needs of author's research. They implemented their experiment in a flexible simulation framework. Suitable metrics are formulated so that their combined examination reveals the doings of the system in terms of performance. Precision of the system's state information is always balanced by the simplicity of the negotiation protocol. The disadvantages of existing system are it does not utilize any special selection policy as the tasks are generated and sequentially dispatched; the mixture of processing time is the elapsed time between the arrival and the completion of the task at the processor takes more time will lead to higher delay; degradation of performance may occur when high information policy complexity is combined with important communication overheads whereas the advantage is proposed algorithm efficiency can be enhanced when intense workload is adequately combined with increased delay.

Grid is a dynamic environment, where the resources may join or leave the environment at any time and the jobs also arrives at different intervals of time. To obtain the demands and requirements of the dynamic environment, to minimize the makespan and to maximize the resource utilization an effective grid scheduling technique is needed (Kamalam and Bhaskaran, 2012). We propose grid architecture as a collection of clusters with multiple worker nodes in each cluster. Here proposed a new scheduling algorithm Novel Adaptive Decentralized Job Scheduling Algorithm (NADJSA) that applies both Divisible Load Theory (DLT) and Least Cost Method (LCM) and also considers the user demands. The proposed Novel Adaptive Decentralized Job Scheduling Algorithm is compared with the Decentralized Hybrid Job Scheduling Algorithm. The proposed Novel Adaptive Decentralized Job Scheduling Algorithm minimizes the makespan, improves the resource utilization and satisfies the user demands and well suits for the grid environment.

The issues associated are technical difficulties in implanting real time cloud whereas the advantages are necessary multiplexing to achieve elasticity and the illusion of infinite capacity requires each of these resources to be virtualized to hide the implementation of how they are multiplexed and shared and SaaS provider can devolve some of its problems to the Cloud Computing provider.

The Grid Scheduler must select proper resources for executing the tasks with less response time. There are various reasons such as network failure, resource conditions overloaded, or unavailability of required software components for execution failure. So, fault-tolerant systems should be able to identify and handle failures and support reliable execution in the presence of failures. Therefore the integration of fault tolerance measures and communication time with scheduling gains much importance (Keerthika and Kasthuri, 2012). In this study, a new fault tolerance based scheduling approach Fault Tolerant Min-Min (FTMM) for scheduling statically available meta tasks is proposed wherein failure rate and the fitness value are calculated. The main objective of this study is to design a new scheduling algorithm that reduces the makespan which is the total time taken to complete a set of jobs. Also, the idle time of the resources should be less which assures that no resources are kept idle for a long time. It also ensures that fault tolerant measures are satisfied. The tasks are scheduled after the fault rate of all the resources is calculated. The proposed algorithm considers both system performance and user satisfaction. Hence, most of the jobs are completed within their expected completion time with minimum number of failures.

Cloud System job scheduling is one of the essential functionality performed in all the computing environments. In order to increase the efficiency of working cloud environments, job scheduling is a task that is performed in order to gain maximum profit. Here (Ambike *et al*., 2012), they proposed a system for scheduling the multiple requests from users. All users are classified and authenticated into two types namely service-uploading and downloading by an web application. Multiple requests are processed by utilizing non-pre-emptive priority algorithm. The Cloud Service Provider (CSP) main motive is to provide fast services to the multiple requests. On this study they presented a corresponding strategy and algorithm to gain optimistic value of service considering the goals of users and service providers for Quality of Service (QoS). Resources are utilized in a transient manner. The disadvantage of proposed system is decentralized scheduling has high implementation complexity therefore most of the work is done on centralized schedulers whereas the advantage is that multiple user requests are processed by the use of non-pre-emptive priority algorithm with utilization of resources is done in a very transient manner.

The distinctiveness of Particle Swarm Optimization algorithm (PSO) is that it is capable of solving large-scale combination optimization problem that are easy to fall into the search speed slowly and partially the most superior with global fast convergence of simulated annealing algorithm is utilized to combine particle swarm optimization algorithm in each iteration that enhances the convergence rate and improves the efficiency. Zhan and Huo (2012) presented an improved particle swarm optimization algorithm in resources scheduling strategy of the cloud computing. It also can reduce the average running time of task and raises the rate availability of resources. The disadvantage of proposed system is that strong randomness of these algorithms are easy to sink into defects of local optima and low convergence rate when solving large scale optimization problem whereas the advantages are PSO can solve the large-scale combination optimization problem with the average search speed and proposed algorithm in each iteration that enhances the convergence rate and improves the efficiency.

Cloud computing must be advanced to focus on resource utilization and resource management as they are one of the predominant challenges in cloud. Considering the time of processing, utilization of resource based on CPU usage, throughput and memory usage, the cloud environment with the service node to control all clients request that could provide maximum service to all clients. Resource scheduling and tasks separately involves more waiting time and response time. Linear Scheduling for Tasks and Resources (LSTR) is a scheduling algorithm (Abirami and Ramanathan, 2012) that performs tasks and resources scheduling. The disadvantages are First In First Out (FIFO) scheduling is used by the master node to distribute resources to the waiting tasks and virtualization deals with the existence of the resources that are not physical whereas the advantages are resource allocation is made based on the selection criteria which will improve the efficiency of the cloud environment and the manager of memory is responsible for allocating memory resources to the clients.

Generally, resources scheduling strategy is the key technology in cloud computing. Zhu *et al*. (2012) proposed a new business calculation mode in cloud computing. They performed study of cloud computing system structure and the mode of operation with the key research for cloud computing as the process of the work scheduling and resource allocation problems based on ant colony algorithm. Analysis and design of the specific implementation for cloud resources scheduling is also described. The issue is that resource scheduling is a crucial question of distribution and in cluster calculation it determines the user task execution efficiency whereas the advantages are cloud computing platform is a strong network of collaborative work and it’s connected with a lot of computing resources and services operating resources.

Cloud computing is a rising technology and it lets users to pay as you need and posses very good performance. Cloud computing is a heterogeneous system as well and it contains large amount of application data. It is acknowledged that optimizing the transferring and processing time is crucial to an application program, during the process of scheduling some intensive data or computing an intensive application. In this study (Guo *et al*., 2012) in order to minimize the cost of the processing we formulate a model for task scheduling and propose a Particle Swarm Optimization (PSO) algorithm which is based on small position value rule. The PSO algorithm embedded in crossover and mutation and in the local research converges and runs faster. The issue is that efficient scheduling of all the application tasks and data are the most important problem whereas the advantages are minimizing the processing cost by formulating a model for task scheduling and proposed Particle Swarm Optimization (PSO) algorithm which is based on small position value rule.

Existing solutions to task scheduling problems are unsuitable for Cloud computing because they only focus on a specific purpose like the minimization of execution time or workload and do not use characteristics of Cloud computing for task scheduling. A task scheduler in Cloud computing has to satisfy cloud users with the agreed QoS and improve profits of cloud providers. In order to solve task scheduling problems in Cloud computing, this study (Jang *et al*., 2012) proposes a task scheduling model based on the genetic algorithm. In the proposed model, the task scheduler calls the GA scheduling function every task scheduling cycle. This function creates a set of task schedules and evaluates the quality of each task schedule with user satisfaction and virtual machine availability and the function iterates genetic operations to make an optimal task schedule. Issues are task scheduler in Cloud computing doesn't satisfy cloud users with the agreed QoS and improve profits of cloud providers whereas the advantage is that the task scheduler of this scheduling model calls the GA scheduling function to make task schedules based on information of tasks. The function iterates reproducing populations to output the best task schedule.

1.2. Grid Workflow Scheduler

The proposed workflow scheduling algorithm utilizes the Grid-Framework model with regard to grid networking and resource organization. Grid-Framework aggregates distributed resource brokering and allocation services as part of a cooperative resource sharing environment. The Grid-Framework, $G_F = \{R_1, R_2,\ldots,R_n\}$, consists of a number of sites, n, with each site contributing its resource to the framework. Every site in the framework has its own resource description R_i which contains the definition of the resource that it is willing to contribute. R_i, can include information about the CPU architecture, memory size, number of processors, operating system type, secondary storage size.

In this study, $R_i = \{p_i, x_i, \mu_i, \varnothing_i\}$, which includes the number of processors p_i, processor architecture x_i, their speed μ_i and installed operating system type $\varnothing_i$. Resource brokering, indexing and allocation in Grid-Framework are facilitated by a Resource Management System (RMS) known as Grid-Framework Model (GFM). **Figure 1** shows an example Grid-Framework resource sharing model consisting of Internet-wide distributed parallel resources. Every contributing site maintains its own service which is composed of 3 software entities: Grid Resource Manager (GRM), Local Resource Management System (LRMS) and Distributed Information Manager (DIM) or Grid Peer. Here, we consider the scientific workflow applications as the case study for the proposed scheduling approach. A Scientific workflow application can modeled as a Directed Acyclic Graph (DAG), where the tasks in the workflow are represented as nodes in the graph and the dependencies among the tasks are represented as the directed arcs among the nodes.

We focus on scheduling of workflow application, which consists of a collection of tasks. Our approach supports allocation of different tasks in a workflow across multiple sites in the Grid-Framework (**Fig. 2**), if the total number of processors needed for executing all the tasks in a workflow are not available within a single Grid site. In our application model, each task needs availability of only one processor within a Grid site. Thus the resource claim object for a task encapsulates request for a single processor, i.e. the requirement of the number of processors available is 1. In case, at any given instance of time, if no resource ticket is able to offer single processor as requested by a resource claim object then the claim object is stored in the coordination spaced until one of the Grid site publishes a resource ticket offering one available processor. Sites of grid publish resource tickets after a certain interval of time. Algorithms for (i) task scheduling; (ii) resource provisioning and (iii) resource coordination is given in paper (Rahman *et al*., 2010).

The grid system consists of geographically dispersed computational sites having different administrative polices and heterogeneous resources. Any computational node may employ one or multiple fault-tolerance mechanisms for more reliable computation. Here, we consider the following four fault-tolerance mechanisms:

- Job Retry (JRT) mechanism: The JRT mechanism is the simplest fault-tolerance technique, which will re-execute the failed job from the beginning on the same computational node
- Job migration/Job Migration without checkpointing (JMG) mechanism: The JMG mechanism will move the failed job to another computational node and re-execute the job from the beginning on the latter computational node
- Job migration with Checkpointing (JCP) mechanism: The JCP mechanism will record the state of the job periodically at rum time. If the job fails, it is moved to another computational node and resumed the execution from the last checkpoint
- Job Replication (JRP) mechanism: The JRP mechanism replicates a job to multiple computational nodes such that the job has higher success rate. If one of those replicas has already completed, then all other replicas should stop their execution to save the computing power

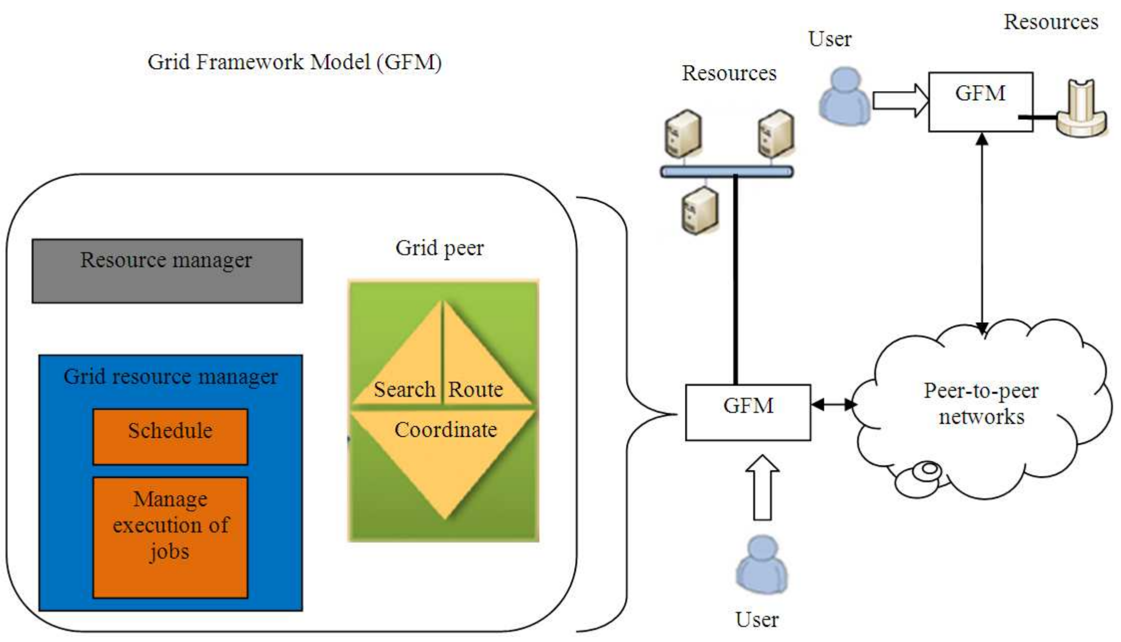

Fig. 1. Grid Framework Model (GFM)

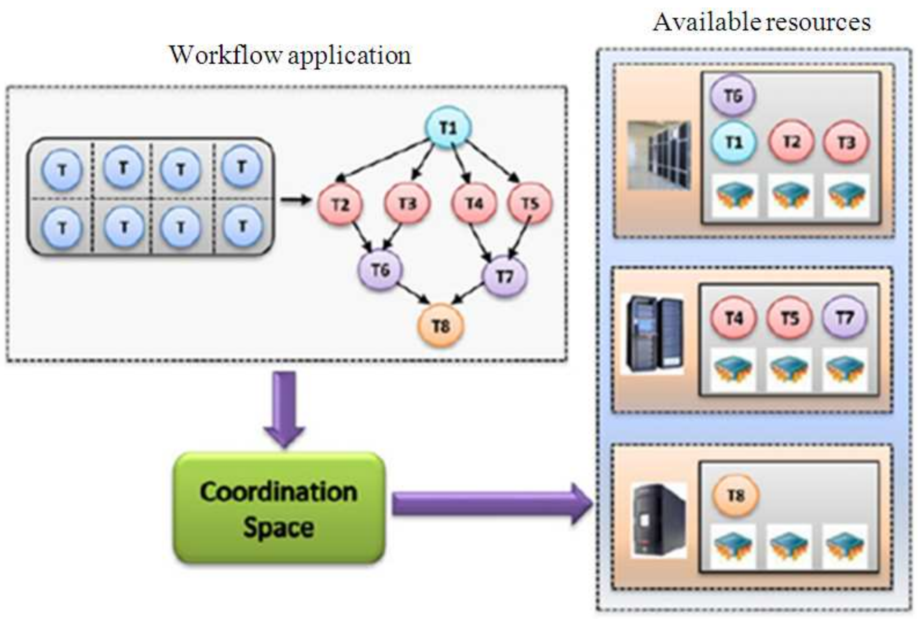

Fig. 2. Multi-site allocation of workflow tasks

In the grid system, each computational site supports one of the following three mechanisms: JRT, JMG and JCP. As for the supporting of JRP, the scheduler will allocate multiple computational sites to execute a certain job concurrently. Furthermore, the scheduler can execute a certain job by any combination of these four different fault-tolerance mechanisms. For instance, a job may be executed concurrently in a node supporting JRT as well as a node supporting JCP, resulting in that JRP is also applied to the job in effect.

1.3. The Glowworm Swarm Optimization (GSO) Algorithm

In GSO, a swarm of agents are initially randomly distributed in the search space. Agents are modeled after glowworms and will be called glowworms in the following of this study. Accordingly, they carry a luminescent quantity called luciferin along with them. The glowworms emit a light whose intensity is proportional to the associated luciferin and interact with other agents within a variable neighborhood. It starts by placing a population of n glowworms randomly in the search space so that they are well dispersed. In the beginning, all the glowworms contain an equal quantity of luciferin. All iteration consists of a luciferin-update phase followed by a movement phase based on a transition rule. The following is the load balancing algorithm that utilizes GSO for effective scheduling:

1. Initialize the number of virtual machines VM= $\{vm_1,\ldots\ldots,vm_n$ n number of resources and T= $\{t_1,\ldots\ldots,t_n\}$ t is the n number of tasks.
2. Calculate the processing time $t_{i,j}$ to process task t on resource i is known; and T is m×n matrix such that:

$$T = \begin{bmatrix} t_{11}\, t_{12} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m1}\, t_{m2} & \cdots & t_{mn} \end{bmatrix}$$

3. Set number of dimensions = m
4. Set number of glowworms = n
5. Let s be the step size
6. Let $x_i(t)$ be the location of glowworm i at time t
7. deploy agents randomly
8. Define smallest position value (SPV) $S^0 = S^0{}_1, S^0{}_2, S^0{}_3,\ldots.,S^0{}_N$ and apply SPV rule to solve discrete problems at Step 17.
9. Find the optimal resources vector using $R^0 = R^0{}_1, R^0{}_2, R^0{}_3,\ldots.,R^0{}_N$:

$$R_i^k = (S_i^k \bmod m) + 1$$

10. Calculate the $E(T_j^i)$ represents the expected execution time for Job i running in Node j at step 20.
11. Set maximum iteration number = iter_max
12. Set t =1
13. while ($t \le$ iter_max) do
14. for i =1 to n do $l_i(0) = l_o$
15. $r_d^i(0) = r_0$
16. $N_i(t) = \{j: d_{ij}(t) < r_d^i(t); l_i(t); l_i(t) < l_j(t)\}$
17. j = select$_{glowworm}(\vec{p})$
18. $x_i(t+1) = x_i(t) + s\left(\frac{x_j(t)-x_i(t)}{\left\|x_j(t)-x_i(t)\right\|}\right)$
19. $r_d^i(t+1) = \min\{\gamma_s, \max\{0, r_d^i(t)+\beta(n_i - |N_i(t)|)\}\}$
20. SPV rule to obtain the discrete permutation, where $S^K_{i,j}$ represents the resource ID to which the task j is assigned.
21. Calculate the $E(T_j^i)$ represents the expected execution time for Job i running in Node j:

$$E(T_j^i) = E_{JRT}(T_j^i)\left(1+\frac{1}{2}P_j^i+\frac{1}{2}P_j^{i^2}+\frac{1}{2}P_j^{i^3}\right)\times\frac{SZ_i}{C_j}$$

where, SZ_i is the size of Job i and C_j is the computing capacity of Node j.

22. If node a to j fails job is migrated to another computational node $E(T_j^i)$ represents the expected execution time for Job i running in Node j, k, q:

$$E_M\left(T_j^i\right) = \left(1-\frac{1}{2}P_j^i\right)\times\frac{SZ_i}{C_j} + P_j^i \times MC_{j,k}^i$$

$$E_M\left(T_k^i\right) = P_j^i \times\left(\left(1-\frac{1}{2}P_k^i\right)\times\frac{SZ_i}{C_k} + P_k^i \times MC_{k,q}^i\right)$$

$$E_M\left(T_q^i\right) = P_j^i \times P_k^i\left(\left(1-\frac{1}{2}P_q^i\right)\times\frac{SZ_i}{C_q}\right)$$

Where:

$$MC_{x,y}^i = \frac{D_i}{Bw_{x,y}}$$

$MC_{x,y}^i$ is the migration cost of the condition that Job i moves from Node x to Node y, D_i is the data size of Job_i, $Bw_{x,y}$ and is the communication bandwidth between Node j and Node j and Node k, where x; $y\in\{j,k,q\}$.

23. If node i to j fails job transient process states to the check pointing server periodically the process to backup node before resuming the unfinished job:

$$E_{cp}(T_j^i) = (1 - P_j^i)\left(\frac{SZ_i}{C_j} + \left(\frac{\frac{SZ_i}{C_j}}{PR}\right) \times OH_j\right)$$

$$\times P_j^i\left(\frac{SZ_i}{2 \times C_j} + \left(\frac{\frac{SZ_i}{2 \times C_j}}{PR}\right) \times OH_j + MC_{j,k}\right)$$

$$RM_j^i(j,k,q) = SZ_i - \left(\left(\frac{\frac{SZ_i}{2 \times C_j}}{PR}\right) \times PR \times C_j\right)$$

$$E_{cp}(T_k^i) = P_j^i\left((1 - P_j^i)\left(\frac{RM_j^i(j,k,q)}{C_K}\right) + \left(\frac{\frac{RM_j^i(j,k,q)}{C_K}}{PR}\right) \times OH_K\right)$$

$$+P_k^i\left(\frac{RM_j^i(j,k,q)}{2 \times C_K}\right) + \left(\left(\frac{\frac{RM_j^i(j,k,q)}{2 \times C_K}}{PR}\right) \times OH_K + MC_{k,q}\right)$$

$$RM_k^i(j,k,q) = RM_j^i(j,k,q) - \left(\left(\frac{\frac{RM_j^i(j,k,q)}{C_K}}{PR}\right) \times PR \times C_k\right)$$

$$E_{cp}(T_q^i) = P_j^i \times P_k^i\left((1 - P_j^i)\left(\frac{RM_k^i(j,k,q)}{C_q}\right) + \left(\frac{\frac{RM_k^i(j,k,q)}{C_q}}{PR}\right) \times OH_q\right)$$

$$+P_q^i\left(\frac{RM_k^i(j,k,q)}{2 \times C_q}\right) + \left(\left(\frac{\frac{RM_k^i(j,k,q)}{2 \times C_q}}{PR}\right) \times OH_K\right)$$

where, $RM_k^i(j,k,q)$ is the remaining job size for Job i to be executed when a failure occurs in Node x. OH_x is the overhead of performing one check pointing operation for Node x.

24. Let the set RP_i consists of those nodes that will execute Job i independently. Assume Job i starts to be executed in Node j at time s_j^i if Node j belongs to the set RP_i. If Job i is executed successfully, then the job will be finished at time $f_j^i = s_j^i + \frac{SZ_i}{c_j}$. Because the execution of job i in Node j will continue after time f_j^i only if all previous executed replicas fail, the probability that Job i will continue after time f_j^i will be:

$$P_{const_i}\left(f_j^i\right) = \prod_{\substack{w \in RP_i \\ f_w^i \leq f_j^i}} P_w^i$$

25. Let the execution time of each replica is broken into multiple pieces by f_j^i, where $j \in RP_i$. Each piece has an execution probability and its expected execution time is equal to multiplying the continuation probability at the beginning of a piece by the execution time of executing Job i in Node j is calculated as follows:

$$E\left(T_j^i\right) = E_{JRP}\left(T_j^i\right)$$
$$= P_{start}\left(s_j^i\right) \times \left(-s_j^i\right) + \sum_{\substack{nx, j \in RP_i \\ s_j^i \leq f_x^i < f_y^i \leq f_j^i \\ \neg\exists f_z^i . f_x^i < f_z^i \leq f_y^i}} P_{const_i}\left(f_x^i\right) \times (f_y^i - f_x^i)$$

26. Result of the execution time for Job i running in Node j, k, q
27. If $\left(S_{i,j}^K \leq E_M\left(T_j^i\right) \leq E_{cp}\left(T_j^i\right)\right)$ then
    Go to step 20 and result of execution time.

Else
If $\left(E_M\left(T_j^i\right) \leq S_{i,j}^K \leq E_{cp}\left(T_j^i\right)\right)$ then
Go to step 22 and result of execution time
Else
If $\left(E_{cp}\left(T_j^i\right) \leq S_{i,j}^K \leq E_M\left(T_j^i\right)\right)$ then
Go to step 23 and result of execution time
25. For each glowworm i do:

$$\ell_i(t) = (\ell - \rho)\ell_i(t-1) + \gamma J(x_i(t))$$

26. for each glowworm $j \in N_i(t)$do:

$$p_{ij}(t) = \frac{\ell_j(t) - \ell_i(t)}{\sum_{k \in N_i(t)} \ell_k(t) - \ell_i(t)}$$

```
28. end if
29. end if
30. Evaluate new solutions and update light intensity.
30. end for j
31. end for i
32. t = t+1
33. Rank the glowworms and find the current global best
    and update the iteration parameter.
34. Repeat the above phases until the termination
    condition is met.
```

1.4. Weighted GSO

Weighted GSO is also similar to General GSO but if any glow worm that does not able to find any best solution, the intensity of glow worm i is absorbed and it will be invisible to all other glow worm in the space. Hence weighted GSO surmounts this problem by assigning pre-defined weight parameters w_i to each glow worm improves the efficiency and result.

2. MATERIALS AND METHODS

In this section we have made an attempt to decentralized cooperative workflow scheduling in a dynamically distributed resource sharing environment of Grids. This can be done by using Gridsim. This approach ignored the fact that each grid site may have its own fault-tolerance strategy because each site is itself an autonomous domain. For instance, if a grid site employs the job check-pointing mechanism, each computation node must have the ability of periodical transmission of transient state of the job execution by computational node to the server. When there is a failure of job, it will migrate to another computational node and resume from the last stored checkpoint. A Glow worm Swarm Optimization (GSO) for job scheduling is used to address the problem of heterogeneity in fault-tolerance of computational grid but Weighted GSO that overcomes the position update imperfections of general GSO in a more efficient manner shown during comparison analysis.

3. RESULTS

The following are the graphical results of our implemented systems namely GSO and Modified GSO (MGSO) and the parameters considered for the comparison of these methods are namely:

- Response time
- Co-ordination delay and
- Makespan

4. DISCUSSION

Response time for a task is the delay between the submission time and the arrival time of execution output which is shown in **Fig. 3**. Effectively, the response time includes the latencies for coordination and the CPU time. In **Fig. 3**, Number of tasks ranging from 50 to 500 is taken along x-axis and average response time per task (in seconds) is taken along y-axis ranging from 0 to 500. It can be inferred from the graph that response time of MGSO is lesser than GSO which shows MGSO is more responsive than GSO.

The metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell, (ii) waiting time till a resource ticket matches with the claim and (iii) notification delay from coordination service to the relevant GFM which is shown in **Fig. 4** on which number of tasks ranging from 50 to 500 is taken along x-axis and average coordination delay time per task (in seconds) is taken along y-axis ranging from 0 to 250. It can be inferred from the graph that coordination delay time of MGSO is lesser than GSO which shows MGSO is more coordinating than GSO.

Makespan is measured as the response time of a whole workflow, which equals the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow which is shown in **Fig. 5**. Note that, these measurements (except makespan) are collected by averaging the values obtained for each task in the system. The measurement of makespan is taken by averaging over all the workflows in the system. In **Fig. 4**, Number of tasks ranging from 50 to 500 is taken along x-axis and average makespan workflow is taken along y-axis ranging from 0 to 5000. It can be inferred from the graph that makespan of MGSO is lesser than GSO which shows MGSO is more quicker in completion of workflow than GSO.

From all the above graphs we can conclude that MGSO is better than GSO in terms of response time, coordination delay and makespan.

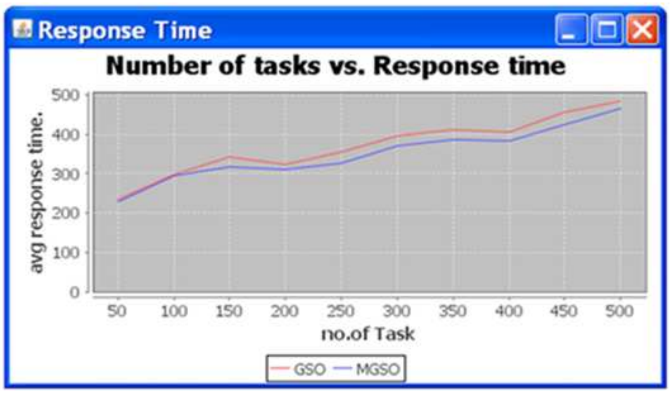

Fig. 3. Response time graph

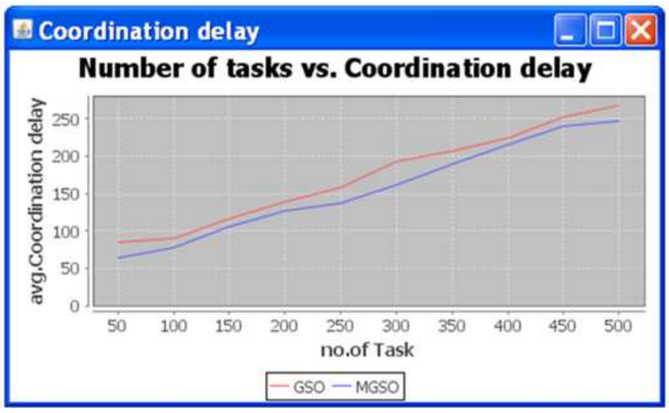

Fig. 4. Coordination delay graph

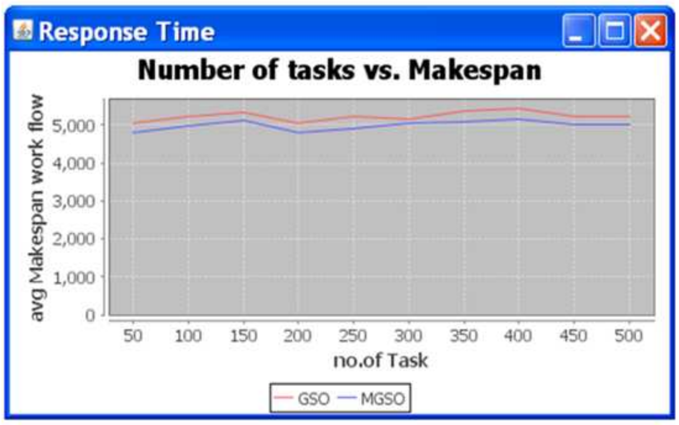

Fig. 5. Makespan graph

5. CONCLUSION

We have implemented a decentralized and cooperative scheduling technique for workflow applications with a GA-based job scheduling strategy for a large-scale computational grid. We considered the computational grid in which each computational site supports one or two of four kinds of fault-tolerance mechanisms, including job migration, job retry, the job migration with checkpointing and the job replication mechanisms. The scheduler will decide which kinds of fault-tolerance mechanisms will be applied to each individual job for more reliable computation and shorter makespan. To induce effective scheduling we utilized Glowworm Swarm Optimization that is even capable of handling discontinuities in the objective function in finding the best scheduling method. In future, we intend to address the resource failure and fault tolerance issues into our scheduling technique. Future work in this direction would involve a thorough analytical study of the effect of various parameters on algorithm performance, aimed primarily toward providing an analytical justification to the conclusions reached by experimentation.

6. REFERENCES

Abirami, S.P. and S. Ramanathan, 2012. Linear scheduling strategy for resource allocation in cloud environment. Int. J. Cloud Comput. Services Arch., 2: 9-17.

Ambike, S., D. Bhansali, J. Kshirsagar and J. Bansiwal, 2012. An optimistic differentiated job scheduling system for cloud computing. Int. J. Eng. Res. Applic., 2: 1212-1214.

Gu, J., J. Hu, T. Zhao and G. Sun, 2012. A new resource scheduling strategy based on genetic algorithm in cloud computing environment. J. Comput., 7: 42-52.

Guo, L., S. Zhao, S. Shen and C. Jiang, 2012. Task scheduling optimization in cloud computing based on heuristic algorithm. J. Netw., 7: 547-553. DOI: 10.4304/jnw.7.3.547-553

Jang, S.H., T.Y. Kim, J.K. Kim and J.S. Lee, 2012. The study of genetic algorithm-based task scheduling for cloud computing. Int. J. Control Automat., 5: 157-162.

Kamalam, G.K. and V.M. Bhaskaran, 2012. Novel adaptive job scheduling algorithm on heterogeneous grid resources. Am. J. Applied Sci., 9: 1294-1299. DOI: 10.3844/ajassp.2012.1294.1299

Keerthika, P. and N. Kasthuri, 2012. An efficient fault tolerant scheduling approach for computational grid. Am. J. Applied Sci., 9: 2046-2051. DOI: 10.3844/ajassp.2012.2046.2051

Rahman, M., R. Ranjan and R. Buyya, 2010. Cooperative and decentralized workflow scheduling in global grids. Future Generat. Comput. Syst., 26: 753-768. DOI: 10.1016/j.future.2009.07.002

Sharma, M. and P. Sharma, 2012. Performance evaluation of adaptive virtual machine load balancing algorithm. Int. J. Adv. Comput. Sci. Applic., 3: 86-88.

Zhan, S. and H. Huo, 2012. Improved PSO-based task scheduling algorithm in cloud computing. J. Inform. Comput. Sci., 9: 3821-3829.

Zhu, L., Q. Li and L. He, 2012. Study on cloud computing resource scheduling strategy based on the ant colony optimization algorithm. Int. J. Comput. Sci., 9: 54-58.