Abstract

The objective of this research is to develop a task scheduling algorithm for grid services with improved resource utilization, fault tolerance capability and to avoid deadlocks. The algorithm simulated using GridSim toolkit, which is used to measure the effectiveness of scheduling and related algorithms in grid environments with reduced complexity than the real implementation. Two advanced methods, Two Phase Commit Protocol with novel back filling Technique (TPCNBF) and Two Phase Commit Protocol with novel back filling technique and task migration (TPCNBFM) were developed to schedule the tasks. The backfilling technique is used to improve resource utilization and fault tolerance in job scheduling. Two phase commit protocol prevents deadlocks in task scheduling. Fault tolerance in task scheduling is improved by task migration to protect tasks from any resource failure. TPCNBF and TPCNBFM methods give higher performance improvement than the current method called Cost Driven Work Flow Scheduling (CDWFS). In this method, the scheduling is based on deadline and the resource status. The experimental results showed that the TPCNBF and TPCNBFM have less makespan, cost, high success rate and throughput. The new algorithm can be used to schedule computing intensive tasks in grid services like Astronomy, high energy physics with deadlock avoidance, fault tolerance and improved resource utilization.

Keywords: Back Filling, Deadlock Avoidance, Fault Tolerance, Grid Computing, Migration, Two Phase Commit Protocol

1. Introduction

Grid computing, a turning point in distributed computing to manage large-scale resource distribution and problem resolving in dynamic, inter institutional virtual organizations. With the help of fast network, grid computing allows people to work together and allocate facilities among community, institutional, and geographic boundaries without losing local authority. Atomic tasks are combined together to form a grid workflow and process in a systematic order using distributed resources to obtain an efficient solution within the deadline. Huge scientific data or compute intensive applications like Astronomy, high energy physics, medical image processing are using Directed Acyclic Graph (DAG) to represent its workflow. GridAnt and DAGMan are two distinguished grid workflow management schemes developed, based on DAG and used in Globus and Condor projects. In grid computing DAG is used to represent the task scheduling of an application and based on this graph, atomic tasks are scheduled in suitable resources in a grid environment. The problem can be defined as N tasks are to be mapped on M computing resources based on the following conditions, the implementation precedence is among two inter-dependent tasks, only one task can be enforced on a host and task processing is non-preemptive. Such a scheduling issue was uncovered by NP-complete. Grid resources are heterogeneous in nature so that competent and successful scheduling becomes important. Great difficulties are there to reduce job execution time with high resource usage.

Grid architecture consists of three entities as user, grid scheduler (resource broker) that split and for-
Novel Backfilling Technique with Deadlock Avoidance and Migration for Grid workflow Scheduling

Towards the tasks to the local scheduler for execution and grid information service which provides resource information. Grid scheduling consists of three steps, collection of resource information, sending the job to resource and execution. Task scheduling is defined as the way of allocation of processes to available resources. Scheduling consists of QoS parameters like Resource utilization, Throughput, Turnaround time, Waiting time and Response time. The common optimization condition in scheduling is minimum execution time. Flow time is the total completion time of tasks and it measures the QoS of the grid system. Flow time is reduced when jobs are handled in an ascending order of processing time on a specific grid resource. Due to the cost-effective aspect, resource providers are concerned with maximizing the resource usage. In grid computing a task is submitted to the best potential machine where the execution time is minimum. Due to the dynamic nature of a grid, computation time required for a scheduler is also a significant condition to choose a proper scheduling method. In grid scheduling, it is difficult to get an efficient solution always due to the dynamic environment, but minimum time solutions are accepted.

The objective of a scheduling algorithm is to share the load among the resources and improve the utilization with reduced task execution time. The job scheduler is accountable for effective utilization of resources while scheduling a job. The scheduler has to ensure that each job is allocated with enough resources. The scheduling of a dependent task is more challenging than independent task scheduling. The scheduling of both dependent and independent tasks with QoS requirements of users are the challenging issues in grid computing.

Most of the QoS based task scheduling techniques use QoS metric to calculate the priority of a task and high priority task is allocated to a resource first. The allocated task, based on priority, may cause low resource usage. Deadlock occurrence while scheduling tasks with dependencies are not considered in Priority based scheduling algorithms. Resource degradation due to overloaded resource may happen due to the lack of an overload consideration in the existing algorithms.

The primary aim of this research is to improve the QoS in task scheduling by introducing a new Backfilling technique with fault tolerance mechanism, Overload detection and avoidance mechanism and Deadlock avoidance by a two phase commit protocol. We introduce these techniques one by one and evaluate the usage of grid resources with other performance parameters. These methods not only improve utilization, but also reduce the processing cost to both user and grid provider due to maximum resource utilization.

The basic batch scheduling algorithm is First-Come-First-Serve (FCFS)\(^2\). In this algorithm, jobs are processed, based on arrival time. If there are enough resources available to process a job, the resources are allocated to process the job. Otherwise, the first task in the queue has to wait for resource availability. This leads to wastage of resource power as the resource sit idle to free, enough resource for the first job in the queue.

Quinn Snell et al.\(^3\) introduced a new backfilling method. Backfilling is an optimization method for space sharing to improve resource usage without much change in the First Come First Serve order. This method permits smaller jobs to override priority and execute in unused resources, which can't be used by First Come First Serve order at that point of time. This method also avoids starvation in job queue. Backfilling algorithms are classified into two:

- Backfilling with Conservative Method
- Backfilling with Aggressive Method

Backfilling algorithm based on conservative method was introduced by Ahua et al.\(^4\). Jobs are scheduled based on their arrival time, if there are enough free resources available for scheduling, otherwise smaller jobs with later arrival time are scheduled as per this method. It provides reservation for all jobs and limits the slowdown. Aggressive method is an alternative method in which reservation permitted for the first job in a queue to preempt other jobs. During user job submission, guarantee of response time is not known in this algorithm.

Backfilling Gang Scheduling algorithm introduced by Moreiraz et al.\(^5\). The combined optimization of Backfilling and Gang scheduling is called Backfilling Gang scheduling algorithm. Gang scheduling algorithm creates virtual machines for backfilling jobs. The new approach ensures better performance than individual gang scheduling or backfilling.

Migration Gang Scheduling proposed by Moreiraz et al.\(^6\) migrate a job to free resources available in a row for the job execution. A job can migrate from a starting row to a destination row by means of two possible cases. In the first case, suppose two jobs A, B are in the same column, but in different rows as row 1 and row 2, then job B is shifted to some other column in row 2 and job A is shifted...
to row 2. In the second case, job A is immediately shifted to second row.

Job migration to any other rows with the availability of unused resources was introduced by Moreira et al. If a task is unable to move any other row due to an existing job in that set of processors, the existing job in the new processors to be shifted to a different set of processors for the migration of job already initiated. A job is to be assigned in similar set of processors to avoid ping-pong of jobs and can be in different rows of a matrix.

Job Kill Based EASY Backfilling algorithm introduced by Xiaogang Qiu et al. permits job to process in background resources, which is not suitable for backfilling as per EASY algorithm. Better performance can be achieved, if there is an opportunity to run the job in free foreground resources. In Reservation based EASY backfilling algorithm, killing of jobs are not permitted while scheduling. After allocation of a job to background resources, the processing cannot be transferable to a different set of processors. In this free foreground resources are permitted to execute the tasks.

Berger model scheduling algorithm was introduced by Booming Xu et al. to meet the QoS constraints. This algorithm schedules a task based on bi-fairness constraint. In this algorithm, the user task is categorized on QoS measures such as size, bandwidth and completion time. The tasks are mapped with grid resources. If there is a match, the tasks execute within the completion time otherwise, change completion time and map it again with grid resources. After adjusting the completion time, the task can complete within completion time. After adjusting the completion time, if the tasks cannot map with system resources, then the tasks remain incomplete.

Abrishami and Naghibzadeh proposed a workflow scheduling algorithm based on critical path heuristics, which try to schedule the critical tasks first. Scheduling of partial critical path recursively, to last scheduled task is done in this algorithm. Efficient usage of storage is an advantage of this method. Oversubscribed resources create deadlocks and scheduler has to avoid deadlocks for maximizing performance. Performance can be measured by makespan and throughput.

Integrated management of heterogeneous workloads introduced by David Carrera et al. It uses relative performance functions to compare both workloads of transactional applications and long-running jobs. This method performs well in heterogeneous workloads compared to similar scheduling algorithms. Different types of application workload in a data center can’t be allocated using this and high re-execution time of a job, are the two disadvantages of this scheduling algorithm.

Multi criteria and multi decision, priority driven scheduling algorithm introduced, by Shamsollah Ghanbari and Mohedh Othman. Object level, attribute level and alternate level are different stages and priority is set by job resource ratio in this algorithm. High throughput and reduced completion time are two advantages. The disadvantages are that the finish time cannot be predicted and response time is high.

Dynamic critical path based adaptive workflow scheduling algorithm for grids (DCP-G) was proposed by Mustafizur Rahman et al. After critical path calculation of the workflow task graph at each step, tasks are allocated to grid resources dynamically. The schedule is computed only once for a task graph. DCP-G considers all available resources to schedule a task on critical path, but DCP considers resource allocation together with parent and child tasks.

Adaptive Hybrid Heuristic scheduling approach for clouds was for task to service scheduling using GA with minimum execution cost as per user’s budget, deadline, service and bandwidth. Multi objective optimization is not introduced to introduce QoS parameters like reliability of service in task to service scheduling.

Higher priority task be scheduled first and a task be finished at the earliest were the basic ideas of QoS scheduling algorithm of Xiaoian et al. Special attributes are used to calculate priority of a task and sort tasks based on priority. After calculating the completion time of jobs in available resources, schedule the tasks into the resources in which the completion time is minimum. QoS parameters like price, scalability, throughput, resource usage, success rate, reliability and availability are not used in this algorithm.

Measure of uneven utilization of a resource is called skewness and introduced by Zhen Xiao et al. Minimization of skewness improves overall resource utilization in multidimensional resource constraints. Load prediction algorithm predicts future resource usage of applications without examining the resources.

A priority based scheduling algorithm for consolidation of parallel workload in cloud environment introduced by Xiaoeng Liu, et al. In this computing capacity is divided into foreground and background tiers. Conservative migration and consolidation supported backfilling (CMCBF)
and Aggressive Migration and Consolidation supported Backfilling (AMCBF) are two priority methods used in this algorithms for consolidation of parallel workloads. Foreground resources are used to process a task, whenever a resource is free or the job demands the usage of a foreground resource. Run time monitoring of data is used to collect the usage of foreground resources. Communication cost reduction by scheduling jobs to nearby resources, is an issue in this algorithm.

The objectives of the proposed scheduling algorithm are summarized as follows. 1. While handling data dependencies in tasks, the occurrence of deadlock is avoided by two-phase commit protocol. 2. The improved backfilling technique is to increase resource usage as well as fault tolerance. Backfilling also avoids failure of task completion to obtain fault tolerance. 3. Execution of a task will be aborted whenever a task misses its deadline and migration technique is used to complete the aborted task in a new resource.

2. Fault Tolerance and Deadlock Aware Scheduling

Grid Computing is next generation distributed computing, which enables flexible sharing of resources among many computational users to solve complex problems. Due to the increased users in a grid environment, resource allocation and scheduling should be carried out efficiently. The allocation of user submitted tasks in the grid environment should be optimized to avoid deadlocks. If the resources are not allocated properly, they are hindered by running tasks before deadline. The best user friendly grid environment can be obtained by locating resources in correct locations and allocating the tasks into the resources efficiently.

2.1 Finding a Critical Path for the Workflow

The partial critical path algorithm is used to find out the flow of dependable tasks and where the tasks need to be executed. The critical path is defined as the path between the starting node and the ending node. The scheduling is done based on user defined deadline of a workflow in which the dependable tasks finish their execution. First assign the tasks belonging to the critical path to ensure the earliest completion before their deadline. To complete workflow scheduling without any interruption, two dummy nodes are added. Schedule the tasks based on user deadline from the exit node. The task to be executed after the current task is decided using a path assigning algorithm as explained in Algorithm 1.

Algorithm 1, Assign Path

1. Optimized ← Ø
2. j ←initial job in the route
3. while (j not Ø) do
   a. R ← adjacent resource with less speed ε R
   b. If (R = Ø or allotment of j to R not possible) then
      i. j ← allot older job on route and execute while loop
   c. end if
   d. if (j is the final job in the route) then
      i. if (present allocation cheaper than Optimized) then
      ii. fix present allocation as Optimized end if
      iii. j ← old task in the route
   e. else
      i. j ← next task in the route
   f. end if
4. end while
5. if (Optimized is Ø) then
   a. fix sub deadline (j) = EST (j) + MET (j) for the entire jobs j in the route
6. else
   a. fix EST and sub deadline as per Optimized for all jobs ε route
7. end if
8. mark as allocation of all tasks on the route completed

2.2 Two Phase Commit Protocol with the Novel Back Filling Technique (TPCNBF)

The deadlock problem involves a circular wait, where one or more transactions wait for resources to available. The deadlock never terminates its execution and resources held by it are not useable to any other process, that resulting on an indefinite wait in processing. A Two Phase Commit protocol (TPC) is a solution to avoid a deadlock in distributed environment as explained in Algorithm 2.

TPC protocol was introduced to safeguard ACID properties in distributed process management with four properties, atomicity, consistency, isolation and durability. A Transaction Manager (TM) manages distributed process management in coordination with Resource Managers (RM) which is managing resources for sub transactions.
In a global distributed process management, several RM's are controlled by a TM. It is also possible to divide a RM among multiple TM for parallel transactions. Usually a distributed process consists of many sub transactions and that are processed in several networked resources. Each sub transaction is a member of the process under the control of a TM. TPC protocol is used to ensure that a process either successfully completed or failed.

When the resources are allocated based on the TPC, resource failure may be due to improper resource allocation. To obviate this problem, the new back filling approach is combined with TPC mechanism to obtain better resource usage in consideration with fault tolerance and deadlock avoidance.

Backfilling is used increase resource utilization by permitting the first task in the queue to reserve unavailable resources by judging the performance of sequential tasks that are available in submission queue. The task in the submission queue are executed, if they are not use the resources reserved by the first task in queue or the tasks in the submission queue complete its processing within the allocated time. Backfilling needs the details of allocated processing time of tasks in the queue. Novel Backfilling based scheduling algorithm (NBF) used in this work effectively partitions the load among nodes. Partitioning resource computing capacity into multiple tiers can improve resource utilization and responsiveness to process parallel jobs in a resource. The existing algorithms divide a resource's computing capacity into two tiers. But proposed algorithm partitions a resource's computing capacity into 4-tiers as low CPU priority, medium CPU priority, high CPU priority and very high CPU priority. Jobs are forwarded to a corresponding dispatcher based on their computing requirements for allocation. The algorithm 3 is triggered when a new job arrives or a job finishes execution.

**Algorithm 2, Two-phase Commit Algorithm**

**Input**: Task (n), Resource (m)

**Output**: The scheduled list

1. For all n ∈ Task(n)
2. Each task sends request to access the resources;
3. Create the EET matrix, MET matrix and Cost matrix;
4. Initialize Mapping list;
5. 
6. Select resource with minimum time and cost CT (i, j) and MET (i, j) for task (i)
7. }
8. Create a new mapping pair (i, j) in the Mapping list;
9. Update MET matrix and Cost;
10. Delete task T_i from task set Task (n);
11. }
12. for the mapping list
13. }
14. Novel Backfilling based scheduling algorithm (NBF) // Schedule task T_j on to resource m_j according to the backfilling mechanism;
15. }

In the scheduling operation, to set and update MET matrix and Cost matrix, processing time and cost of tasks on different resources are reckoned with and search for the least component value for each task in the course of updating MET matrix and Cost matrix. Enter the least value in the matrix and send the task into the selected resource.

The NBF algorithm allocates the tasks into resources based on four categories of computing capacity, low CPU priority, medium CPU priority, high CPU priority and very high CPU priority. Jobs are forwarded to a corresponding dispatcher based on their computing requirements for allocation. The algorithm 3 is triggered when a new job arrives or a job finishes execution.

**Algorithm 3 input job process in NBF**

**Input** current mapping list, M and arriving jobs j

**Output** new mapping list (M') with added jobs

1. Start
2. Insert job (j) to queue (Q);
3. \( N_j <— \) total resources required by j;
4. \( N_{L_{idle}} <— \) total unused Low_CPU resources;
5. \( N_{M_{idle}} <— \) number of nodes with idle of Medium_CPU resource;
6. \( N_{H_{idle}} <— \) number of nodes with idle of High_CPU resource;
7. \( N_{VH_{idle}} <— \) number of nodes with idle of VeryHigh_CPU resource;
8. if \( N_j \leq N_{L_{idle}} \) then
   a. Detach job (j) from queue (Q);
   b. Send (Low_CPU, j);
   c. Return;
9. if \( N_j \leq N_{M_{idle}} \) then

---

Shinu Mathew John and Maluk Mohamed

Indian Journal of Science and Technology | 795
a. Detach job (j) from queue (Q);
b. Send (Medium_CPU, j);
c. Return;

10. If $N_j \leq N_{H_idle}$ then
   a. Detach job (j) from queue (Q);
   b. Send (High_CPU, j);
   c. Return;

11. If $N_j \leq N_{VH_idle}$ then
   a. Detach job (j) from queue (Q);
   b. Send (VeryHigh_CPU, j);
   c. Return;

In the above algorithm, the computing requirements of the input tasks are calculated and categorized based on its type (low, medium, high, very high). After sorting, the jobs are removed from the queue and sent to the corresponding job scheduler. The algorithms below explain the allocation of jobs on the resources based on its computing capacities.

**Algorithm 4, departure of Low_CPU jobs in NBF algorithm**

**Input**, arriving jobs added to Queue (Q)

$J_{High_CPU}$: High_CPU job execution list
$J_{Medium_CPU}$: Medium_CPU job execution list
$J_{Very_High_CPU}$: VeryHigh_CPU job execution list

**Old job schedule map**: M

**Output**, new job schedule map M’

1. Start
2. Take out departure jobs from $J_{Low_CPU}$, $J_{High_CPU}$, $J_{Very_High_CPU}$
3. For every job $j \in$ Queue (Q) do
   a. $N_j \leftarrow$ total resources needed to job j;
   b. $N_{L_idle} \leftarrow$ total idle Low_CPU Resources;
   c. If $N_j < N_{L_idle}$ then
      i. Detach j from Q;
      ii. Send (Low_CPU, j);

The algorithm 4 is used to execute a task in Low CPU resources whenever the numbers of free Low CPU resources are enough or used by tasks reaching later than it, cater the task requirements. Simultaneously, it permits tasks to run in medium CPU Resources, high CPU Resources and very High CPU Resources with those Low CPU Resources to improve node utilization.

**Algorithm 5, departure of Medium_CPU jobs in NBF algorithm**

**Input**, arriving jobs added to Queue (Q)

$J_{Low_CPU}$: Low_CPU, job execution list,
$J_{High_CPU}$: High_CPU, job execution list,
$J_{Very_High_CPU}$: VeryHigh_CPU, job execution list,

**Old job schedule map**: M

**Output**, new job schedule map M’

1. Start
2. Take out departure jobs from $J_{Low_CPU}$, $J_{High_CPU}$, $J_{Very_High_CPU}$
3. For every job $j \in$ Queue (Q) do
   a. $N_j \leftarrow$ total resources needed to job j;
   b. $N_{M_idle} \leftarrow$ total idle Medium_CPU Resource;
   c. If $N_j < N_{M_idle}$ then
      i. Detach j from Q;
      ii. Send (Medium_CPU, j);

The algorithm 5 is used to execute a task in medium CPU resources whenever the numbers of free medium CPU resources are enough or used by tasks reaching later than it, cater the task requirements. Simultaneously, it permits tasks to run in low CPU Resources, high CPU Resources and very High CPU Resources with those medium CPU Resources to improve node utilization.

**Algorithm 6, departure of High_CPU jobs in NBF algorithm**

**Input**, arriving jobs added to Queue (Q)

$J_{Low_CPU}$: Low_CPU, job execution list,
$J_{Very_High_CPU}$: VeryHigh_CPU, job execution list,
$J_{Medium_CPU}$: Medium_CPU, job execution list,

**Old job schedule map**: M

**Output**, new job schedule map M’

1. Start
2. Take out departure jobs from $J_{Low_CPU}$, $J_{Very_High_CPU}$
3. For every job $j \in$ Queue (Q) do
   a. $N_j \leftarrow$ total resources needed to job j;
   b. $N_{H_idle} \leftarrow$ total idle High_CPU Resources;
   c. If $N_j < N_{H_idle}$ then
      i. Detach j from Q;
      ii. Send (High_CPU, j);

The algorithm 6 is used to execute a task in High CPU resources whenever the numbers of free High CPU resources are enough or used by tasks reaching later than it, cater the task requirements. Simultaneously, it permits tasks to work in low CPU Resources, medium CPU Resources and very High CPU Resources with those high CPU Resources to improve node utilization.
Algorithm 7, NBF- departure of a VeryHigh_CPU jobs in NBF algorithm

**Input**, arriving jobs added to Queue (Q)
J_{VeryHigh_cpu} list of jobs running in the High_CPU;
Old job schedule map: M

**Output**, new job schedule map M'

1. Start
2. j ← load initial job j from Q \(\bigcup J_{VeryHigh_cpu}\);
3. while j ≠ null do
   a. N_{VH_idle} ← total idle VeryHigh_CPU Resources;
   b. if N_{j} > N_{VH_idle} Then
      i. N_{backfill} ← total resources executing jobs that coming after job j;
      ii. if N_{j} ≤ (N_{backfill} + N_{VH_idle}) then
         1. If j \(\in\) J_{VeryHigh_cpu} Then
            1. Remove j from J_{Low_CPU}, J_{Medium_CPU}, J_{High_CPU}
            2. Record status of j and stop execution on Low_CPU, Medium_CPU, High_CPU;
         3. If j \(\notin\) J_{VH_idle} then
            1. Switch j to the Low_CPU or Medium_CPU or High_CPU tier save and suspend backup VeryHigh_CPU jobs as per the decreasing order of entry time until N_{VH_idle} ≥ N_{j}.
   iii. If j \(\in\) J_{VeryHigh_cpu} Then
      1. Remove j from J_{Low_CPU}, J_{Medium_CPU}, J_{High_CPU}
      2. Record status of j and stop execution on Low_CPU, Medium_CPU, High_CPU;
   iv. Else
      1. Detach j from Q;
      2. send(VeryHigh_CPU, j);
      \(\bigcup J_{Low_CPU} \bigcup J_{Medium_CPU} \bigcup J_{High_CPU}\);
   3. j ← load first job from queue(Q)
4. function dispatch(flag, j)
5. Start
   a. As per job requirements, concurrent processes of a job are arranged by decreasing order of resource usage.
   b. If “flag” == “VeryHigh_CPU” then
      i. p ← arranged Idle resources in increasing order of their resource usage (caused by VeryHigh_CPU load);
      ii. For all jobs j, in the ordered list do
         1. Load j in p_{i} and execute in VeryHigh_CPU Resource;
2. evict the Low_CPU, Medium_CPU, High_CPU jobs, if usage of j_{i} exceeds threshold;
3. Else
   i. p ← arranged Idle resources in increasing order of their resource usage (caused by VeryHigh_CPU load);
   ii. for all jobs j do
      iii. Load j in p_{i} and execute in Low_CPU, Medium_CPU, High_CPU Resources, if usage of p_{i} is below threshold;
1. Load j in J_{Low_CPU} or J_{Medium_CPU} or J_{High_CPU}

The algorithm 7 is used to execute a task in Very-High CPU resources whenever the numbers of free Very-High CPU resources are enough or used by tasks reaching later than it, cater the task requirements. Simultaneously, it permits tasks to work in low CPU resources, medium CPU resources and High CPU resources with those Very-High CPU resources to improve node utilization.

This work allocates the tasks into the resources with the consideration of resource utilization level and the deadlock avoidance. It cannot avoid task failure when the resources cannot meet deadlines, when executing the tasks. Thus, this work can be improved by adding another method called task migration. This task migration is performed when the resources fail to meet the deadline of tasks as explained below.

### 2.3 Two Phase Commit Protocol with Novel Backfilling Technique and Task Migration (TPCNSBM)

So far, the tasks are allocated to resources in consideration of resource utilization and deadlock avoidance. After allocation of tasks the resources there is a chance of task failure due to physical failure of resources. In this situation, the resource cannot complete task execution within the guaranteed deadline. The tasks are migrated to other resources to complete processing of failed tasks.

Transfer of a task from the current resource to a new free resource for processing is called task migration. Backfilling based migration is applicable to first task in a queue to override the priority of other jobs. Except the first job in a queue, all other jobs are sent to only free resources and pre-emption is not applicable to them.

The Two Phase Commit Protocol and the Backfilling Techniques are used as described above in task migration. Task migration can also use when resources fail to
satisfy the QoS requirements. Task migration algorithm is described below in algorithm 8.

**Algorithm 8, NBF with Migration**

1. for all $n_i \in \text{Task}(n)$
2. Calculate current time $t_i$ being executed.
3. The difference between EET ($n_i$) of the current task being executed and current time gives the predicted starting time.
4. Calculate expected completion time of all the tasks in the queue
5. The list of tasks in the ready queue is sorted, based on their expected time
6. When the current executing task reaches its deadline, it is migrated to another resource.
7. Choose the task from the ready queue which has the highest expected time and starts its execution.
8. When a new task arrives, put the task at the head of the ready queue, then sort the tasks in the ready queue based on the recalculated expected time.
9. A request of migration is made to scheduler
10. The task is migrated to other resource scheduled based on NBT.
11. The task's migration is completed, once all states are transferred to scheduler.

Migration improves overall system performance and maximizes the utilization of resources. When a task reaches its deadline, a migration request is issued to another resource for the completion of the process and the task to be detached, suspends its processing in the current resource after marking its status as migration state. After queuing arriving messages temporarily, redirect communication channels to the migrated task and deliver them after migration. Till the completion of migration process, task is preserved in the source node. Status of the task is copied into an instance formed in the new resource. After that, destination node will be promoted into regular process. After task migration, source node will delete the copy of migrated task retained there.

### 3. Performance Analysis

The performance analysis of Cost Driven Work Flow Scheduling (CDWFS), Two Phase Commit Protocol with novel back filling technique (TPCNBF) and the Two Phase Commit Protocol with novel back filling technique and migration (TPCNBFM) validated with the help of experimental results explained below. Four parameters were considered to evaluate the performance of scheduling algorithms are:

- Makespan
- Execution Cost
- Throughput
- Average Success Ratio

Experiments were performed using GridSim. Using it created 5 GIS (Grid Information Service), 50 users, 500 jobs, 30 resources with different bandwidth and routers. Each resource runs multiple tasks. Two routers are used in this experiment with different characteristics like baud rate, propagation delay, and the Maximum Transmission Unit (MTU). TPCNBFM showed better performance in simulation.

#### 3.1 Makespan

Response time of entire workflow is called makespan. Makespan is defined as the time difference in between the sending time of an entry task and output receiving time of an exit task in a workflow. The experimental result is as shown in Figure 1. In the experiment 50 to 500 input tasks were given, plotted in x-axis and the average makespan varied from 0 to 500 sec. The values in Table 1 show that TPCNBFM is more efficient than CDWFS and TPCNBF due to the migration support as well as high throughput.

![Figure 1. Makespan.](image-url)
### Table 1. Makespan

| S. NO | Number of Task | CDWFS | TPCNBF | TPCNBFM |
|-------|----------------|-------|--------|---------|
| 1     | 50             | 2510  | 2480   | 2310    |
| 2     | 100            | 2930  | 2749   | 2230    |
| 3     | 150            | 2992  | 2900   | 2450    |
| 4     | 200            | 3230  | 3150   | 2920    |
| 5     | 250            | 3570  | 3412   | 3090    |
| 6     | 300            | 3856  | 3670   | 3450    |
| 7     | 350            | 4010  | 3850   | 3410    |
| 8     | 400            | 4230  | 4089   | 3700    |
| 9     | 450            | 4570  | 4150   | 3830    |
| 10    | 500            | 5010  | 4715   | 4050    |

### Table 2. Execution cost

| S. NO | Number of Task | CDWFS | TPCNBF | TPCNBFM |
|-------|----------------|-------|--------|---------|
| 1     | 50             | 23    | 22     | 17      |
| 2     | 100            | 27    | 24     | 19      |
| 3     | 150            | 29    | 26     | 22      |
| 4     | 200            | 31    | 29     | 26      |
| 5     | 250            | 33    | 31     | 28      |
| 6     | 300            | 36    | 34     | 31      |
| 7     | 350            | 39    | 36     | 32      |
| 8     | 400            | 41    | 37     | 34      |
| 9     | 450            | 42    | 39     | 35      |
| 10    | 500            | 51    | 47     | 37      |

### 3.2 Execution Cost

The execution cost is defined as the cost of processing a task on a particular resource. Execution cost changes according to the resource characteristics. In the experiment, 50 to 500 input tasks were given and processed in different resources as shown in Figure 2. From Table 2, it is clear that the average cost of TPCNBFM between 17$ to 37$ is lesser than CDWFS and TPCNBF due to the conditions in Assign Path algorithm and Backfilling Algorithms.

### 3.3 Throughput

Throughput is defined as the number of tasks processed by the resources within a unit of time. Throughput was calculated from the experimental results for a group of 50 to 500 tasks and the corresponding output in kbps is shown in Figure 3. TPCNBFM had the maximum throughput in

### 3.4 Average Success Ratio

The ratio between the total number of successful tasks to the total number of given tasks is called average success ratio. Average success ratio was calculated from the experimental results for a group of 50 to 500 tasks as shown in Figure 4. The average success ratio of TPCNBFM is higher than CDWFS and TPCNBF because of the reduced failure of tasks due to task migration in TPCNBFM as shown in the experimental result in Table 4.

### 4. Conclusion

Older scheduling methods were to reduce total execution time, without any considerations to reduce usage...
Novel Backfilling Technique with Deadlock Avoidance and Migration for Grid workflow Scheduling

In this paper we introduced a scheduling algorithm to reduce execution time and cost of grid resources with the combination of three different methods. The first method, two-phase commit algorithm is to avoid deadlock occurrence due to data dependency. The second strategy, backfilling technique is to improve resource utilization and fault tolerance in job scheduling process. The final method is task migration to safeguard tasks that miss their deadlines to be migrated to other resources and complete the processing. The Experimental results of the proposed scheduling algorithm showed improved resource utilization, success ratio, throughput and reduced execution time and cost.

5. References

1. Foster I, Kishimoto H, Savva A, Berry D, et al. The open grid services architecture Version1.50, 2003. Available from: http://forge.gridforum.org/projects/ogsa-wg
2. Wieczorek M, Prodan R, Fahringer T. Scheduling of scientific workflows in the ASKALON grid environment. ACM SIGMOD Record. 2005 Sep; 34(3):56–62.
3. Ramakrishnan A, Singh G, Zhao H, Deelman E, Sakellariou R, Vahi K, Blackburn K, Meyers D, Samidi M. Scheduling data intensive workflows onto storage-constrained distributed resources. Proceedings of the 7th IEEE Symposium on Cluster Computing and the Grid (CCGrid’07); Rio De Janeiro; 2007. p. 401–9.
4. Amin K, Hategan M, von Laszewski G, Zaluzec NJ, Hampton S, Rossi A. GridAnt: a client-controllable grid workflow system. Proceedings of the 37th Hawai’i International Conference on System Science; 2004. DOI: 10.1109/HICSS.2004.1265491.
5. Malewicz G, Foster I, Rosenberg AL, Wilde M. A Tool for prioritizing DAGMan jobs and its evaluation. Journal of Grid Computing. 2007; 5(2):197–212.
6. Foster I. Globus toolkit version 4: software for service-oriented systems. Lect Notes Comput Sci. 2005; 3779:2–13.
7. The Condor Project Website. 2014 Oct 12. Available from: http://www.cs.wisc.edu/condor/
8. You SY, Kim HY, Hwang DH, Kim SC. Task scheduling algorithm in GRID considering heterogeneous environment. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’04); Nevada, USA: 2004. p. 240–5.
9. Mandal A, Kennedy K, KoelbelC, Marin G, Mellor-Crummey J, Liu B, Johnsson L. Scheduling strategies for mapping application workflows onto the grid. Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC’05); 2005. p. 125–34.
10. Snell Q, Jackson D, Clement M. Core algorithms of the maui scheduler. Job scheduling strategies for parallel processing. Lect Notes Comput Sci. 2001; 2221:87–102.

11. Mu’alem AW, Feitelson DG. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM sp2 with backfilling. IEEE Trans Parallel Distr Syst. 2001; 12(6):529–43.

12. Moreiraz JE, Sivasubramaniamy A, Zhangy Y, Frankez H. An integrated approach to parallel scheduling using gang-scheduling, backfilling and migration. IEEE Trans Parallel Distr Syst. 2003; 14(3):236–47.

13. Qiu X, Cai Y, Liu X, Chen B, Huang K. Scheduling Parallel jobs using migration and consolidation in the cloud. Math Probl Eng. 2012; 2012:1–18.

14. Xu B, Zhao C, Hu E, Hu Bin. Job scheduling algorithm using Berger model in cloud environment. Adv Eng Software. 2011; 42(7):419–25.

15. Naghibzadeh M, Abrishami S. Deadline constrained workflow scheduling in software as a service cloud. Scientia Iranica. 2012; 19(3):680–9.

16. Carrera D, Steinder M, Whalley I, Torres J, Ayguade E. Autonomic placement of mixed batch and transactional workloads. IEEE Trans Parallel Distr Syst. 2012; 23(2):219–231.

17. Ghanbari S, Othman M. A priority based job scheduling algorithm in cloud computing. International Conference on Advances Science and Contemporary Engineering (ICASCE 2012); 2012. p. 778–85.

18. Rahman M, Hassan R, Ranjan R, Buyya R. Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurrency Comput Pract Ex. 2013; 23(13):1816–48.

19. Wu X, Deng M, Zhang R, Zeng B, Zhou S. A task scheduling algorithm based on QoS-driven in cloud computing. Procedia Computer Science. 2013; 17:1162–9.

20. Xiao Z, Song W, Chen Qi. Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Transactions on Parallel and Distributed Systems. 2013; 24(6):1107–17.

21. Liu X, Wang C, Zhou BB, Chen J, Yang T, Zomaya AY. Priority-based consolidation of parallel workloads in the cloud. IEEE Transactions on Parallel and Distributed Systems. 2013; 24(9):1874–1883.

22. John SM, Mohamed M. Workflow scheduling algorithm for grid services with hierarchical grouping based QoS. Asian Journal of Applied Sciences. 2015; 8(1):1–15.

23. Ranaldo N, Zimeo E. Time and cost-driven scheduling of data parallel tasks in grid workflows. IEEE Systems Journal. 2009; 3(1):104–120.

24. Dhingra A, Paul S. Green cloud: heuristic based BFO technique to optimize resource allocation. Indian Journal of Science and Technology. 2014; 7(5):685–91.