Grid-job scheduling with reservations and preemption

Xiaorong Cao
University of Windsor

Grid-Job Scheduling with Reservations and Preemption

By

Xiaorong Cao

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2009

Your file Votre référence
ISBN: 978-0-494-57628-1
Our file Notre référence
ISBN: 978-0-494-57628-1

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Computational grids make it possible to exploit grid resources across multiple clusters when grid jobs are deconstructed into tasks and allocated across clusters. Grid-job tasks are often scheduled in the form of workflows which require synchronization, and advance reservation makes it easy to guarantee predictable resource provisioning for these jobs. However, advance reservation for grid jobs creates roadblocks and fragmentation which adversely affects the system utilization and response times for local jobs. We provide a solution which incorporates relaxed reservations and uses a modified version of the standard grid-scheduling algorithm, HEFT, to obtain flexibility in placing reservations for workflow grid jobs. Furthermore, we deploy the relaxed reservation with modified HEFT as an extension of the preemption based job scheduling framework, SCOJO-PECT job scheduler. In SCOJO-PECT, relaxed reservations serve the additional purpose of permitting scheduler optimizations which shift the overall schedule forward. Furthermore, a propagation heuristics algorithm is used to alleviate the workflow job makespan extension caused by the slack of relaxed reservation. Our solution aims at decreasing the fragmentation caused by grid jobs, so that local jobs and system utilization are not compromised, and at the same time grid jobs also have reasonable response times.

Acknowledgements

I would like to thank my advisor Dr Angela C. Sodan for her guidance and support during the course of my research work. I am thankful for the technical discussions we have had and the opportunity she gave me to conduct research work.

Also thanks to my committee members Dr. Mitra Mirhassani, Dr. Y.H.Tsin and Dr. Dan Wu for their valuable time and suggestions.

Table of Contents

Author's Declaration of Originality iii

Abstract iv

Acknowledgements v

List of Tables viii

List of Figures ix

1. Introduction 1

2. Related Work 6

2.1 Grid Scheduling 6

2.2 Reservation and Relaxed Reservation 6

2.3 HEFT 8

2.4 Co-reservation 9

2.5 Communication Awareness Allocation 10

3. SCOJO-PECT 12

4. Grid-Job Scheduling with Reservations and Preemption 17

4.1 Extension of SCOJO-PECT 17

4.2 Assumptions 17

4.3 Goals 18

4.4 Grid Job Scheduling Model 20

4.4.1 Workflow Scheduling 20

4.4.2 Model Definition 21

4.4.3 Workflow Scheduling Problem Description 21

4.4.4 HEFT Algorithm: Solution for Makespan Minimization Objective and Task Dependency Constraint 22

4.4.5 Relaxed Advance Reservation in SCOJO-PECT: Solution for Local Scheduling Policy Constraint 24

4.4.6 RARS-HEFT Algorithm.....26

4.5 Grid Scheduler.....28

4.6 Prediction of Response Time for Advance Reservation.....29

4.7 Advantages of Relaxed Reservation in Real Scenarios.....31

4.8 The Conflict between Makespan Minimization Objective and Local Scheduling Policy.....34

4.9 Solution for Conflict: Propagation Heuristics.....35

4.10 System Design.....37

4.10.1 Design Modifications in EventBaseSimulator Package.....37

4.10.2 Design Modifications in HPCSimulation Package.....38

4.10.3 The GridSimulation Package.....39

4.11 Workload Modeling.....41

4.11.1 Local Job Modeling.....41

4.11.2 Grid Job Modeling.....42

5. Experiments and Results Analysis.....44

5.1 Testing Plan.....44

5.2 Experiment Result.....44

6. Summary and Conclusion.....50

7. Future Work.....52

References.....53

Vita Auctoris.....57

List of Tables

Table 1. HEFT Algorithm Example....................23
Table 2. Grid Scheduling Algorithms....................24
Table 3. Experiment Workload Setup....................41
Table 4. Scheduler Parameters....................41

List of Figures

Figure 1. Workflow Job Structure 2

Figure 2. Workflow Job Scheduled in Grid 3

Figure 3. Resource Fragmentation 4

Figure 4. Relaxed Reservation 4

Figure 5. SCOJO-PECT Dynamic Slices 13

Figure 6. Job State Diagram 14

Figure 7. Slice State Diagram 15

Figure 9. Grid Job Processing 28

Figure 10. Original Design of Prediction 30

Figure 11. Modified Design of Prediction 31

Figure 12. Advantage of Relaxed Reservation When Job Ahead Finish Earlier 32

Figure 13. Advantage of Relaxed Reservation during Rescheduling Optimization with Backfilling 33

Figure 14. Advantage of Relaxed Reservation Which Saves the Preemption Cost 34

Figure 15. Spare Time in Propagation Heuristics 35

Figure 16. Best Scenario of Propagation Heuristics 36

Figure 17. Class Diagram of EventBaseSimulator 37

Figure 18. Class Diagram of Resource 38

Figure 19. Class Diagram of HPCSimulation 39

Figure 20. Class Diagram of GridSimulation 40

Figure 21. Generate Workflow Job B by Splitting Simple Job A 42

Figure 22. Average RR of the Clusters According to the Percentage of Task Reservations in All Jobs 45

Figure 23. Average RR of Grid Tasks in the Clusters According to the Percentage of Task Reservations in All Jobs.... 45

Figure 24. Average Makespan of Grid Jobs According to the Percentage of Task Reservations in All Jobs 46

Figure 25. Average RR of Local Clusters with Respect to Presence or Absence of Propagation 47

Figure 26. Average RR of Grid Tasks in Clusters with Respect to Presence or Absence of Propagation 47

Figure 27. Average Makespan of Grid Jobs According to Propagation Heuristic 48

Figure 28. Average Job RR of 4 Clusters According to Reservation Time Frame Portion 48

Figure 29. Average Gird Task RR of 4 Clusters According to Reservation Time Frame Portion 49

1. Introduction

Scheduling is the problem of allocating tasks to resources over time and under certain constraints. In High Performance Computing (HPC) area, job scheduling problem is to allocate the *jobs* (computational tasks) to *resources* (processors) over time. This kind of problem involves resource management based on the characteristics of jobs (computational workload, parallelization level, resource requirement, start/end time constraints, communication time with other jobs, and relative priorities) and the characteristics of resources (scale of processors, scheduling policies, computational speed, reservation, preemption and restoration).

Optimization plays an important role in job scheduling, and has significant impact on the *resource provider* and *resource user*. From *resource providers'* point of view, load balance and system utilization is the key consideration to maximize their profit. From the *resource user* point of view, to minimize the response time of the jobs is preferable. According to the key issues above, our optimization focus on this job scheduling problem is to minimize *response time*, ensure *scheduling fairness*, and maximize the *resource utilization*. *Response time* of a job is the time span from the submission time to finish time of the job, and a long delay in a job may degrade the quality of service. *Fairness* is the overall evaluation based on the comparison between job's actual response time and its acceptable response time. *Resource utilization* is the percentage of used processors over time, and that is to say the less processors stay in idle state, the better it is.

In the scheduling system, the optimization is conducted by the job scheduler, which resides in the operating system or scheduling middleware. Usually, resources are not immediately available and jobs are queued for future scheduling. Resources are shared by jobs through space sharing (subsets of processors are assigned exclusively to certain jobs) and time sharing (multiple jobs share the same resources in different time slices). The job scheduler assigns the queued jobs to the available resources based on scheduling policy, and is responsible for identifying available resources, predicting job start time and finish time, matching job requirement to resource capacity, and making decisions on job preemption, restoration and reordering.

When it comes to *resources* in HPC research, the focus is usually on *cluster* and *grid.* A *cluster* is a group of processors connected by local area network, and the processors work cooperatively under the control of a centralized scheduler. A *grid* is known as a group of geographically decentralized heterogeneous clusters connected by wide area network. In this thesis, the target resources are cluster and grid.

There are two kinds of *jobs* mentioned in this research, *local jobs* and *grid jobs.* Local jobs are submitted and processed in the same cluster, while *grid jobs* demand high amount of resources, and are allocated globally across the clusters to alleviate the heavy workload of one cluster and minimize job response time. One kind of grid job, *workflow job,* consists of multiple tasks which have dependencies among each other (the output of one task is the input to another task); the geometrical description of workflow job is a Directed Acyclic Graph (DAG) as seen in Figure 1.

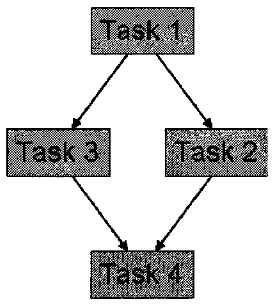


Figure 1. Workflow Job Structure

As can be seen in Figure 2, the *Grid-scheduler* discovers and selects the appropriate cluster for grid jobs; estimates job start time and response time for given resources; then based on the current state of clusters and the job requirements, it decides which clusters to use for grid jobs.

The grid-scheduler in this thesis does not have centralized control over the clusters in the grid. Instead, it gets the prediction of task schedule from clusters before allocating the grid job tasks to clusters, and based on the prediction, grid scheduler makes resources allocation decision. The challenges of the grid-scheduler are to guarantee the schedule of workflow job tasks in an optimized order. *Advanced reservation* is a good solution for this challenge which brings about the guaranteed schedule for grid jobs.

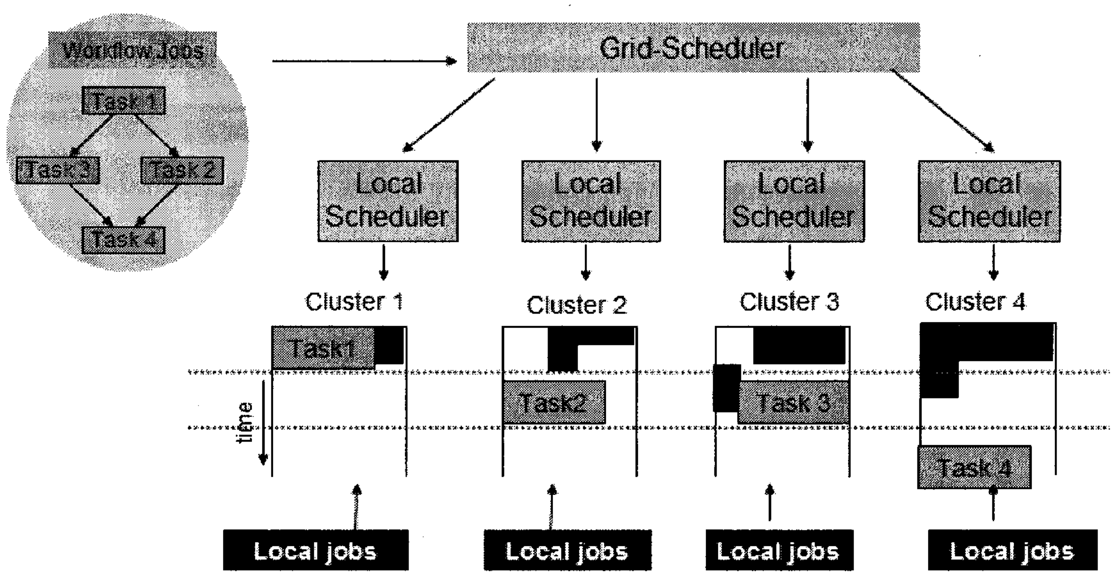


Figure 2. Workflow Job Scheduled in Grid

Advanced reservation is a request for certain amount of processors at certain time periods in future made by cluster user and approved by cluster scheduler. When the reservation is made, the absolute start time and the end time will be reserved. However rigid character of reservations can easily lead to a decrease in the system utilization and delay in the local jobs. As seen in Figure 3, the rigid reservations leave fragmentations (unused resources) too small to be allocated to the next job.

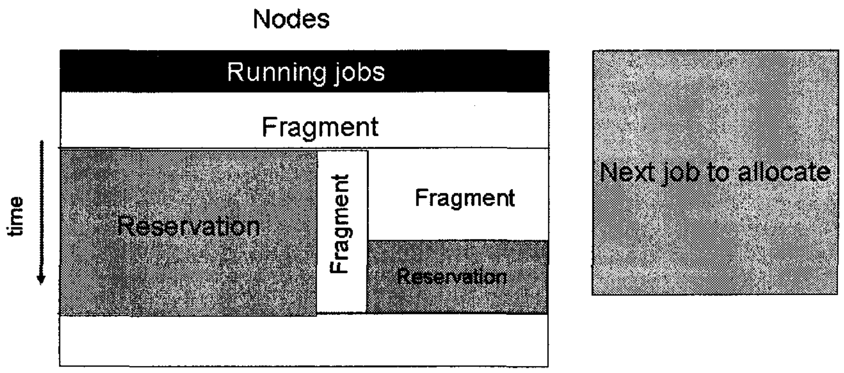


Figure 3. Resource Fragmentation

Relaxed reservation allows some flexibility to the advanced reservation which has a time frame with earliest start time and latest finish time, and based on the cluster status, reservation slides within the time frame to fill the fragmentation. As seen in Figure 4 below, the red rectangle is time frame of reservation, and when fragmentation occurs ahead of reservation, the reservation can slide up to fill the fragmentation if there are enough resources. Here, "slide" means reschedule.

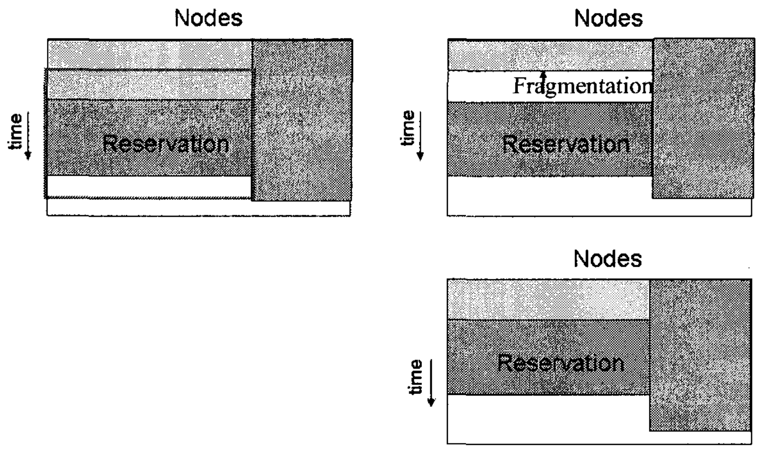


Figure 4. Relaxed Reservation

This thesis proposes a combination of solutions for grid job scheduling including Heterogeneous Earliest Finish Time (HEFT) algorithm for workflow job, relaxed reservation, and extends the implementations in SCOJO-PECT simulation system. Further, the propagation heuristics are used to speed up the workflow jobs. System utilization and average relative response time are used as a metric to evaluate the solutions.

The rest of the thesis is organized as follows. Related work is introduced in Section 2. Implementation details are discussed in Section 3. Experimentation setup and result analysis are presented in Section 4. Section 6 discusses the conclusion. And finally in Section 7 future work is discussed.

2. Related Work

2.1 Grid Scheduling

Grid scheduling maps the multiple tasks of a grid job into multiple clusters, and ensures the resource allocation and predicted time scheduling. Grid scheduling includes multiple issues not only in resource management and scheduling, but also scheduling prediction, service level provision, load balancing, dependencies and synchronization between tasks, and network connectivity between clusters. Current research has shown various approaches based on grid scheduling. In [1], the centralized grid scheduler allocates jobs based on the workload of every cluster, which prevents the cluster from being overwhelmed by large grid jobs. It implements local backfilling schedules and global backfilling schedules to ensure balanced workload between clusters. In [2], a distributed scheduler is introduced which applies resource negotiation and advance reservation, network connectivity element and performance estimation on job scheduling. In [3], bi-criteria grid scheduling is not only optimized over the response time, but also over another optional constraint such as the financial cost of running a task based on service level. This is also a distributed scheduler, and it provides acceptable trade off between response time and any other criterion such as resource accessibility, and leads to better quality of service based on two criteria.

2.2 Reservation and Relaxed Reservation

Unless a global scheduler performs all scheduling of local and grid jobs upon current availability of resources, reservation is absolutely necessary for workflow jobs with multiple tasks to ensure their scheduling with dependencies. Distributed scheduling requires a corresponding negotiation protocol for making the reservations [4]. The scheduling decision requires that multiple possible scheduling slots are obtained from each site and matched against each other and against the requirements of the job [5]. Reservation, though not absolutely necessary, may also be desirable for the workflow model to provide quality of service and ensure that the job completes in a certain time frame [6] [7].

However, reservations involve several problems. They can easily lead to fragmentation which may lead to an increase in average response times [8] [5]. Furthermore, it may be difficult to find all the required time slots in a distributed negotiation protocol, especially if requiring simultaneous time slots [5].

To overcome the problem of potential fragmentation and increase in response times from reservations, reservation times may be kept flexible within certain time frames to adjust to dynamic resources availabilities, including dynamic start times for unpredictable fragmentations ahead and dynamic completion times for the insertion of other jobs ahead of the reservation [9] [10]. This approach is suitable for workflow scheduling but would not work for coordination of simultaneous grid jobs. A similar idea is to extend each reservation for tasks in a workflow graph to deal with uncertainties and possible delays from preceding jobs overrunning their reservations [11]. Closely related reservations which permit different resource allocation to satisfy a scheduling request within a certain time frame increase the flexibility of scheduling [12] but can support only workflow models. Alternatively, size adaptation may be used to schedule local jobs around reservations for grid jobs [13]. Flexible resource allocation in time and space dimension become feasible for simultaneous grid jobs if the start time is fixed after negotiation but the system is granted the dynamic choice to run the job dedicated with fewer number of resources or time-shared with a larger number of resources, while maintaining the same reserved runtime [14].

Other tentative solutions to the fragmentation problem are bucket and space/time partitions. Pricing may be supported by creating different buckets along the space-time axis with potentially different space and time allocations which corresponds to a partitioning of the system in both the time and space dimension. (The latter is well-known from theoretic off-line scheduling as the shelves approach.) Grid jobs may be placed/reserved as they fit into the different buckets and be charged according to different bucket-start-times (in addition to potentially the jobs' sizes and runtimes) [15]. The bucket approach may lead to fragmentation and difficulties in handling jobs with very long and largely varying runtimes or in cases where jobs may run much shorter than predicted.

Preemption of jobs can help avoid fragmentation. It can combine reservations for grid job with the local jobs and schedule them together in the coarse-grain time share system according to the job categories [16]. Similarly, gang scheduling can help increase the probability that reservations for grid jobs and local jobs schedule well together but coordination of simultaneously executing and communicating grid jobs would become difficult [13]

The negative impact of reservations and the difficulty in finding scheduling slots may also be alleviated by imposing limits on the task sizes per site and rather spreading the job over a larger number of sites [17]. However, this approach only works as long as the remote communication cost is acceptable, and the benefits from the probability of finding matching slots for smaller requests have to be balanced with the probability of finding such slots on a larger number of sites. Imposing limits on the size of the reservations also permits improving performance bounds of the scheduling of the other jobs [18].

2.3 HEFT

A workflow job is modeled as workflow graph (task graph) to represent parallel subtasks with dependency between each other. There are two different kinds of workflow scheduling: best-effort based and quality of service constraint based [19], and many algorithms are available to solve the workflow job scheduling. One such algorithm is Heterogeneous Earliest Finish Time (HEFT) algorithm, which is a list scheduling algorithm. The objective of HEFT algorithm is to find a mapping of activities to resources that minimizes the schedule length (makespan, defined under WorkflowJob in section 4.6.2) [20], with comparatively low complexity and high efficiency [6]. HEFT can be both best-effort based and quality of service constraint based. The variations of HEFT algorithm are discussed in many papers. When it comes to the workflow scheduling with deadline, unpredictable delay will lead to rejection of the whole workflow job if a part of the task cannot meet the deadline. Fault tolerance scheduling avoids the heavy penalty due to the unpredictability in the scheduling problem [11]. The spare time is added in every task's execution time to relax the task deadline, and the user requested workflow job deadline is used to distribute slack over tasks [11]. Background workload of clusters is used in HEFT algorithm for scheduling decision to ensure that clusters are not overloaded [21]. During the run time, the global scheduler detects newly available clusters, and migrates the tasks to available clusters through global scheduling [22]. Quality of service constraint can be used as a second dimension to reach an acceptable trade-off between makespan and concerns such as financial cost, deadline guarantees, and data accessibility [23] [3]. An additional optimization lies in the combining subtasks of two different workflow jobs, and interweaving task graphs to minimize fragmentation [24]. A search tree is built up during the creation of HEFT task list, and the tree allows backtracking for better global optimization [20].

2.4 Co-reservation

Co-reservation has been used in current research and application services for maintaining good performance. In Globus Architecture for Reservation and Allocation (GARA), the global co-reservation agents have the responsibility to discover collection of resources that satisfy an end-to-end quality of service requirement defined by the resource user, and at the same time the local Globus Reservation and Allocation Manager (GRAM) addresses the issue of heterogeneity in a resource set [25]. Also, the deadlock caused from reservation failure can be avoided by enforcing a standard on how resources are acquired (for example, IP address followed by disk number) [25]. Reservations for jobs are set sequentially by users or agencies negotiating with local schedulers which is an easier approach than using centralized meta-schedulers to control local schedulers at different sites [26]. A three-layered negotiation protocol can be used for advance reservation using a utility function based on a distance formula. The distance formula depends on resource demand and supply [15]. The three layers are - allocation layer within individual nodes, the co-allocation layer between nodes, and the inter-application layer [15]. In [27] [28], Worst Fit co-allocation policy tries to avoid reusing the same cluster, and leaves roughly equal numbers of idle processors in all clusters to balance the load of clusters.

2.5 Communication Awareness Allocation

The communication overhead among grid job tasks is discussed further in [29-31]. When grid tasks are allocated to different resources, decomposition process is used for partitioning the tasks into task groups, in order to minimize the communication among task partitions when the task components are running in parallel across clusters. There are two job placement policies: Cluster Minimization Policy (splits job into components with fixed processor requirement, and minimizes the number of clusters to be combined for a given parallel job) and Flexible Cluster Minimization Policy (splits job into components according to available processors in each clusters, and minimizes the number of clusters to be combined for a given parallel job and decrease the queue time of jobs) [29]. Two types of decomposition approaches are supposed to be well suited for task decomposition: Domain Decomposition which partitions the tasks of a job according to their disjoint access to data domains, and Computational Decomposition which decomposes the tasks according to their independent use of processors while sharing the same data [29]. Another approach is based on a job structure with an initial set of tasks and all-to-all communication information. This approach groups tasks which communicate more frequently into one task collection by using flow theory, so that the communication between clusters is minimized [30]. Furthermore, when a job's tasks are allocated across multiple clusters, the system puts an upper limit on the inter-connection link capacity between adjacent clusters to reduce communication, and a lower limit on available processor number in each cluster for job partition to minimize the number of job partitions; thus the communication between multiple clusters is minimized [31].

3. SCOJO-PECT

In this thesis, the Grid-Job Scheduling with Reservation and Preemption is proposed as an extension to SCOJO-PECT. SCOJO-PECT is a job scheduling simulation framework with extensive functionality supporting job generation, resource allocation, schedule prediction, job scheduling based on time/space sharing, workload statistics and scheduling statistics. It simulates the scheduling of one *cluster* with 128 processors.

There are several characteristics of SCOJO-PECT:

- **Job Class:** Jobs are classified into 3 classes (short, medium and long) based on their running time. [16]
- **Basic scheduling policy:** The basic scheduling policy is first-come-first-serve (FCFS) for jobs in the same class.
- **Basic Backfilling Policy:** Backfilling is used to increase utilization and decrease fragmentation for the resource and jobs in the back of the queue can be scheduled ahead of the jobs in the front of the queue if the former do not delay the latter. The basic backfilling policies of SCOJO-PECT are easy backfilling and conservative backfilling. In conservative backfilling, a job may be backfilled only if it does not delay any other job ahead of it in the queue. In easy backfilling, a job may be backfilled only if it does not delay any job other than the first one ahead of it in the queue.
- **Different service levels:** The scheduler provides three different slices (short, medium and long) according to each job class. The jobs run in their own slices, and each slice has different duration. For example, long slices have longer duration than the others, which allows long jobs to run longer in one interval (consisting of three different slices). [16]
- **Preemption:** The three slices use preemption as they take turns, and it solves the

problem such as when wide-long jobs block short or medium jobs from being scheduled. [16] At the end of each slice, unfinished jobs are forcibly stopped and suspended; and at the beginning of the same slice in the next round, the stopped jobs are restarted on the same resources.

- **Coarse-grain time sharing scheduler**: Jobs in the same class are processed in parallel in the form of space sharing; jobs in different classes share the resources in the form of time sharing. Unlike gang scheduling which keeps the memory under pressure by preempting job information into memory, SCOJO-PECT is coarse-grained time sharing and suspends jobs to the disk at the end of the slice. [16] One interval with three slices is 60 minutes, and the overhead caused by job suspension only happens three times in 60 minutes.
- **Dynamic slice duration:** The duration of slice is adjusted during different times of the day or adjusted depending on the workload of the submitted job mix. As seen in Figure 5, during the night time, usually there are longer jobs, SCOJO-PECT automatically generates long slices with longer duration, and medium slices with shorter duration. If there are no short jobs, short slices will be removed. Although, the slice time changes, the interval time does not change, and the jobs of each class still have a chance to run in every interval. This dynamic adjustment provides more flexible scheduling according to the overall resource requirements of the different job classes. [16]

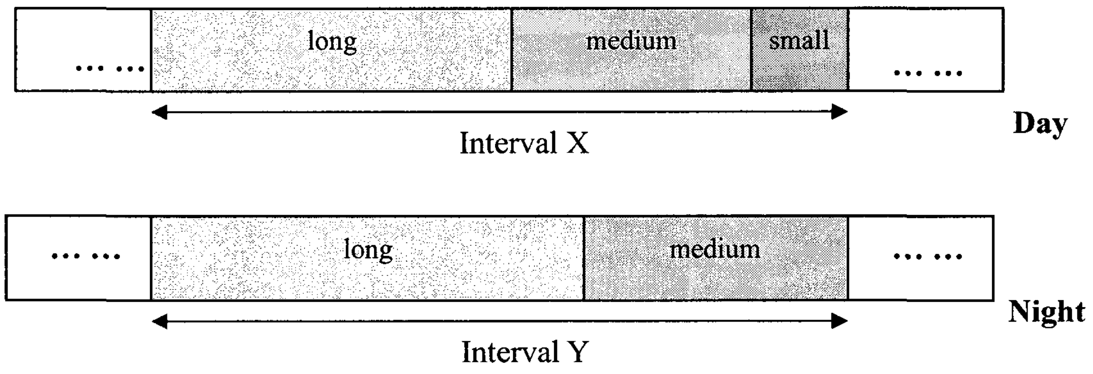


Figure 5. SCOJO-PECT Dynamic Slices

- **State maintenance**
  - **Job status:** When a job is submitted, it will stay in waiting queue until its own slice comes and there are enough resources for the job, it will stay in running queue if the job starts in its own slice or gets backfilled, it will stay in preemption queue after the slice ends and before the job finishes. The states of job are maintained by the queue which the job belongs to. Each job class (long, medium, and short) has their own queue, and the queues of different class are processed respectively. Figure 6 shows the job status diagram.

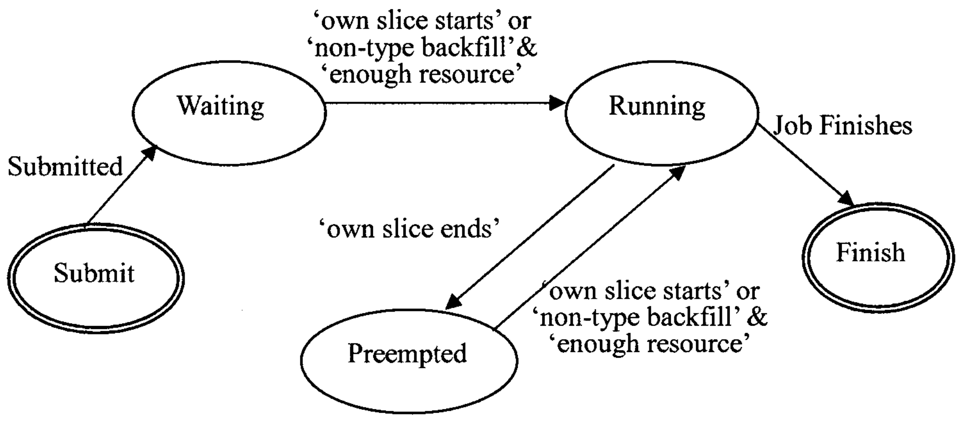


Figure 6. Job State Diagram

  - **Slice Status:** The slice status starts with 'None' status. When there are short jobs in the waiting queue, the system generates three slices (short, medium and long), and goes through the three slice status, then back to 'None' status. When there are no short jobs in waiting queue, the system generates two slices (medium and long), and goes through the two slice status, then back to 'None' status as seen in Figure 7.

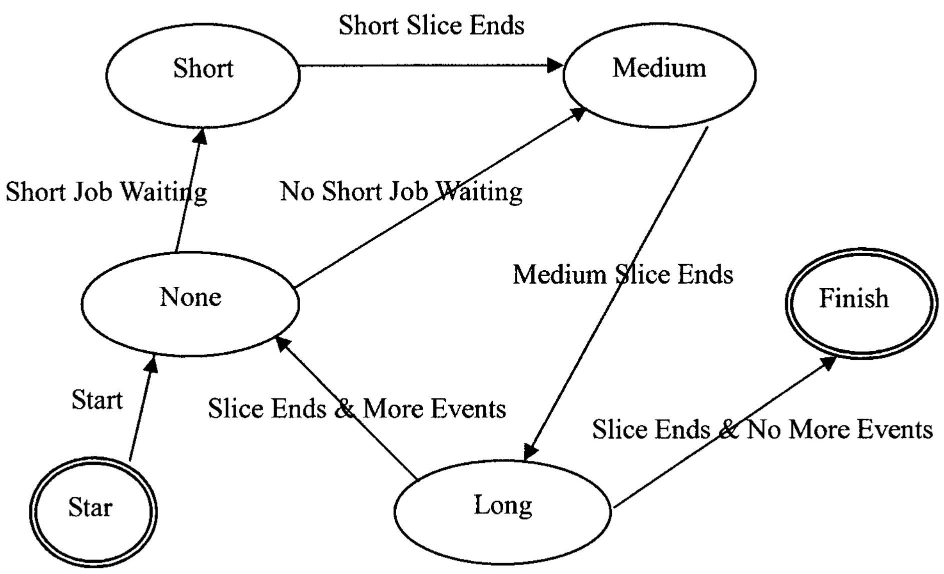


Figure 7. Slice State Diagram

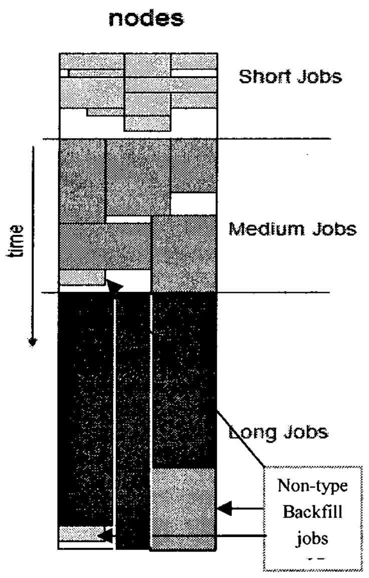


Figure 8. Multi-level Backfilling

- **Safe backfilling:** Safe non-type backfilling is employed, which means the preempted jobs and waiting jobs from other slices can be backfilled to the slices which do not belong to them, if there are enough resources and if it does not cause the delay of any jobs. Jobs backfilled in other slices can continue running on the same resources if their own slice is available. In Figure 8, two short jobs (in yellow) are backfilled into the medium job slice (in green) and the long job slice (in blue), and one medium job is backfilled into long slice. [16]

- **Prediction:** The prediction of response time of individual jobs is very important for quality of service and grid scheduler in making optimized scheduling decisions [32].

The well performed prediction takes advantage of FCFS and conservative backfilling policy. Considering the unpredictability of multi-level backfilling, the real response time would only be better and never be worse than the predicted response time [33]. Grid scheduling in this thesis benefits from this prediction, because it gives the grid scheduler necessary information for making global reservation decisions.

4. Grid-Job Scheduling with Reservations and Preemption

4.1 Extension of SCOJO-PECT

Grid-Job Scheduling with Reservations and Preemption is proposed as an extension of SCOJO-PECT [16], and there are many enhancements in this implementation, such as:

- SCOJO-PECT is a simulator of a single local cluster with a local scheduler. However, in this implementation, the system supports *multiple clusters* with their own schedulers, and a grid scheduler which works cooperatively with local clusters.
- The jobs generated in SCOJO-PECT are independent of each other, while in this implementation, workflow job tasks have dependencies among each other.
- SCOJO-PECT does not support advance reservation, jobs are scheduled based on FCFS policy. This implementation allows a reserved job to be scheduled after its earliest start time, which means the job may run later than some other jobs even though it is submitted before those jobs.
- Furthermore, SCOJO-PECT does not support start time and end time prediction for advanced reservation, it only predicts future job scheduling when jobs can be scheduled any time after submission. In this implementation, the earliest start time is used as a parameter for the reservation start time prediction.

4.2 Assumptions

The implementation of Grid-Job Scheduling with Reservations and Preemption is based on the following assumptions:

- The jobs are assumed to be computational intensive jobs which require significant CPU time, so the optimization is focused on processor resource, but not the resources like network bandwidth, storage or data.

- Reserved jobs are scheduled with best effort policy, not guaranteed policy. There is no deadline for reserved jobs, as long as they are submitted and scheduled, even if they are delayed and exceed the latest finish time, they are still processed as normal jobs. This is unlike some implementations where the reserved jobs are cancelled when they exceed the deadline.
- Network connectivity within the cluster is not considered, it is assumed that the processors have all-to-all connections to each other.
- The partitioning of grid jobs is considered as already decided before the grid job submission. There are complex issues related to partitioning such as optimization of partition tasks size and number based on network connectivity and resources capacity. In this implementation, the focus is job scheduling, so job partitioning issues are simplified and omitted.
- The workflow job execution and communication matrices are assumed to be the input by the users.
- The workflow job modeling is simplified, see Section 4.5.1
- Fault tolerance issues related to cluster and grid, such as network disconnection or job processing exception are not considered.
- The job run time is assumed to be input by the user.

4.3 Goals

The implementation of Grid-Job Scheduling with Reservations and Preemption is designed with the following goals:

- **Robust Design:** The implementation is designed to perform scheduling experiments of grid, grid jobs and reservations. Therefore it should be easy to refactor and extend.
- **Quality of service:** Advance reservation increases the quality of service for grid job scheduling; however it decreases the quality of service for local jobs. The rigid

reservation blocks the local jobs and can cause long response time. The design of relaxed reservation will help alleviate this problem.

- **Fairness:** To guarantee that each class of job has a chance to run during each interval period, the grid job tasks are classified into different classes according to the SCOJO-PECT pattern, which follows the original design based on fairness as a critical issue.
- **Minimize grid job makespan:** The makespan of grid job means the time duration from the start of the first task to the end of the last task. The scheduling should lower the makespan and ensure no unnecessary delay for each task with a local scheduling policy.
- **Accurate prediction:** Because workflow tasks have dependencies among each other, the system needs to ensure that the predecessor task finishes before the successor tasks starts. Furthermore, the system also needs to minimize the unnecessary delay between the completion of predecessor task and the start of the successor task. To reach these two goals requires accurate advanced reservation prediction which if inaccurate may cause heavy penalty for workflow jobs to reschedule from time to time.
- **Minimize degradation in system utilization:** Utilization is the percentage of the used resources over time. Advance reservation creates fragmentations (unused resources over a period), which decreases the system utilization. In this implementation, relaxed advance reservation is used to minimize the fragmentation and minimize degradation in system utilization.

4.4 Grid Job Scheduling Model

4.4.1 Workflow Scheduling

Usually workflow job is presented as directed graph called *workflow graph*. The nodes in the graph are considered as computational tasks, and the edges are considered as communication between tasks. A task cannot start until all its predecessor tasks finish and all communications between its predecessors and itself finish.

Workflow model Simplification

For simplicity, a few assumptions are made in the workflow modeling.

1. The workflow graph is assumed to be static. The real life workflow might be dynamic, which means the edges and nodes of the graphs are undecided before the tasks start, and the nodes and graph are dynamically generated depending on the result of predecessor nodes. In this implementation, dynamic generation of workflow graph is not considered, and static workflow graph is assumed to be the input by the user.
2. Directed Acyclic Graph (DAG) is used to represent workflow graph, which assumes that there is no loop in the directed graph. The real life workflow jobs may be more complex when they have loops which may need loop unrolling [33].
3. There is only one root and one exit in the workflow graph. DAG may have several roots and exits, however, for simplicity multiple roots and exits are not considered in this implementation.
4. When it comes to communication time between tasks, dependencies among tasks are important. Communication time is considered less significant than the computation time. So minimization of communication time is not considered in the model.

4.4.2 Model Definition

Resource

A resource R can be defined as a tuple (P, N, S). S is a set of scheduling policies for a given cluster, P is a set of clusters $p_1, p_2, \ldots, p_j$, which process jobs. All the clusters are fully connected with each other, and $N \subseteq \{(x,y)|x,y \in P\}$ is a set of network speed values between any two clusters.

Workflow Job

The workflow job can be defined as W (T, C) where T is a set of tasks $t_1, t_2, \ldots, t_k$, and C is a set of communications $c_1, c_2, \ldots, c_m$, and $C \subseteq \{(u,v)|u,v \in T\}$ is a set of communications between tasks. M is the makespan of workflow job, which is the time period from the beginning of the first task to the end of the last task. SC is the schedule of a task which leads to minimized makespan of the workflow job. L represents the level of the workflow graph, which is the total number of nodes on the critical path (the longest path in workflow graph from the root node to the exit node). BF (Branching Factor) defines the ratio between the number of nodes and the number of communications [34]. BF is important for evaluating the scheduling policy in terms of collocation capability.

4.4.3 Workflow Scheduling Problem Description

Input: Workflow job W (T, C) and given resource R (P, N, S),

Output: The schedule of every task in the workflow SC

Objective: Minimize the makespan M of workflow Job

Scheduling Constraints:

1. Task Dependency Constraint: The *dependencies* between tasks are satisfied.

2. Local Scheduling Policy Constraint: Schedule the tasks in clusters when the clusters' *local scheduling policies* (S) are satisfied.

4.4.4 HEFT Algorithm: Solution for Makespan Minimization Objective and Task Dependency Constraint

- **HEFT Algorithm: An Introduction**

Heterogeneous Earliest Finish Time (HEFT) is a dependency mode algorithm which provides a solution for ordering and mapping workflow tasks to heterogeneous resources based on workflow graph task dependencies to finish the workflow at earliest time [19].

There are three phases in HEFT algorithm [6]:

1. Weighting phase: For workflow W(T, C), get weights of the tasks in T and weights of communication in C are based on the resource R(P, N).

   Please notice, the resource model R in HEFT does not include local scheduling policy S unlike the model R(P, N, S) discussed in 4.6.2.1

   The weight of a task $t \in T$ is the average value of predicted execution times of all clusters:

   $$\overline{W}(t) = \underset{\forall p \in P}{avg} \{\text{ExcTime}_t(p)\}, \quad \forall\, t \in T \qquad (1)$$

   The weight of communication c is the average value of predicted communication time from cluster p_i to cluster p_j :

   $$\overline{W}(c) = \underset{\forall pi \in P, \forall pj \in P}{avg} \{\text{CommTime}_c(p_i, p_j)\}, \quad \forall\, c \in C \qquad (2)$$

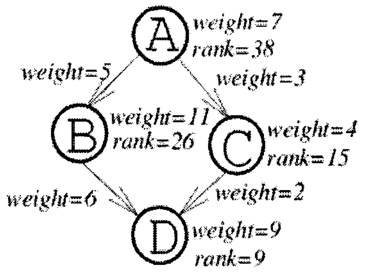
2. Ranking phase: Decide the order of tasks based on when the tasks start. The rank is calculated as the value of the task t's weight plus the maximum rank value of its successors:

   $$\overline{R}(t) = \underset{\forall (t,Succ) \in C}{\max} \{\overline{W}(t) + \overline{W}(t,Succ) + \overline{R}(Succ)\} \qquad (3)$$

3. Mapping phase: Sort the ranked tasks in descending order, and map to the cluster which provides the lowest latest finish time. The mapping phase in this

implementation is related to local scheduling policy, and will be discussed in 4.6.1.5.

Table 1 shows an example of ranking process: For a workflow with tasks A, B, C, D, and a set of resources R1, R2, R3, calculate the weight of each communication and execution. Then calculate the rank $\overline{R}$ of each task, and the tasks are scheduled in non-decreasing order of the rank.



	R1	R2	R3	*avg*
A	5	8	8	7
B	9	13	11	11
C	3	4	5	4
D	7	10	10	9

execution times on different resources

$$\overline{W}(A) = \frac{5+8+8}{3} = 7.$$

$$\overline{R}(A) = max\{\overline{W}(A) + \overline{W}(A,B) + \overline{R}(B), \overline{W}(A)+ \overline{W}(A,C)+\overline{R}(C)\} = max\{7+5+26, 7+3+15\} = 38.$$

	R1->R2	R1->R3	R2->R3	*avg*
A->B	6	4	5	5
A->C	4	2	3	3
B->D	7	4	7	6
C->D	1	1	4	2

data transfer time between different resources

Table 1. HEFT Algorithm Example [6]

- **The advantage of HEFT**

Table 2 shows the comparison of the well known scheduling methods. In the table, the batch mode scheduling is applied to reduce the communication delay, by grouping heavy communication tasks or by minimizing data transmission. However, in this implementation, dependency issue is the focus, and communication time is considered not as significant as the computation time and optimization of communication is omitted. So minimization of communication time is not involved in the model. The complexity of HEFT is comparatively low especially in this implementation since tasks number is not high.

Scheduling Method			Algorithm	Complexity*	Features
Heuristics	Individual task scheduling		Myopic	O(vm)	Decision is based on one task.
Heuristics	List scheduling	Batch mode	Min-min Max-min Sufferage	O(vgm)	Decision based on a set of parallel independent tasks
Heuristics	List scheduling	Dependency mode	HEFT	$O(v^2m)$	Decision based on the critical path of the task
Heuristics	List scheduling	Dependency-batch mode	Hybrid	$O(v^2m+vgm)$	Ranking tasks based on their critical path and re-ranking adjacent independent tasks by using a batch mode algorithm
Heuristics	Cluster based scheduling		TANH	$O(v^2)$	Replicating tasks to more than one resources in order to reduce transmission time
Heuristics	Duplication based scheduling		TANH	$O(v^2)$	Replicating tasks to more than one resources in order to reduce transmission time
Metaheuristics	Greedy randomized adaptive search procedure (GRASP)			guided random search	Global solution obtained by comparing differences between randomized schedules over a number of iteration.
Metaheuristics	Genetic algorithms (GA)			guided random search	Global solution obtained by combining current best solutions and exploiting new search region over generations.
Metaheuristics	Simulated annealing (SA)			guided random search	Global solution obtained by comparing differences between schedules which are generated based on current accepted solutions over a number of iterations, while the acceptance rate is decreased.

** where v is the number of tasks in the workflow, m is the number of resources and g is the number of tasks in a group of tasks for the batch mode scheduling.*

Table 2. Grid Scheduling Algorithms [19]

4.4.5 Relaxed Advance Reservation in SCOJO-PECT: Solution for Local Scheduling Policy Constraint

As introduced in Chapter 3, SCOJO-PECT is a simulation framework for job scheduling. It provides multiple slices for different classes of jobs (short, medium and long). The basic policy is coarse grain time sharing among different job classes, and first come first serve space sharing with conservative backfilling within one job class.

To coordinate with the grid scheduling, reservation is used to guarantee the necessary processor resources for workflow tasks in future. Relaxed reservation is used to alleviate the problem of fragmentation. The model of relaxed advance reservation is described next.

Problem description:

The attribute of task t: EST(t) is the earliest start time of task t, EXC(t) is the execution duration, SST(t) is the scheduled start time. Communication time between task t and its predecessor is COMM(Pred,t). Predicted response time PRT(t,p,s,ST(t)) is the time when task t finishes. LFT(t,p,s,ST(t)) is the latest finish time of task t. A workflow slack portion SLPA is set to calculate the slack ahead of the task in the task schedule. A workflow slack portion SLPB is set to calculate the slack behind the task in the task schedule. MAF is the move-ahead factor for calculating the amount of time the task can move ahead because the jobs ahead of it are non-type backfilled and finish in advance. NBP is the Non-type Backfilling Probability of the job type which the scheduled task belongs to. It is 0.025 for medium jobs and 0.04 for long jobs.

Relaxed reservation schedule: setting attributes of the task t, including processing resources p, EST(t), ST(t), LFT(t,p,s,ST(t)).

Input:

A set of resources R (P, N, S). P is a set of clusters, N is a set of network speed values between any two clusters, and S is a set of scheduling policies on P

Task t in workflow job, and $t \in T$.

The earliest start time of task t, EST(t)

Output:

The relaxed reservation schedule of task t on cluster p which provides min LFT(t,p,s,ST(t)), p follows scheduling policy $s \in S$, given the start time EST(t)

Objective: As in the third phase of HEFT, the grid scheduler schedules the task t in the cluster p which provides min LFT(t,p,s,ST(t)), p follows scheduling policy $s \in S$, given the start time ST(t) .

LFT(t,p,s,EST(t)) is the predicted latest finish time of task t, predicted by cluster $p \in P$, following scheduling policy $s \in S$, given the start time ST(t).

LFT(t,p,s,EST(t)) = PRT(t,p,s, ST(t))+ EXC(t)* SLPB (4)

PRT(t,p,s,ST(t)) is the predicted response time of task t, predicted by cluster $p \in P$, following scheduling policy $s \in S$, given the start time ST(t).

The earliest start time is start time subtracts Slack

EST(t) = ST(t) – EXC(t)* SLPA – MAF* NBP*(ST(t)-CurrentTime) (5)

EST(t) is the earliest start time of task t, and it is the maximum value of the predecessors' latest finish time plus the communication time between this task and the predecessor.

$ST(t) = \max_{\forall(Pred,t)\in C} \{ LFT(Pred) + COMM(Pred, t) \}$ (6)

EXC(t) and COMM(Pred, t) are user inputs, and SLPA, SLPB, NBP and MAF are constants. CurrentTime is the system time of the cluster.

4.4.6 RARS-HEFT Algorithm

RARS-HEFT (Relaxed Advance Reservation SCOJO-PECT HEFT) Algorithm

Input:

Workflow tasks W(T, C), T is a set of tasks, C is a set of communications between tasks

A set of Resources R (P, N, S), P is a set of clusters, N is a set of connections between cluster, S is a set of scheduling policies of the clusters

Output:

A list of scheduled relaxed reservation (earliest start time, start time, latest finish time) for tasks in T

```
compute the average execution time for each $t \in T$ according to (1)
compute the average communication time for each $c \in C$ according to (2)
compute the rank value for each task according to (3)
sort the tasks in a scheduling list L by descending order
While (L is not empty) do
    t = the first task in L
    remove first task in L
    predecessorsList = t.getPredecessors()
    Set MinLatestFinishTime to the largest integer
    Compute ST(t) according to (6)
    For clusterIndex:=1 To P.size() Do
        Call local P[clusterIndex]'s predict function to get PRT(t)
        Compute EST(t) of t according to (5)
        Compute the LFT(t) according to (4)
        If (MinLatestFinishTime > LFT(t))
            MinLatestFinishTime = LFT(t)
            Index = clusterIndex
        End If
        Cluster P[Index] schedules the task t with EST(t), ST(t) and
        MinLatestFinishTime
    End For
End While
```

4.5 Grid Scheduler

Grid scheduler is known as the scheduler that works on top of local schedulers. It makes grid job scheduling decisions based on negotiation with local schedulers, finds the optimized scheduling for grid jobs in a global view, and matches the multiple tasks in grid jobs to the multiple local clusters.

When grid jobs are submitted to the grid scheduler, they are kept in the grid job queue, and based on the necessity of each task in grid job, grid scheduler negotiates with local scheduler. When the negotiation is satisfied, the grid scheduler puts the tasks into the queues of different clusters as seen in Figure 9.

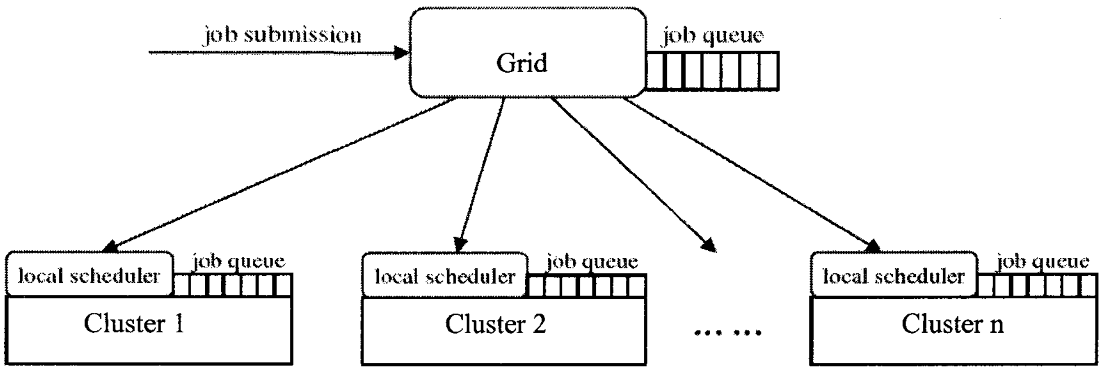


Figure 9. Grid Job Processing from [15]

Task admission:

The grid jobs are submitted to grid scheduler, which is responsible for job admission and job generation.

Resource allocation:

Just-in-time allocation and look-ahead allocation are two different strategies for resource allocation [1]. Just-in-time allocation is usually applicable for the tasks without dependency concerns, while look-ahead allocation requires the scheduler to make decisions to allocate all the tasks in advance [1]. This thesis uses look-ahead allocation for allocating jobs with dependencies and parallelization.

Because grid scheduler needs to decide on which cluster to allocate a job for the best scheduling, job response time prediction is critical. SCOJO-PECT provides a framework for response time prediction, however, it does not support advanced reservation prediction. This implementation adds the prediction functionality for advance reservation.

Task coordination:

The scheduler should be able to coordinate resource request based on tasks' dependencies, parallelization and network communication. So the scheduler needs to make the decision with the least amount of delay caused by resource unavailability and data transfer. To ensure the guaranteed availability of resources in clusters during certain period of time, advance reservation is necessary.

4.6 Prediction of Response Time for Advance Reservation

Prediction of the start time and the response time of individual tasks is very important for grid scheduler to make grid job scheduling decisions, because all the grid tasks need to be scheduled at once before the first task starts. In this implementation, the RARS-HEFT algorithm needs to use the response time prediction as an input during the grid job scheduling, but the original implementation of SCOJO-PECT only supports prediction of response time for non-reserved jobs. In this implementation, modification is made to extend the advance reservation prediction functionality.

Original design of prediction: In Figure 10 A, the two dimensional graph shows the resource usage of one cluster, Job 1 and Job 2 are previous scheduled jobs, two free Resources 1 and 2 are available, and job 3 is waiting for start time estimation. The scheduler will schedule the job as seen in B, Job 3 is scheduled to occupy part of the free

Resource 2 where there are enough processors. Then free Resource 2 is decreased in processors, and free Resource 3 is added.

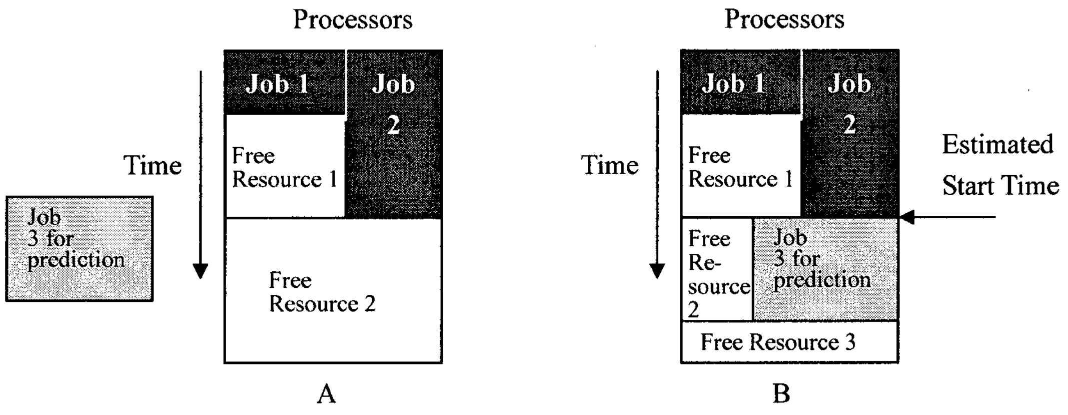


Figure 10. Original Design of Prediction

The original prediction can only predict the job which "attached" to the scheduled jobs, for example, Job 3 is attached to Job 2. However, the prediction of future reservations is not considered. This implementation provides a solution for this problem.

<u>Prediction in this implementation</u>: Figure 11 A is similar to the previous graph but the Job 3 is a reservation which cannot start before its reservation time. Job 3 is scheduled to occupy part of the free Resource 2 with the following 3 steps:

Step 1. Identify the time duration of new free Resources 2 and 3 as in B;

Step 2. Decrease the free processors in free Resource 3 as in C

Step 3. Add free Resource 4 as in D

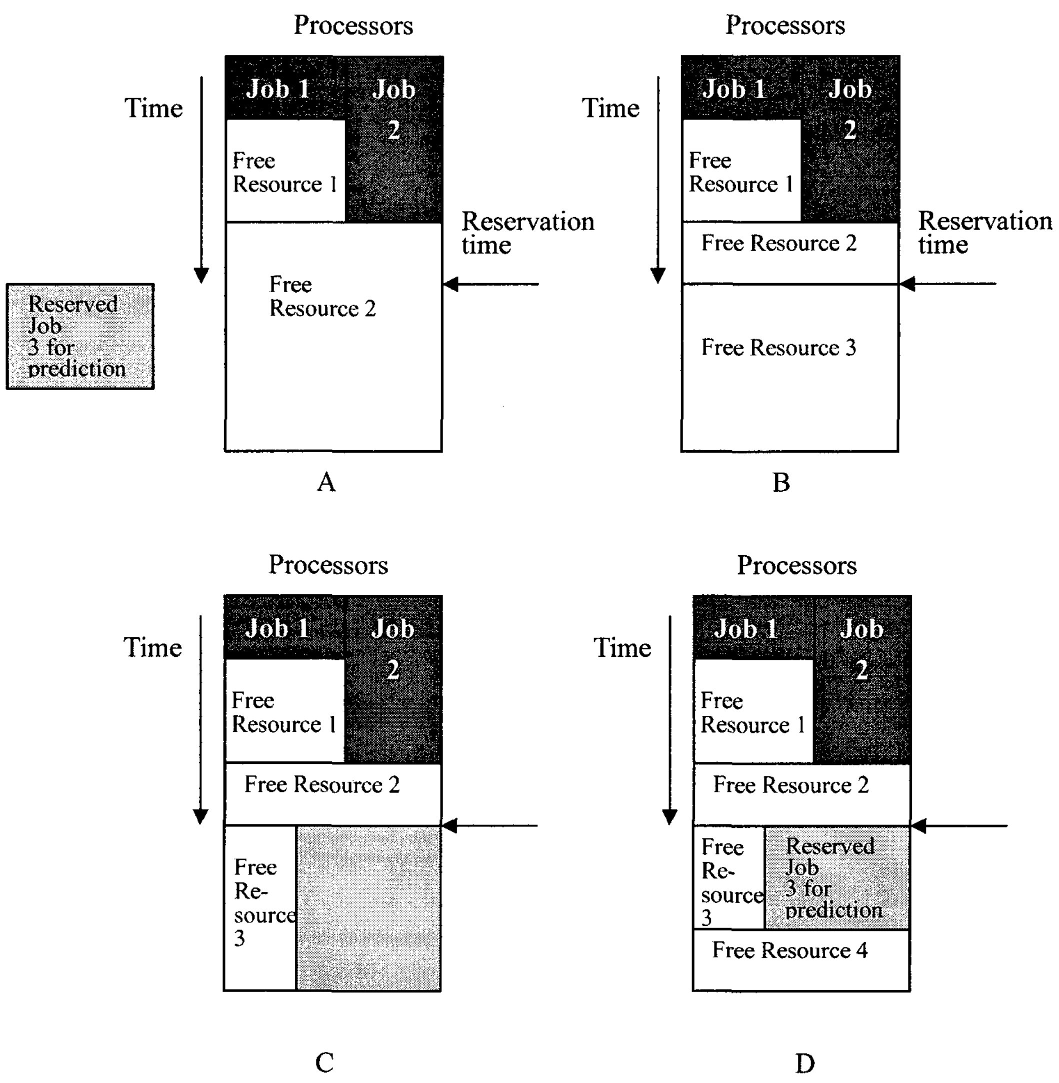


Figure 11. Modified Design of Prediction

4.7 Advantages of Relaxed Reservation in Real Scenarios

The local scheduling policy includes relaxed reservation and coarse grain time share. There are several scenarios which describe the advantages of the cooperative effect of these two elements.

- **Advantage of relaxed reservation when job ahead finish earlier**

The relaxed advance reservation ensures the synchronized scheduling of grid jobs and decreases the fragmentation. It creates many possibilities for different optimizations.

Some examples illustrating the advantages are explained below. In Figure 12 A, there is a relaxed reservation with the red square representing the time frame, the job ahead of the reservation finishes earlier as seen in B, the reserved job can "slide" up to fill the fragmentation as seen in C. Here slide means reschedule with the relaxed time frame. In this scenario, the fragment is filled up by the reservation to increase the system utilization.

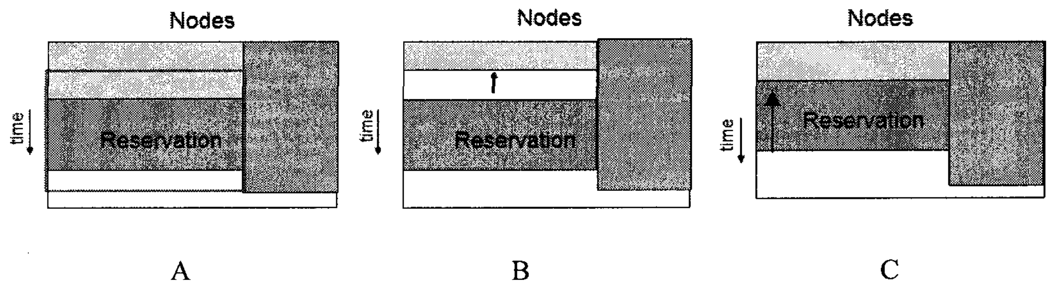


Figure 12. Advantage of Relaxed Reservation When Job Ahead Finish Earlier

- **Advantage of relaxed reservation during the rescheduling optimization with backfilling**

The relaxed reservation provides a good flexibility for maximizing the system utilization. The schedule of reservation is not fixed but can slide up and down, which may create just enough resources for other jobs to backfill. In Figure 13 A, there is a relaxed reservation and a job for backfilling, however, the fragment ahead of the reservation does not have enough resources for the job for backfilling to insert inside. In B, the reservation "slides" down and gives the job for backfilling enough resources to fill the fragmentation. This feature is not implemented in this thesis, but the relaxed reservation mechanism provides sufficient conditions for further extension of rescheduling optimization with backfilling.

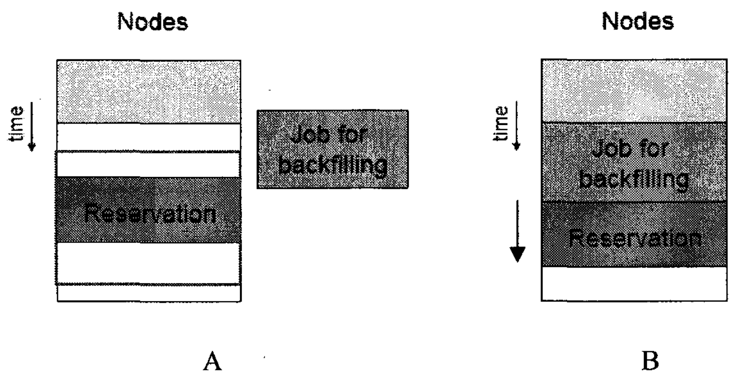
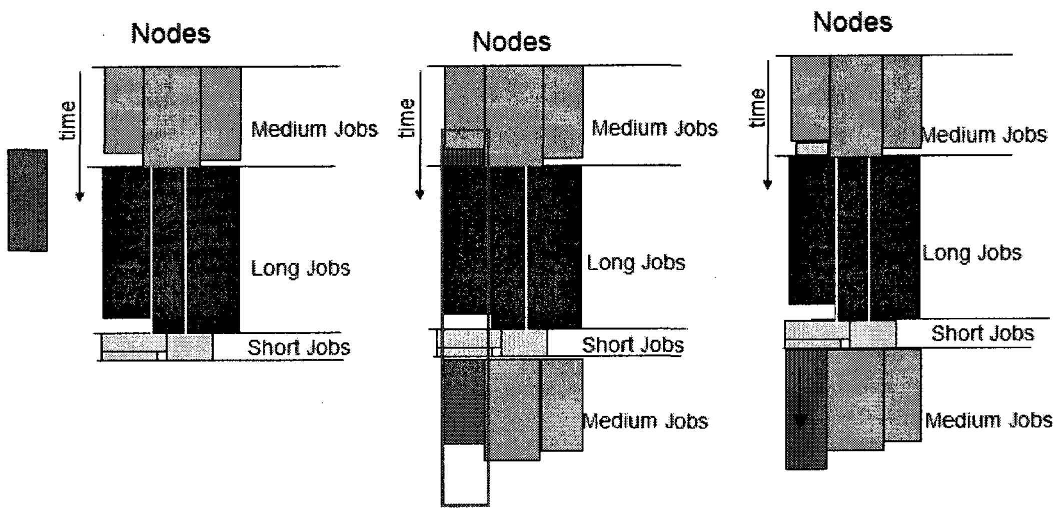


Figure 13. Advantage of Relaxed Reservation during Rescheduling Optimization with Backfilling

- **Advantage of relaxed reservation when preemption is avoided**

Preemption happens when jobs are not finished before the end of slices. In preemption, job information will be written in hard disk and reloaded before the next slice of the same type occurs. Because preemption involves overhead which increases the relative response time of jobs, it is preferable to avoid preemption if possible. In Figure 14 A, there is a red job waiting to be scheduled, while in B the relaxed reservation provides a possibility to reschedule the job in the next medium job type slice, rather than scheduling the job and preempting it right after it starts. In C, the job "slides" down to the next own type slice, so that the preemption cost is avoided and the "given up" available resources can be used for another short job to backfill. In this sense, the relative response time of the read job is decreased. This feature is also not implemented in this thesis, but the relaxed reservation mechanism provides sufficient conditions for further extension of rescheduling in order to avoid preemption.



A B C

Figure 14. Advantage of Relaxed Reservation Which Saves the Preemption Cost

4.8 The Conflict between Makespan Minimization Objective and Local Scheduling Policy

Relaxed reservation is designed to decrease the system fragmentation and increase the utilization; however it creates "spare time" for tasks, which means the tasks need to wait a longer time to start. Therefore, the overall makespan of workflow is extended compared to no "spare time". As seen in Figure 15, the makespan of the workflow job which has tasks without spare times in A is shorter than that in B which has tasks with spare times.

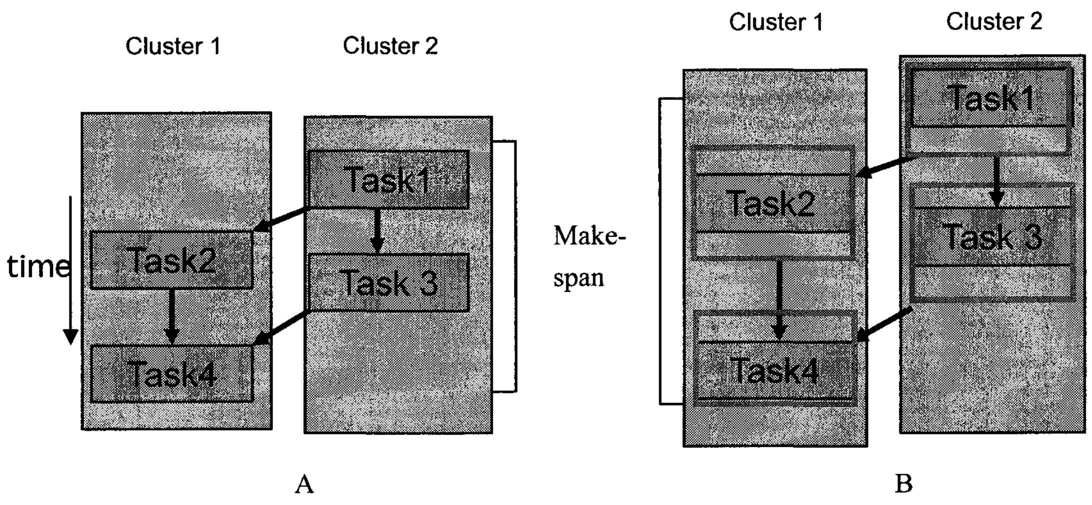


Figure 15. Spare Time in Propagation Heuristics

4.9 Solution for Conflict: Propagation Heuristics

Propagation Heuristics mediates the conflict between relaxed reservation and workflow makespan minimization by tracking in real time the status of tasks and reschedules the tasks when necessary. After the reservation of all the tasks in the workflow, the system keeps on tracking the status of all the tasks. If a task's predecessors have finished, this task's earliest start time will be changed to the current time. When there are enough resources, this task will be rescheduled. As seen in Figure 16, in A, there is an workflow job. In B, when task 1 finishes, the earliest start times of task 2 and task 3 are adjusted accordingly, so that they have the opportunity to be rescheduled earlier, given enough resources. C describes the best scenario that task 2, 3 and 4 are rescheduled earlier by propagation heuristics.

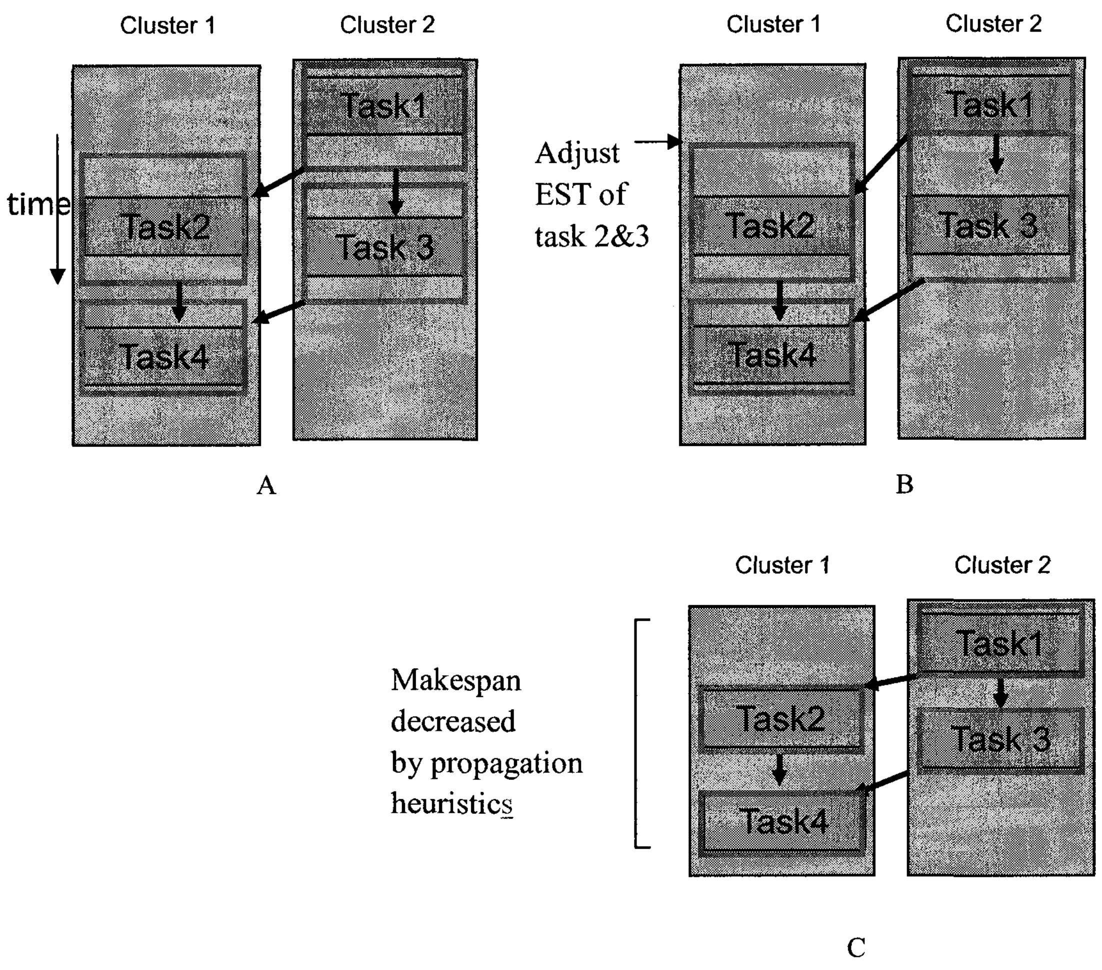


Figure 16. Best Scenario of Propagation Heuristics

Algorithm 2. Propagation Algorithm

```
On the event when task t finishes
  Query task t's dependents $d_i$
  For Each $d_i$
        If all $d_i$'s predecessors $p_j$ has finished
          $EST(d_i) = currentTime + Max((finishTime\ (p_j)) + (communicationTime(d_i, p_j)))$
        End If
   End For
```

4.10 System Design

There were two packages in the original SCOJO-PECT project, *EventBaseSimulator* package defines the basic functionality of event processing, and *HPCSimulator* package consists of the common libraries of SCOJO-PECT including job generation, job queuing, resources allocation, slice switching, statistics and so on. In this thesis, the original design of both packages above has been modified. The *GridSimulation* package is added for grid scheduling and grid jobs implementation.

4.10.1 Design Modifications in *EventBaseSimulator* Package

The new design abstracts *EventProcessResource* from *EBSimulator* and makes *EventProcessResource* a generalized representation of resources process jobs. The *EventProcessResource* is used as the super class of *Cluster* and *Grid.* The new design allows multiple *EventProcessResources* to handle events. Event class has a new *EventProcessResources* attribute to identify which resources the event instance belongs to.

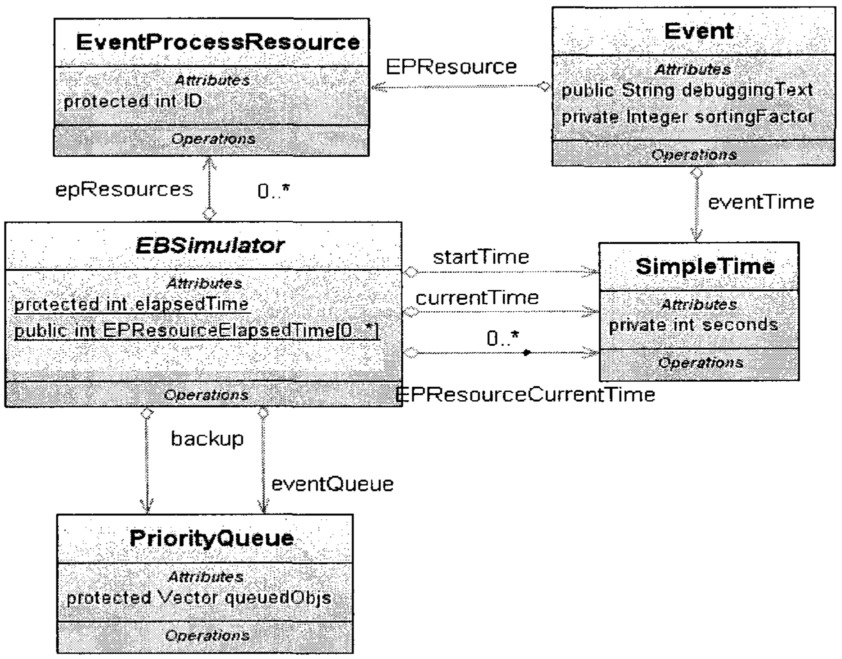


Figure 17. Class Diagram of EventBaseSimulator

4.10.2 Design Modifications in HPCSimulation Package

The new design seperates the original *ClusterSimulator* class into two different classes: *Cluster* class and *Simulator* class. *Cluster* class as a subcluss of *EventProcessResource* is abstraction of resources with functionality such as event handling and resources allocation. *Simulator* as the subclass of *EBSimulator,* is responsible for system initialization, event queue controlling and system time maintainance.

Grid class is added to represent the data structure of grid and functionality of grid scheduler (see Section 4.5). *Grid* and *Cluster* are the two subclasses of *EventProcessResource*, as both of them are deemed as reources but at different levels and provide different functionalities. *Grid* generates and allocates the grid jobs in clusters (workflow jobs and simultatneous jobs), while *Cluster* processes local jobs. The design details are represented in the class diagram as follows.

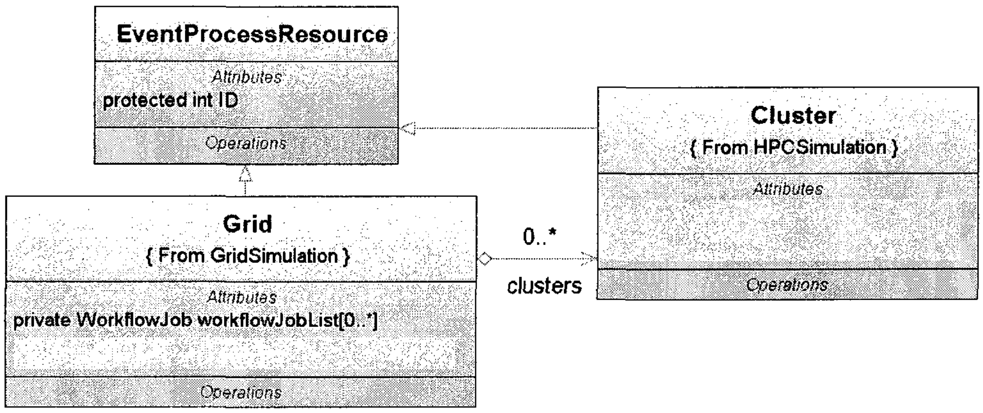


Figure 18. Class Diagram of Resource

One *Simulator* instance has one grid instance and multiple cluster instances, every C*luster* has one *JobScheduler (PreemptionScheduler*) and one *ShareControl.* The design details are represented in the class diagram as follows:

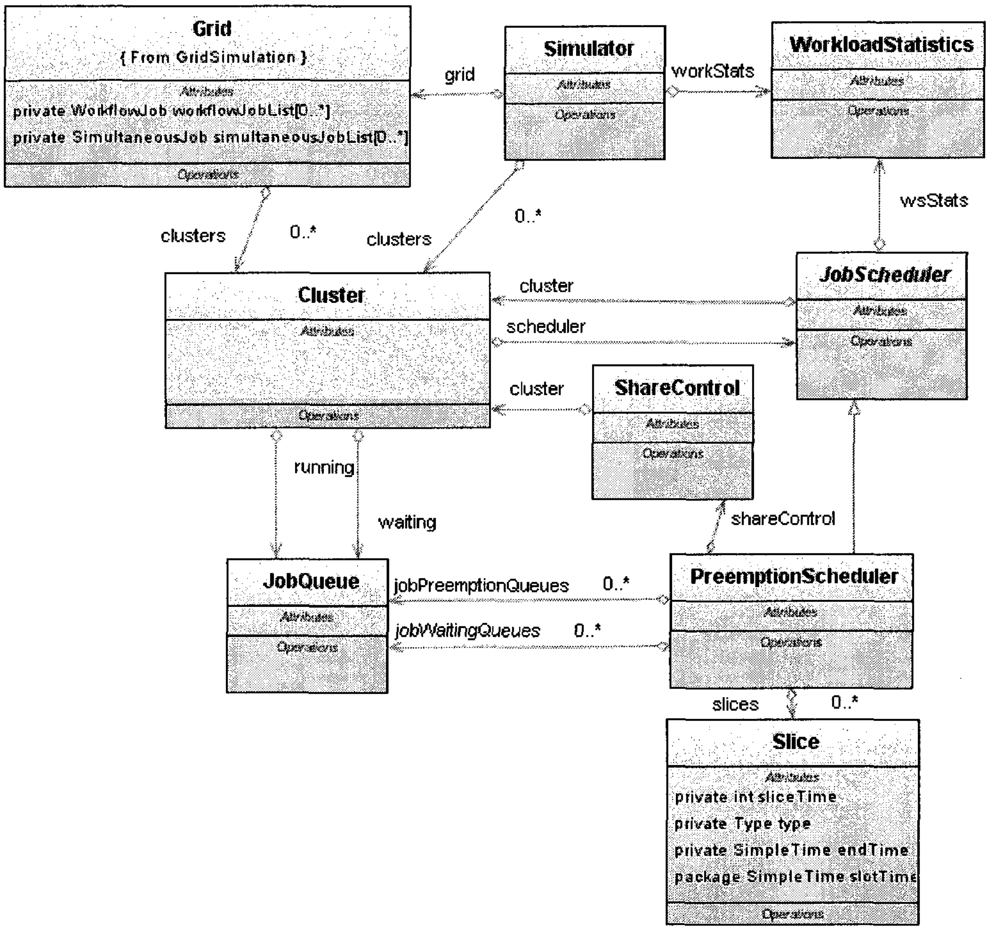
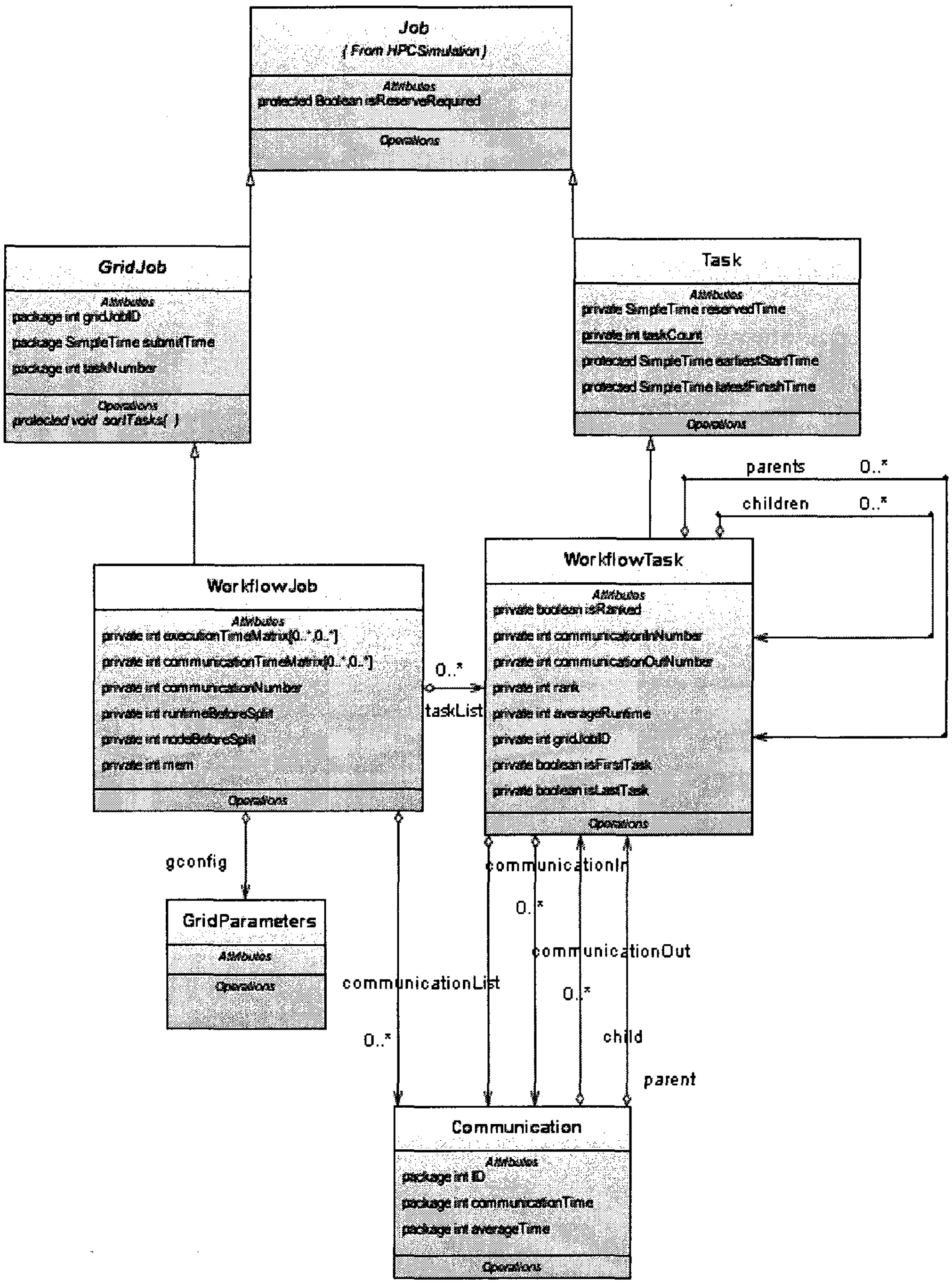


Figure 19. Class Diagram of HPCSimulation

4.10.3 The GridSimulation Package

GridJob class and *Task* class are subclasses of *Job*; *WorkflowJob* is the subclass of *GridJob*; and *WorkflowTask* is the subclass of *Task.* One *WorkflowJob* instance has a list *of WorkflowTask* instances, and a list of *Communication* instances which provide the communication among *WorkflowTasks*. One *WorkflowTask* instance has two lists of *WorkflowTask* as predecessors (*parents*) and successors (*children*). One *WorkflowTask* instance also has two lists of *Communication instances*: *communicationIn* which is the communication between an instance and its parents, and *communicationOut* which is the communication bewteen an instance and its children. *Communication* Class records the two workflow tasks and communciation time between them. The class diagram is given below. The modeling of workflow job is discussed further in Section 4.6.1.



Job
(From HPCSimulation)
Attributes
protected Boolean isReserveRequired
Operations
GridJob
Attributes
package int gridJobID
package SimpleTime submitTime
package int taskNumber
Operations
protected void sortTasks( )
Task
Attributes
private SimpleTime reservedTime
private int taskCount
protected SimpleTime earliestStartTime
protected SimpleTime latestFinishTime
Operations
parents
0..*
children
0..*
WorkflowJob
Attributes
private int executionTimeMatrix[0..*,0..*]
private int communicationTimeMatrix[0..*,0..*]
private int communicationNumber
private int runtimeBeforeSplit
private int nodeBeforeSplit
private int mem
Operations
0..*
taskList
WorkflowTask
Attributes
private boolean isRanked
private int communicationInNumber
private int communicationOutNumber
private int rank
private int averageRuntime
private int gridJobID
private boolean isFirstTask
private boolean isLastTask
Operations
gconfig
GridParameters
Attributes
Operations
communicationIn
0..*
communicationOut
0..*
communicationList
0..*
child
parent
Communication
Attributes
package int ID
package int communicationTime
package int averageTime
Operations

Figure 20 Class Diagram of GridSimulation

4.11 Workload Modeling

4.11.1 Local Job Modeling:

In this implementation, Lublin-Feitelson statistical workload model [34] is used for creating the synthetic workload and for result evaluation. This workload model is derived from real-life workload traces from San-Diego Supercomputer Center, Los-Alamos National Lab, and Swedish Royal Institute of Technology in Stockholm. The workload model is widely used in job scheduling research which includes the connection between size and runtime of parallel jobs and serial jobs, the density of job submission during different times of the day. The job traces represent the workload of a real life system which is suitable for the research in this thesis.

The experiment is scheduled with 20000 jobs. The workload setup is as follows:

Work load	% of Jobs			Average size			Average length			Offered Load
	Short	Med	Long	Short	Med	Long	Short	Med	Long	
Seed 71	64.0	19.4	16.6	8.7	16.5	19.7	79	5757	19200	3.08
Seed 13	63.9	19.4	16.7	8.7	16.7	20.6	79	5821	19100	3.05
Seed 3	64.2	19.8	16.0	8.7	17.2	20.1	79	5763	19087	3.14
Seed 31	64.0	19.8	16.2	8.8	16.6	20.5	79	5829	18954	2.96

Table 3. Experiment Workload Setup

The scheduler follows the following parameters:

Interval time	3600 seconds
Share for short slices	Dynamic, but less than 600 seconds
Share for medium slices	30% relative share of one interval
Share for long slices	70% relative share of one interval
Preemption overhead	60 seconds
Short job duration	Less than 600 seconds
Medium job duration	Less than 3 hours
Long job duration	Larger than 3 hours

Table 4. Scheduler Parameters

4.11.2 Grid Job Modeling:

Workflow jobs in this thesis are not included in the Lublin-Feitelson statistical workload model, therefore this implementation modifies job generation to fit the requirement. The jobs, which have runtime larger than 6000 and smaller than 60000 and processor requirement larger than 10, will be available for the modification. Certain percentage (depends on the configuration) of the modification available jobs will be split into workflow which contains 5 tasks, Figure 21 shows the splitting details.

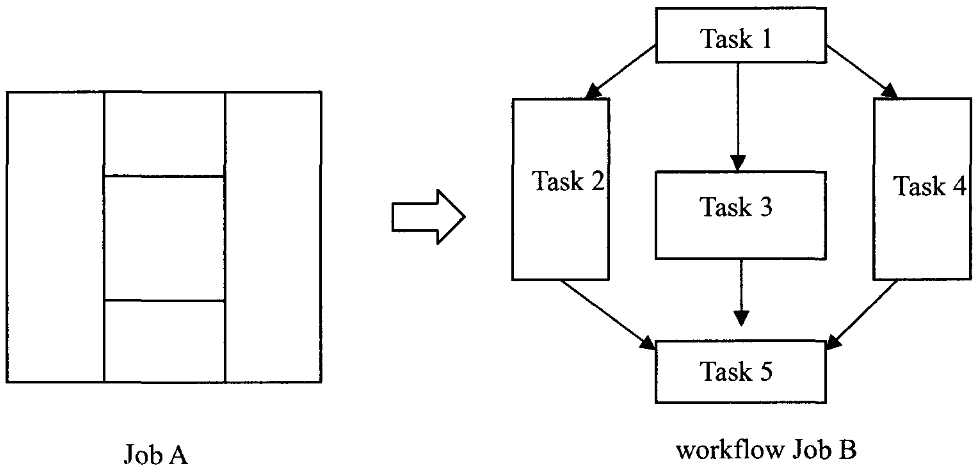


Figure 21Generate Workflow Job B by Splitting Simple Job A

In Figure 21, job A was partitioned into grid job B with 5 tasks. The workloads of job A and the grid job B are the same. Task 1 is the first task to run, and when task 1 finishes, task 2, 3, and 4 (after communication with task 1) start and run in parallel; finally task 5 starts after the completion of its predecessors tasks 2, 3 and 4 and communication between task 5 and the predecessors . The communication time among tasks are generated randomly from a range of 20 seconds to 600 seconds. Because the testing in this thesis is not focused on the workflow job model, the level of the workflow graph L is assumed to be 3, and the branching factor BF is assumed to be 5/6.

Furthermore, two matrices are generated: The *communicational matrix* represents the data transfer time of every communication in workflow graph over every combination of two clusters. The *computational matrix* represents the execution time of every task in workflow graph in every cluster.

The simplification of grid job model is to maintain the stability of Lublin-Feitelson statistical workload model, and to reach the predictable workload and evaluation statistics.

5. Experiments and Results Analysis

5.1 Testing Plan

The testing has been done to identify the advantages of relaxed reservation and propagation heuristics.

The following criteria have been tested:

- Local job (long and medium) average relative response time
- Grid job (long and medium) average relative response time
- Grid job makespan

The testing has been conducted for different values of :

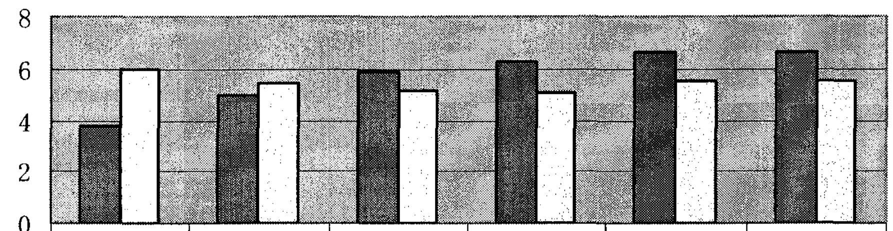
- Amount of slack in time frame
- Percentage of grid jobs
- Propagation heuristics – enabled or disabled.

Because short jobs are not affected by the grid tasks which are long and medium jobs and which only occupy long and mediums slices, so the result analysis of short jobs is omitted

5.2 Experiment Result

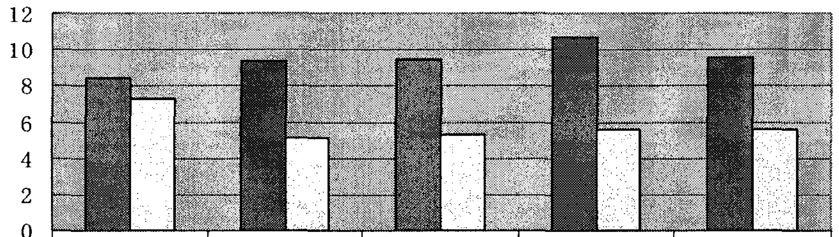
The definitions of terms used in the following figures:

RR-Medium: Average relative response time of medium jobs.
RR-Long: Average relative response time of long jobs.
RR-Long-Grid: Average relative response time of long grid tasks.
RR-Medium-Grid: Average relative response time of medium grid tasks.
71T: Propagation heuristics use is set to True and job generation seed is 71.
71F: Propagation heuristics use is set to False and job generation seed is 71.
31T: Propagation heuristics use is set to True and job generation seed is 31.
31F: Propagation heuristic is set to False and job generation seed is 31.
3T: Propagation heuristic is set to True and job generation seed is 3.
3F: Propagation heuristic is set to False and job generation seed is 3.
13T: Propagation heuristic is set to True and job generation seed is 13.
13F: Propagation heuristic is set to False and job generation seed is 13.



	0%	7%	13%	18%	22%	26%
▨ RR-Medium	3. 8125	4. 9275	5. 8925	6. 2425	6. 58	6. 645
☐ RR-Long	5. 965	5. 44	5. 1175	5. 0475	5. 5475	5. 53

Figure 22. Average RR of the Clusters According to the Percentage of Task Reservations in All Jobs



	7%	13%	18%	22%	26%
▨ RR-Medium-Grid	8. 455	9. 395	9. 435	10. 7	9. 5875
☐ RR-Long-Grid	7. 27	5. 1475	5. 28	5. 5675	5. 555

Figure 23. Average RR of Grid Tasks in the Clusters According to the Percentage of Task Reservations in All Jobs

Figure 22 shows the average relative response time of jobs in the clusters including both local jobs and grid jobs for different values of the percentage of task reservations. In the tests, we use the jobs with runtime between 6000 and 60000, and pick 0%, 20%, 40%, 60%, 80%, 100% of these jobs to generate grid tasks. The percentage of task reservations presented in the Figures (0%, 7%, 13%, 18%, 22%, 26%) is the total number of task reservations divided by the total number of medium and long jobs including the task reservations. The job generation seed is set to 71. The time frame portion SLPA is 1, SLPB is 0, MAF is 4 and propagation heuristics is disabled. The result implies that the relative response time of medium jobs processed in the clusters increases with an increase in percentage of grid tasks. This is because the higher percentage of reservation causes higher amount of fragmentation, which makes it more difficult for jobs to get enough resources, so waiting for sufficient resources to be available extends their response time.

However, unlike medium jobs, the relative response time of long jobs does not increase because the long runtime of long jobs undermines the change in response time.

Figure 23 shows the average relative response time of grid tasks in the clusters according to task reservations. Similarly, the result here leads to the conclusion that the relative response time of medium grid tasks increases with an increase in percentage of grid job tasks. The time frame portion SLPA is 1, SLPB is 0, MAF is 4 and propagation heuristics is disabled.

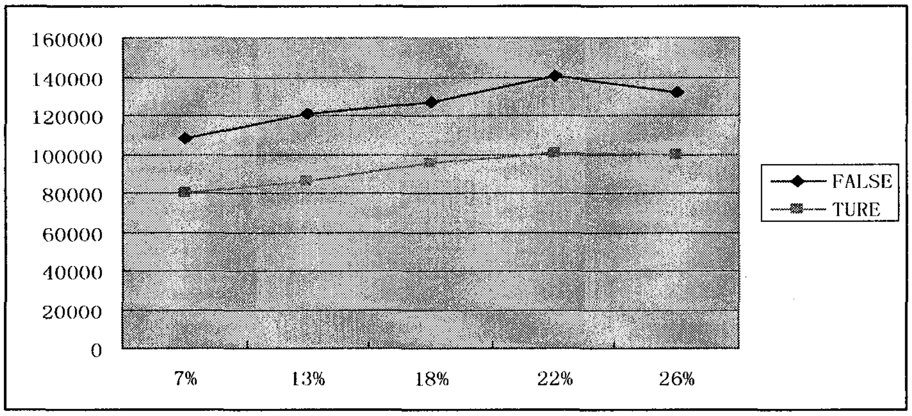
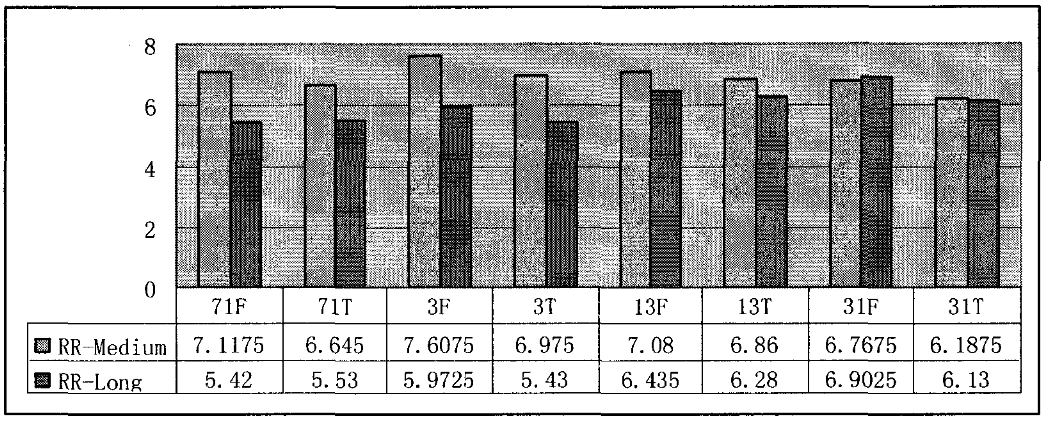


Figure 24. Average Makespan (in seconds) of Grid Jobs According to the Percentage of Task Reservations in All Jobs. TRUE and FALSE refer to the enabling and disabling of propagation heuristics, respectively.

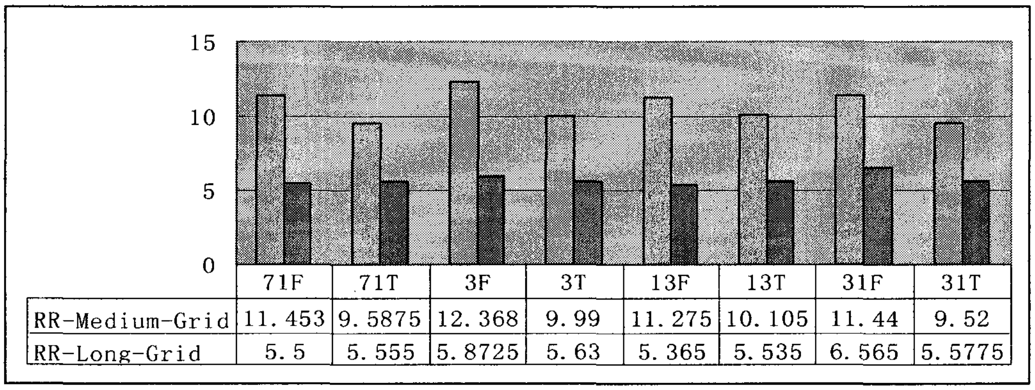
Figure 24 shows the average makespan of grid jobs with respect to the task reservation percentage, and in the presence of absence of the propagation heuristics. Time frame portion SLPA is 1, SLPB is 0, and MAF is 4. Based on this result, the makespan of grid jobs increases with an increase in the percentage of grid jobs. The reason is the same as the analysis for Figure 22. Furthermore, the makespan is lower when propagation heuristics is enabled, which means the propagation heuristics is helpful for minimizing the makespan of workflow jobs. This experiment shows that the propagation heuristics algorithm significantly decreases the workflow makespan by approximately 20%.

Figures 25 and 26 show average relative response time of jobs in the clusters, when the job generation seeds are set to 71, 3, 31, 13, and propagation heuristic is enabled or disabled. Time frame portion SLPA is 1, SLPB is 0, and MAF is 4.The graphs clearly show that when the propagation heuristic used, the relative response time is lower for both medium and long jobs.



	71F	71T	3F	3T	13F	13T	31F	31T
▨ RR-Medium	7.1175	6.645	7.6075	6.975	7.08	6.86	6.7675	6.1875
■ RR-Long	5.42	5.53	5.9725	5.43	6.435	6.28	6.9025	6.13

Figure 25. Average RR of Local Clusters with Respect to Presence or Absence of Propagation



	71F	71T	3F	3T	13F	13T	31F	31T
RR-Medium-Grid	11.453	9.5875	12.368	9.99	11.275	10.105	11.44	9.52
RR-Long-Grid	5.5	5.555	5.8725	5.63	5.365	5.535	6.565	5.5775

Figure 26. Average RR of Grid Tasks in Clusters with Respect to Presence or Absence of Propagation

Figure 27 shows the average makespan of grid jobs, when the job generation seeds are set to 71, 3, 13, 31, and propagation heuristics is set to present or absent. The time frame portion SLPA is 1, SLPB is 0, and MAF is 4. It clearly shows that the propagation heuristics decreases the makespan of grid jobs.

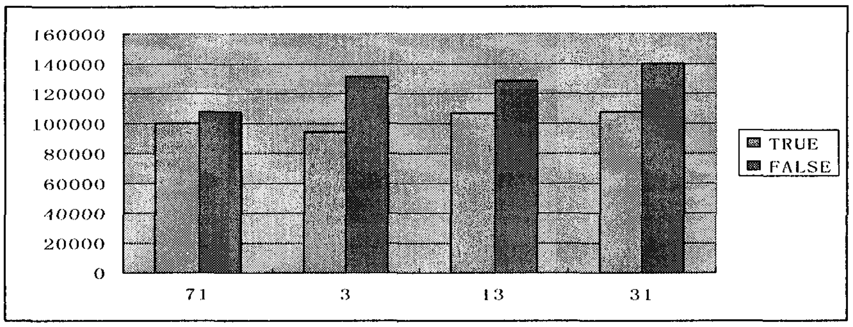
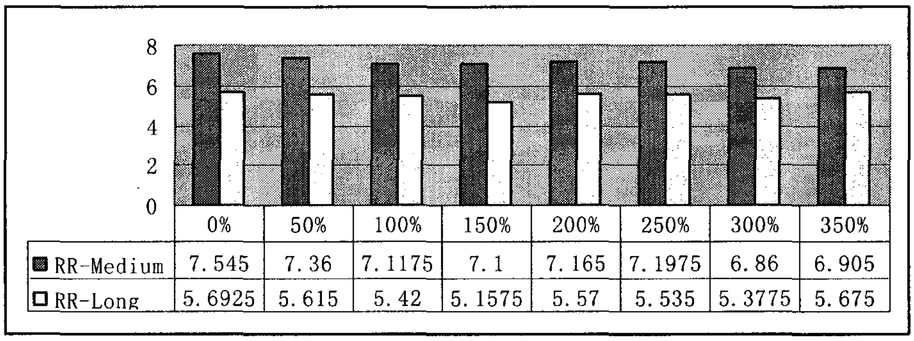


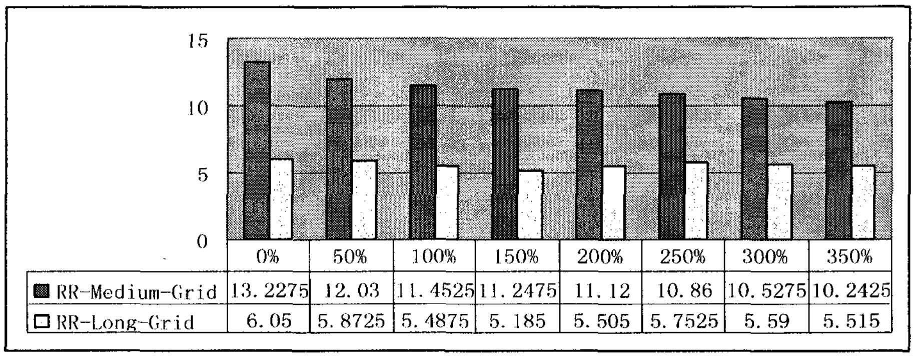
Figure 27. Average Makespan(in seconds) of Grid Jobs According to Propagation Heuristic, TRUE and FALSE *refer to the enabling and disabling of propagation heuristics, respectively*

Figures 28 and 29 show, that if the propagation heuristics is enabled, the RR of overall medium jobs and RR of medium grid tasks decrease as the portion of time frame SLPA of reservation increases from 0% to 350%. SLPB is 0, and MAF is 0, and the propagation heuristics is enabled. This decreasing trend indicates that because of the proper amount of slack in relaxed reservation, medium grid jobs have shorter response time when they "slide" up in the time frame to fill the fragmentation. However, it is hard to identify the trend of RR of long jobs because the long runtime undermines the change in response time.



	0%	50%	100%	150%	200%	250%	300%	350%
▩ RR-Medium	7.545	7.36	7.1175	7.1	7.165	7.1975	6.86	6.905
□ RR-Long	5.6925	5.615	5.42	5.1575	5.57	5.535	5.3775	5.675

Figure 28. Average Job RR of 4 Clusters According to Reservation Time Frame Portion



	0%	50%	100%	150%	200%	250%	300%	350%
■ RR-Medium-Grid	13. 2275	12. 03	11. 4525	11. 2475	11. 12	10. 86	10. 5275	10. 2425
□ RR-Long-Grid	6. 05	5. 8725	5. 4875	5. 185	5. 505	5. 7525	5. 59	5. 515

Figure 29. Average Gird Task RR of 4 Clusters According to Reservation Time Frame Portion

6. Summary and Conclusion

In this thesis, the grid job scheduling with reservations and preemption is presented. The grid jobs which contain multiple tasks and communications between tasks are allocated to the clusters which provide earliest response time, so that the objective of minimizing makespan of workflow job is attained. HEFT algorithm, a grid job scheduling algorithm, is used to ensure the dependencies between tasks by sorting them and scheduling them in order.

Moreover, this implementation inherits the advantages of original design of SCOJO-PECT scheduling framework: ensuring the fairness of scheduling different class of jobs, and ensuring the predictability of job scheduling. The advance reservation is used to insure workflow task synchronization. To avoid the fragmentation and system utilization loss caused by advance reservation, the relaxed reservation is applied which provides certain flexibility for reservations to "slide" up and down to fill fragmentation and decrease its response time.

Furthermore, a propagation heuristics algorithm is used to alleviate the workflow job makespan extension caused by the slack of relaxed reservation, and it also provides enough opportunities for a task to reschedule to a time earlier than its original schedule.

As predicted, the experiments show that the relaxed reservation plays a detectable role in decreasing overall job relative response time for medium grid tasks. However, to get the right amount of slack, further research and testing is required

Furthermore, experiments indicate that the propagation heuristics algorithm performs very well by significantly decreasing the workflow makespan by approximately 20%. The propagation heuristics also brings the advantages of lowering the relative response time of both medium and long jobs in clusters.

When the reservation runtime is less than 60000 seconds, most of the time, the system utilization decreases less than 2%. However, occasionally the system utilization decreases around 4%. Experiments also showed that the same percentage of reservations with runtimes longer than 60000 seconds adversely affects the utilization. The degradation of utilization means that the system is overcommitted. It is likely that there is too much fragmentation for large percentages of reservations, and the percentage of reservations should be lower than 26%. The experiments show that if the percentage of reservations is 13% or less, then the system utilization is not adversely affected. For reservation percentages of more than 13%, it may already help if the reservation workload is better balanced among the clusters since we found only one out of the four clusters having decreased utilization. The investigation for this problem needs further experimentation and analysis.

One of the significant contributions of this thesis is the use of the combination of HEFT grid scheduling algorithm, relaxed reservation, and SCOJO-PECT coarse grain time share framework to schedule grid jobs. Also, the novel idea of propagation heuristics was introduced in this thesis to further improve the grid job scheduling performance. Further work is needed to explore an optimal solution to set the slack and to alleviate the system utilization degradation.

7. Future Work

Grid scheduling of workflow job tends to increase the system capability and performance of resource demanding jobs, and thus improves the quality of service of the system. A good quality of service is highly desirable, and its metrics include many concerns other than minimizing the makespan, such as financial cost of services, deadlines guarantees, data accessibility and so on.

The partitioning of grid jobs into tasks in this implementation is provided by the users. However, in real life scenarios, smart partitioning can minimize the communication time between tasks. When the tasks are scheduled in the same cluster their communication time would be very small and can be ignored compared to the inter-cluster communication. But when the tasks are scheduled in different clusters, the communication between remote sites takes much more time. Therefore, partitioning the tasks and keeping the frequently communicating ones in the same cluster is another topic of interest.

In this thesis, the workflow job is modeled with the concern of dependencies between tasks; however, in real life workflow, there might be parallelization concerns. The parallelization means multiple tasks might be required to run at the same time. The workflow job can have both dependency constraints and parallelization constraints among tasks; considering this combination, the modeling and implementation would need further exploration.

The real life grid jobs might be dynamic jobs, the workflow graph might be dynamically changing based on the result of tasks. This unpredictable behavior of the dynamic workflow also provides more possibilities for further research in non-deterministic behavior of workflow, and scheduling based on this behavior.

References

[1] William M. Jones and Walter B. Ligon III, "Ensuring Fairness Among Ensuring Fairness Among Participating Clusters During Multi-site Parallel Job Scheduling," ICPADS, Proceedings of the 12th International Conference on Parallel and Distributed Systems, Volume 2, 2006, pp. 109 - 114

[3] Marek Wieczorek, Stefan Podlipnig, Radu Prodan, Thomas Fahringer, "Bi-criteria Scheduling of Scientific Workflows for the Grid," CCGRID, 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008, pp. 9-16

[2] Yonghong Yan, Barbara Chapman, "Scientific Workflow Scheduling in Computational Grids – Planning, Reservation, and Data/Network-Awareness," International Conference on Grid Computing, Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, 2007, pp. 18-25

[4] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems," 2002, pp. 153-183.

[5] Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour, "Economic Scheduling in Grid Computing", Springer Berlin, Job Scheduling Strategies for Parallel Processing, 2002, pp. 128-152

[6] Marek Wieczorek, Mumtaz Siddiqui, Alex Villazon, Radu Prodan, Thomas Fahringer, "Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid," e-science, Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), 2006, pp.82

[7] Henan Zhao and Rizos Sakellariu, "Advance Reservation Policies for Workflows," Springer Berlin, Book Job Scheduling Strategies for Parallel Processing, 2007, pp. 47-67

[8] W. Smith, I. Foster, and V. Taylor, Scheduling with Advanced Reservations, Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, 2000, pp. 127-132.

[9] Umar Farooq, Shikharesh Majumdar, Eric W. Parsons, "A Framework to Achieve Guaranteed QoS for Applications and High System Performance in Multi-Institutional Grid Computing," ICPP, 2006 International Conference on Parallel Processing (ICPP'06), 2006, pp. 373-380

[10] Neena R. Kaushik, Silvia M. Figueira, Stephen A. Chiappari, "Flexible Time-Windows for Advance Reservation Scheduling," mascots, 14th IEEE International Symposium on Modeling, Analysis, and Simulation, 2006, pp.218-225

[11] Henan Zhao and Rizos Sakellariu, "Advance Reservation Policies for Workflows," Springer Berlin, Book Job Scheduling Strategies for Parallel Processing, 2007, pp. 47-67

[12] Sumit Naiksatam and Silvia Fiueira, "Elastic Reservations for Efficient Bandwidth Utilization in LambdaGrids," Future Generation Computer Systems 23, Elsevier, 2007, pp. 1-22.

[13] Angela C. Sodan, Chintan Doshi, Lawrence Barsanti, and Darren Taylor. Gang Scheduling and Adaptive Resources Allocation to Mitigate Reservation Impact. IEEE CCGRID, Singapore, May 2006.

[14] Angela C. Sodan and *Garima Gupta. Time vs. Space Adaptation with ATOP-Grid. Proc. ACM Workshop on Adaptive and Reflective Middleware (ARM), Melbourne, Australia, Nov. 2006.

[15] M. Siddiqui, A. Villazon, and T. Fahringer, "Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS," Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference, 2006, pp. 21-21

[16] Bryan Esbaugh and Angela C. Sodan. "Preemption and Share Control in Parallel Grid Job Scheduling," CoreGrid Workshop on Grid Middleware (in conjunction with ISC), Dresden, June, Springer, 2007

[17] A.I.D. Bucur, D.H.J. Epema, "Trace-Based Simulations of Processor Co-Allocation Policies in Multiclusters," HODC,12th IEEE International Symposium on High Performance Distributed Computing , 2003, pp.70

[18] Lionel Eyraud-Dubois, Gregory Mounie, Denis Trystram, "Analysis of Scheduling Algorithms with Reservations," IPDPS, 2007 IEEE International Parallel and Distributed Processing Symposium, 2007, pp.114

[19] Jia Yu, Rajkumar Buyya, Workflow Scheduling Algorithms for Grid Computing, Springer Berlin / Heidelberg, Volume 146/2008, Metaheuristics for Scheduling in Distributed Computing Environments, pp. 173-214

[20] Joerg Decker, Joerg Schneider, "Heuristic Scheduling of Grid Workflows Supporting Co-Allocation and Advance Reservation," ccgrid, Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), 2007, pp. 335-342

[21] L. He, S.A. Jarvis, D.P. Spooner, D. Bacigalupo, G. Tan, G.R. Nudd, "Mapping DAG-based applications to multiclusters with background workload," ccgrid, vol. 2, Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2, 2005, pp. 855-862

[22] Zhifeng Yu, Weisong Shi, "An Adaptive Rescheduling Strategy for Grid Workflow Applications," IPDPS, 2007 IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 115

[23] Yingchun , Xiansong Li, Chenxia Sun, "Cost-effective Heuristics for Workflow Scheduling in Grid Computing Economy," GCC, Sixth International Conference on Grid and Cooperative Computing (GCC 2007), 2007, pp. 322-329

[24] A Sulistio, W Schiffmann, R Buyya, “Advanced Reservation-based Scheduling of Task Graphs on Clusters,” High performance computing , 2006, International Conference on High Performance Computing N°13, Bangalore , INDE (2006) , vol. 4297, pp. 60-71

[25] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, Alain Roy, “A Distributed Resources Management Architecture that Supports Advance Reservation and Co-Allocation,” Quality of Service, 1999. IWQoS '99. 1999 Seventh International Workshop, 1999, pp. 27-36

[26] Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews, San Diego Supercomputer Center, “Co-scheduling with User-Settable Reservations,” Springer-Verlag Berlin Heidelberg 2005, Volume 3834/2005, pp. 146-156

[27] Jove M. P. Sinaga, Hashim H. Mohamed, Dick H. J. Epema, “A Dynamic Co-allocation Service in Multicluster Systems.” JSSPP 2004, pp. 194-209

[28] Hashim H. Mohamed, Dick H. J. Epema, “The Design and Implementation of the KOALA

Co-allocating Grid Scheduler," EGC 2005, pp. 640-650

[29] O. Sonmez, H.H. Mohamed, and D.H.J. Epema. "Communication-Aware Job Placement Policies for the KOALA Grid Scheduler." Proc. of the Second IEEE International Conference on e-Science and Grid Computing, 2006, pp. 79–87

[30] V. V. Toporkov, "Flow and greedy algorithms of resources co-allocation in distributed systems," Journal of Computer and Systems Sciences International, Issue Volume 46, Number 2 / April, 2007, pp. 269-278

[31] Jinhui Qin, Michael A. Bauer, "An Improved Job Co-Allocation Strategy in Multiple HPC Clusters," HPCS, 21st International Symposium on High Performance Computing Systems and Applications (HPCS'07), 2007, pp. 18

[32] Angela C. Sodan. Autonomic Share Allocation and Bounded Prediction of Response Times in Parallel Job Scheduling for Grids. Workshop on Adaptive Grid Computing (NCA-AGC) of IEEE Int. Symp. on Network Computing and Applications (NCA), Cambridge, July 2008.

[33] S. Ostermann, R. Prodan, T. Fahringer, A. Iosup, D. Epema, On the characteristics of Grid Workfloww, CoreGrid TR-0132, 2008

[34] U. Lublin, D.G. Feitelson, The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs, Journal of Parallel and Distributed Computing, 2003, pp. 1105-1122.

Vita Auctoris

Xiaorong Cao was born in Nov. 1976 in China. She went to Huazhong University of Science and Technology in Wuhan, China, where she obtained a degree in Bachelor of Engineering. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and will graduate in Summer 2009.