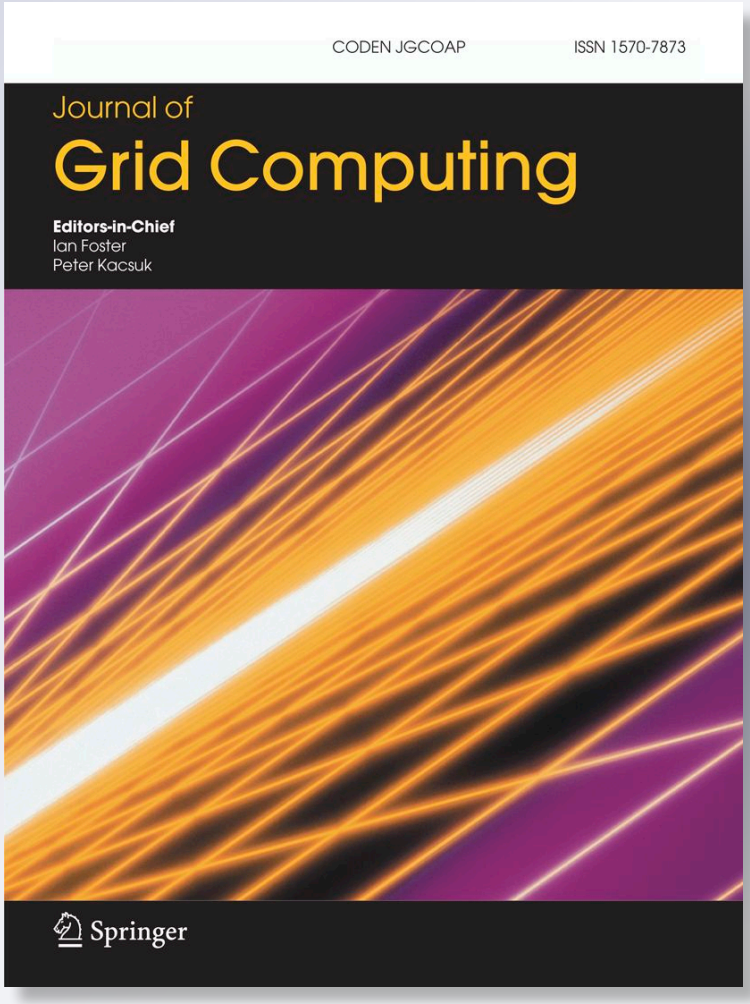
Strategies for Rescheduling Tightly-Coupled Parallel Applications in Multi-Cluster Grids

H. A. Sanjay & Sathish S. Vadhiyar

Journal of Grid Computing

ISSN 1570-7873
Volume 9
Number 3

J Grid Computing (2011) 9:379-403
DOI 10.1007/s10723-010-9170-z



Springer

J Grid Computing (2011) 9:379–403
DOI 10.1007/s10723-010-9170-z

Strategies for Rescheduling Tightly-Coupled Parallel Applications in Multi-Cluster Grids

H. A. Sanjay · Sathish S. Vadhiyar

Received: 1 February 2010 / Accepted: 26 October 2010 / Published online: 9 November 2010

Abstract As computational Grids are increasingly used for executing long running multi-phase parallel applications, it is important to develop efficient rescheduling frameworks that adapt application execution in response to resource and application dynamics. In this paper, three strategies or algorithms have been developed for deciding when and where to reschedule parallel applications that execute on multi-cluster Grids. The algorithms derive rescheduling plans that consist of potential points in application execution for rescheduling and schedules of resources for application execution between two consecutive rescheduling points. Using large number of simulations, it is shown that the rescheduling plans developed by the algorithms can lead to large decrease in application execution times when compared to executions without rescheduling on dynamic Grid resources. The rescheduling plans generated by the algorithms are also shown to be competitive when compared to the near-optimal plans generated by brute-force methods. Of the algorithms, genetic algorithm yielded the most efficient rescheduling plans with 9–12% smaller average execution times than the other algorithms.

Keywords Rescheduling · Parallel applications · Multi-phase · Rescheduling plans

1 Introduction

Grids have been found to be powerful research-beds for executing various kinds of parallel applications [1, 2]. Due to the dynamic nature of Grid environments in terms of varying availability and performance of resources, there has been increasing interest in recent years to develop efficient rescheduling mechanisms that adapt application execution in response to change in resource and application characteristics [3–8]. Rescheduling involves changing the resource set on which an application is executing.

One of the primary challenges in building a rescheduling framework involves developing policies for deciding when and where to reschedule the applications. Existing efforts on rescheduling policies consider simple classes of parallel applications [7] and homogeneous applications with uniform behavior throughout application execution [3, 4]. Most of the existing efforts use dynamic or online profiling techniques to monitor perfor-

This work is supported by Department of Science and Technology, India. project ref. no. SR/S3/EECE/59/2005/8.6.06.

H. A. Sanjay · S. S. Vadhiyar (✉)
Supercomputer Education and Research Centre,
Indian Institute of Science, Bangalore, India
e-mail: vss@serc.iisc.ernet.in

H. A. Sanjay
e-mail: sanjay@rishi.serc.iisc.ernet.in

mance of the application processes and reschedule the processes on performance degradations [6–9]. Online profiling of application characteristics during application execution can incur large execution overheads.

In this paper, we focus on developing rescheduling policies for parallel applications with multiple phases of computation and communication characteristics. In these multi-phase applications, the execution times of the different phases on even a dedicated set of resources are significantly different. The phases can either be qualitatively different phases inherent to the application or can be iterations of an iterative application in which the execution characteristics of the different iterations are different.

The work derives *rescheduling plans* for executing the multi-phase applications on multi-cluster Grids based on the different performance characteristics of different application phases. A rescheduling plan consists of potential points in application execution for rescheduling and schedules of resources for application execution between two consecutive rescheduling points. The plan is built for a specific set of resource characteristics and considers change in application behavior between different phases. The best schedule for one phase in the plan can be different from the best schedule of another phase due to application dynamics. On Grids with high load dynamics or high variability in resource characteristics, these plans can be updated periodically during application execution, thus also considering resource dynamics. Thus, application adaptation to both resource and application dynamics are considered in this paper. To our knowledge, this is the first work on algorithms to determine points of rescheduling for multi-phase parallel applications considering both application and resource dynamics.

Three algorithms were developed, namely an incremental algorithm, a divide-and-conquer algorithm and a genetic algorithm, for deriving a rescheduling plan for a parallel application execution. The plans are derived using the performance modeling strategies, developed in a previous work [10], that predict execution times of parallel applications for dedicated or non-dedicated resources. The algorithms use performance model functions built for a single homogeneous dedicated or non-dedicated cluster to derive rescheduling plan for the cluster. An algorithm, that uses rescheduling plans derived on different clusters to form a single coherent rescheduling plan for application execution on a Grid consisting of multiple clusters, was also developed. Using large number of simulations with five complex scientific multi-phase parallel applications, it is shown that the rescheduling plans generated by the algorithms result in large reductions in application execution times when compared to executions on single schedules on dynamic multi-cluster Grids. The genetic algorithm gave the most efficient rescheduling plans with 9–12% smaller average execution times than the other algorithms. The rescheduling plans generated by the algorithms are also highly competitive when compared to the near-optimal plans generated by brute-force methods.

In one of our previous efforts [10], performance modeling strategies were developed for predicting the execution times of parallel applications on non-dedicated homogeneous clusters of machines. Another previous work [11], discussed scheduling algorithms that use the predictions by the performance models of the earlier effort to choose a set of resources in a non-dedicated cluster for execution of a single-phase parallel application with uniform computation and communication characteristics throughout its execution. The focus of the current work is to propose rescheduling algorithms for adaptive execution of multi-phase parallel applications on single and multiple clusters. Specifically, given a parallel application with multiple phases of computation and communication, three algorithms are first proposed for determining a rescheduling plan for adaptive execution on a single non-dedicated cluster. Another algorithm is then proposed for using the rescheduling plans derived for a single cluster to determine a single coherence rescheduling plan for adaptive application execution on a Grid consisting of multiple clusters. A rescheduling plan divides the application into intervals, with each interval consisting of a consecutive set of phases, such that an interval is executed on a set of resources and the application is migrated between intervals to different sets of resources. For effecting the migration of the application between intervals to different sets of resources with different number

of processors during execution, the existence of a checkpointing framework, such as the work by Vadhiyar and Dongarra [12], is assumed.

In Section 2, related work is described. Section 3 describes our earlier work on performance modeling and scheduling that are used by the algorithms for generating rescheduling plans. Section 4 defines the problem of finding rescheduling plans and motivates the need for algorithms for generating plans. The algorithms for forming rescheduling plans for an application on a cluster are described in Section 5. Section 6 explains the method for forming rescheduling plans on multi-cluster Grids using the plans generated for single clusters. Section 7 describes the experiment setup and presents results showing the efficiency of the rescheduling plans. The application of the rescheduling strategies for workflow applications is described in Section 8. Section 9 presents conclusions and future work.

2 Related Work

Recently, many frameworks have been developed to reschedule executing parallel applications on Grids [3, 6, 7, 13–16]. In this section, the current work is compared with those efforts that deal with tightly-coupled parallel applications and workflow applications.

2.1 Rescheduling Tightly-Coupled Parallel Applications

Adapting loosely applications to Grid dynamics are relatively simpler since individual tasks can be migrated to better resources with minimal coordination with other tasks for adaptation [7].

There has been recent interest in developing rescheduling or reconfiguration frameworks for MPI based tightly coupled parallel applications [3, 6, 13]. The work by Fernandez et al. [13] allows application reconfiguration by over-decomposing parallel application into large number of entities (processes or threads) and migrating the threads when the availability of nodes change. These techniques are not practical for Grid systems where node availability can frequently vary. When the resource availability is small, many entities will be executed on a single node leading to application performance degradation and overloading of system resources. The work reports about 36–88% increase in execution times of some applications when many threads are mapped to a single processor.

The work by Hussein et al. [8] considers migration of components in coupled scientific models in response to resource and application dynamics based on predictions using exponential smoothing techniques. The ReSHAPE framework [3] can shrink and expand processor configurations for parallel applications on homogeneous cluster. The framework supports only homogeneous applications with uniform behavior throughout application execution and allows only shrinking to processor configurations on which the applications have previously run. The framework employs trial-and-error for rescheduling to different processor configurations with the objective of executing on a processor configuration that gives best performance for the application. Thus the method can involve rescheduling to inefficient processor configurations.

Varela et al. have developed a modular framework called Internet Operating System (IOS) [6, 9] for supporting adaption to both changing application and resource characteristics. Their decisions for rescheduling consider only process-level characteristics and are not suitable for large scale multi-phase scientific applications involving application-level change in characteristics in different phases. In the work by Bal et al. [7], an adaptation coordinator uses profile information from application processes and calculates weighted average efficiency for the application. This metric considers processor utilization by the processes and the speed of the processors. The framework adds and removes nodes whenever the efficiency values are beyond some predefined range. The framework does not consider large-scale reconfiguration or migration of executing applications and are more suitable for reconfiguring processes of divide-and-conquer applications.

All the above frameworks use profiling to obtain process-level characteristics during application execution. Online profiling can lead to large application overheads and these efforts report in-

crease in execution times of up to 20% in some cases [6, 7]. The GrADSolve infrastructure by Vadhiyar and Dongarra [4, 17] uses performance models that give expected or predicted performance for each iteration or phase of a parallel application. The framework is more suitable for iterative applications with uniform behavior throughout application execution. The existing rescheduling strategies also do not consider future change in application characteristics. In the strategies described in this paper, potential points of rescheduling are formed based on the global knowledge of all the application phases.

2.2 Scheduling and Rescheduling Workflows

Various hybrid scheduling and rescheduling strategies have been proposed for workflow applications [14, 15, 18–20]. The work by Zhang et al. [18] compares different scheduling algorithms for a large number of execution environments and workflow DAG structures for scheduling workflow applications on Grids. They consider list-based and level-based scheduling algorithms and a hybrid heuristic scheduling (HHS) algorithm that combines both list and level-based strategies. The work also proposes a new measure called effective aggregated computing power (EACP) for use as a metric in level-based heuristics for resource selection. Gong et al. [15] propose a dynamic resource-critical workflow scheduling algorithm that takes into account environmental heterogeneity including resource capacities and software configurations, and dynamism including load, queue waiting time and availability. Their algorithm reassigns the schedule based on tasks completed in the workflow and resource status at runtime.

The work by Sakellariou and Zhao [20] attempts to reschedule workflow applications at selected points during the execution to reduce the rescheduling costs. They perform selective rescheduling of selected workflow tasks based on measurable properties including estimated and actual start times, and slacks (minimum spare time of the path from the node to the exit node) of the nodes. The work by Zhang et al. [14] builds a hybrid rescheduling strategy for workflow applications. In the first step of this hybrid strategy, a subset of batch queue resources are statically selected based on aggregate computing power the batch queues. In the second step, the workflow tasks are dynamically scheduled to resources, where the mapping of a task is computed at runtime based on a list-based scheduling algorithm. When the ratio between the actual and estimated performance of the application exceeds a tolerance level, their application manager performs rescheduling by performing resource re-selection. They follow a two phase rescheduling procedure in which the old and new resource pools are combined, and the rest of the application DAG is migrated to a new set of resources. Thus rescheduling is performed if either the resource conditions change or to mitigate the effects of earlier bad rescheduling decisions.

These efforts primarily perform remapping or migration of a task/application of a workflow before its execution based on change in resource characteristics. In the current effort described in this paper, the algorithms derive a rescheduling plan for migrating an application in the middle of its execution based on both application and resource dynamics. This work work can be integrated into the workflow rescheduling strategies where in addition to changing the mapping decisions of the workflow tasks, the tasks can be migrated during execution based on the rescheduling plans derived in this paper.

2.3 Integration with Realistic Schedulers

The current work can be integrated into Grid meta-schedulers complying with the Grid standards [16, 21–24]. Emperor [24] is a OGSA metascheduler that integrates models for predicting resource characteristics, scheduling algorithm that uses these measures for deriving a schedule, and a job submitting mechanism for spawning the application on the schedule. They use AR and ARIMA methods for resource load predictions, the MDS, GRIS, GIIS, and GASS mechanisms of Globus toolkit [25] for information retrieval, resource discovery and file staging, and job execution time predictions in their scheduling algorithm. The current work can be added to the scheduling algorithm of EMPEROR, by which the algorithms described in this paper can use the re-

source performance measures in the EMPEROR framework to derive the rescheduling plan. The algorithms can also make use of Globus' Resource Specification Language (RSL) for description about application phases, thereby enabling portability. They can also be generalized to use the standard naming schemes as in WSRF [26] and XML for resource and application specifications.

Huedo et al. have developed a GridWay metascheduler that is compliant with the web service (WS) framework of the Globus toolkit [16]. They demonstrated that the performance of their metascheduler is reasonable. The GridWay metascheduler provides a virtualization layer on top of Globus services and performs job execution management and resource brokering. Dumitrescu et al. have proposed a resource brokering architecture called GRUBER [22] for providing user-level service agreement in dynamic Grid environment with different virtual organizations involving multiple administrative domains. Using the architecture, the resource providers can specify the terms of usage of the resources, thus proving controlled resource sharing. Moltó et al. [23] have developed generic WSRF-based multi-user resource brokering and metascheduling architecture for remote execution of scientific applications.

These architectures and schedulers provide for easy replacement of resource selection algorithms in the metascheduling framework. Thus, the resource selection algorithm of the current work involving rescheduling plans can be substituted for the scheduling policies in these real schedulers or frameworks, thereby leveraging various services in-built in the frameworks. These services include the Grid Resource Allocation and Management (GRAM) [27] for execution management of rescheduled applications, WebMDS for monitoring and discovery [28] for obtaining dynamic resource properties, GridFTP [29] for data access and movement and application migration, and GSI for security [30].

3 Background

3.1 Grid and Application Models

This paper considers execution of multi-phase tightly-coupled parallel applications on multi-cluster Grids consisting of dedicated or non-dedicated, interactive or batch systems. Tightly-coupled applications are defined as applications that involve frequent heavy communications among the parallel tasks. These applications are differentiated from workflow applications [31] where communications on an edge of a workflow graph are not as frequent as the inter-task communications in tightly-coupled applications and mostly happen once after the completion of an application component. Hence, while loosely coupled and workflow applications achieve good performance when executed across multiple clusters, tightly-coupled applications exhibit poor performance across multiple clusters due to low-speed network links between the clusters and are typically executed within a single cluster consisting of homogeneous machines. In the strategies for rescheduling a multi-phase tightly coupled parallel application in multi-cluster Grids, a phase of the application is executed in a single cluster while the different phases can be executed in different clusters due to rescheduling between the phases.

3.2 Performance Models for Single-Phase Applications

In our previous work [10], performance modeling strategies were developed for predicting execution times of tightly-coupled parallel applications on dedicated or non-dedicated homogeneous resources. The time taken for execution of a parallel application is calculated as:

$$T(N, P, minAvgAvailCPU, minAvgAvailBW) = \frac{f_{\text{comp}}(N)}{f_{\text{cpu}}(minAvgAvailCPU) \cdot f_{\text{Pcomp}}(P)} + \frac{f_{\text{comm}}(N)}{f_{bw}(minAvgAvailBW) \cdot f_{\text{Pcomm}}(P)} \quad (1)$$

where

- N: problem size or data size; P: number of processors;
- $minAvgAvailCPU$, $minAvgAvailBW$: represent the transient CPU and network characteristics, respectively.

- f_{comp}, f_{comm}: indicate the computational and communication complexity, respectively, of the application in terms of problem size;
- f_{cpu}: function to indicate the effect of processor loads on computations;
- f_{Pcomp}: used along with computational complexity to indicate the computational speedup or the amount of parallelism in computations;
- f_{bw}: function to indicate the effect of network loads on communications;
- f_{Pcomm}: used along with communication complexity to indicate the communication speedup or the amount of parallelism in communications.

The formula shown in (1) splits the execution time of a parallel application into two parts, f_{comp} and f_{comm}, for representing computation and communication aspects, respectively, of the parallel application. The scalability of the computational and communication times with increasing number of processors, is represented by f_{Pcomp} and f_{Pcomm}, respectively. The increase in CPU and network loads on non-dedicated systems increase the computation and communication times, respectively. $minAvgAvailCPU$ and $minAvgAvailBW$ represent the inverse of the CPU and network loads, respectively. Hence, the corresponding functions, namely, f_{cpu} and f_{bw}, are contained in the denominators. The formula shown in (1) generalizes the parallel runtime equations of many parallel numerical kernels that deal with memory-resident data [32]. These parallel kernels have single phase of uniform computations and communications and are integral to many scientific applications. The calculations of $minAvgAvailCPU$ and $minAvgAvailBW$ using available CPUs and bandwidths, obtained from Network Weather Service (NWS) [33], are explained in the previous work [10]. Available CPU is a fraction of the CPU that can be used for the application and is inversely proportional to the amount of CPU load. Available bandwidth of a link is the bandwidth on the link available to an application, and is usually lesser than the link capacity and is inversely proportional to the network load on the link. The details of the performance models, calculations of $minAvgAvailCPU$ and $minAvgAvailBW$, modeling procedure and results can also be found in the previous work [10].

3.3 Performance Models for Multi-Phase Applications

For predicting the execution times of large-scale parallel applications with multiple phases of computation and communication complexities, the major phases of execution in the applications will have to be determined. For this work, the phases are marked at compile time. We use the work by Shen et al. [34, 35] that employs a technique called *active profiling* for identifying execution phases. Active profiling uses controlled inputs and analysis of execution traces of basic blocks to identify candidate phase markers or phase boundaries, real inputs to eliminate false positives, and detailed analysis for identifying inner phase markers. For some parallel applications not amenable to active profiling, manual analysis of high-level structure of the source code is used and long-running subroutines and loop-nests are considered as candidate phases [36]. For each of the detected phases, the CPU and network load measurements and execution times within the phase boundaries can be used to derive per-phase performance models as shown in (1).

3.4 Scheduling Strategies

For scheduling tightly-coupled parallel applications on a non-dedicated or dedicated homogeneous cluster, the performance model of (1) is used for comparing various candidate schedules, and the best schedule with minimum predicted execution time for the application is chosen. In our previous work [11], a novel heuristic called Box Elimination was proposed for determining the best schedule. The algorithm works on a 3-D box of Grid points with each Grid point corresponding to a ($minAvgAvailCPU$, $minAvgAvailBW$, number of processors) tuple. The algorithm repeatedly searches through the space of hypothetical points in the box, maps the points to real schedules of machines and evaluates the schedules to determine the best schedule. The algorithm carefully avoids evaluating large number of poor

schedules by eliminating regions of hypothetical points in the box based on a hypothetical point that was searched. The details of the algorithm can be found in the previous work [11].

4 Problem Statement

Let us consider a multi-phase parallel application of a given problem size, S, with P contiguous phases of execution, $p_1 < p_2 < ... < p_P$, ordered by the times of execution. $p_i < p_j$ denotes that phase i is executed before phase j in the application. Let $f_1, f_2, \cdots, f_P$ represent the performance model functions, determined by (1), for the P phases. Let us also consider T processors with their available CPU values given by a vector of size T and the available bandwidths of the links connecting the processors given by a matrix of size $T \times T$. Let *rcost* be the worst-case overhead for rescheduling the executing application. The model functions, $f_1, f_2, \cdots f_P$, number of processors, T, the available CPUs and bandwidths and *rcost* constitute the inputs to the problem of determining a rescheduling plan.

Let us consider a disjoint partitioning of the phases into L intervals, $I_1, I_2, \cdots I_L$, with the following properties.

1. Interval I_j consists of m_j contiguous phases, $p_{\text{start}_j} < p_{\text{start}_j+1}, \cdots p_{\text{start}_j+m_j-1}$ where start_j is the index of the starting phase in I_j. $f_{\text{start}_j}, f_{\text{start}_j+1}, \cdots f_{\text{start}_j+m_j-1}$ are the corresponding performance model functions of the phases in interval I_j.
2. The intervals are ordered: $I_1 < I_2 < \cdots I_L$ where $I_l < I_m$ denotes that phases in interval I_l are executed before the phases in interval I_m.
3. The cumulative performance model function, g_j, for interval I_j is given as $g_j = \sum_{k=1}^{m_j} f_{\text{start}_j+k-1}$.
4. t_j is the execution time corresponding to the schedule determined by box elimination heuristic for interval I_j, using the performance model function g_j, problem size, S, and the resource characteristics.

The concept of phases and intervals is illustrated in Fig. 1. The disjoint set of intervals along

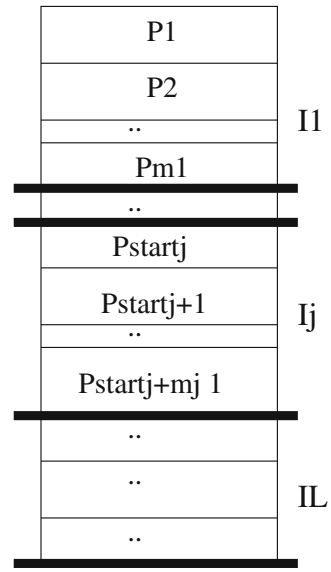


Fig. 1 Phases and intervals

with the schedules for executing the intervals represent a rescheduling plan. The application can be potentially rescheduled at the end of an interval.

The problem is to find an optimal rescheduling plan $\{I_1 < I_2 < \cdots I_{Lopt}\}$ such that

$$\sum_{i=1}^{Lopt} t_i + (Lopt - 1)rcost \tag{2}$$

is minimized subject to the condition

$$t_i > t_{\text{thres}}, \forall i \in \{1, 2, \cdots Lopt\} \tag{3}$$

In (2), the first term represents the total time spent in execution of the application and the second term represents the total rescheduling cost of the rescheduling plan. The condition given in (3) is used to avoid frequent rescheduling or rescheduling trashing by stipulating a minimum threshold of t_{thres} for execution of an interval. For this work, t_{thres} was set to 20 min since the rescheduling cost can be 5–10 min in some real reconfiguration frameworks [4]. A valid interval is defined as the interval for which the predicted execution time is less than t_{thres} and a valid rescheduling plan as the plan in which all intervals are valid.

4.1 Complexity

The total number of ways to partition a contiguous set of N items into different number of partitions is equal to $2^{(N-1)}$. This brute-force complexity can be significantly reduced by using a dynamic programming algorithm as follows. Let $T(k, p)$ be the time for optimal rescheduling plan for executing the first p phases using k reschedules and $t[a, b]$ be the estimated execution time for the interval of phases $[a, b]$, using the performance model functions in the interval. Then the problem can be formulated as:

$$T(1, p) = t[1, p] \qquad \forall p$$
$$T(k, p) = \min_{r} (T(k-1, p-r) + t[p-r+1, p] + rcost)$$
$$\forall p, \forall k > 1 \tag{4}$$

The minimum of $T[k, P]$ for all k will then correspond to the optimal rescheduling plan. However, even this dynamic programming solution leads to evaluation of 21,000 candidate plans for an application consisting of 50 phases of execution. Since evaluating such high number of candidate rescheduling plans is time-consuming and since the rescheduling plan will have to be updated periodically during runtime to adapt to resource dynamics, various heuristics for generating efficient rescheduling plans are adopted.

5 Algorithms for Generating Rescheduling Plans in a Single Cluster

All the algorithms accept as input the application problem size, *problemSize*, the total number of execution phases for the application, *phaseCount*, the performance model functions, $f_1, f_2, \cdots f_{\text{phaseCount}}$, for the phases, the rescheduling cost, *rcost*, the total number of processors, P, and the available CPU and bandwidth values of all the processors and links, respectively. The algorithms produce as output a rescheduling plan, *plan*, which includes a set of identifiers of the phases at the end of which the application can be potentially rescheduled and the schedules for executing the application between the phases. These phases are referred to as *interval markers*.

5.1 Incremental Algorithm

This algorithm incrementally tries to construct a rescheduling plan by adding intervals to the plan in increasing order. The pseudo code of the algorithm is given in Fig. 2. For forming the first interval, it considers adding phases to the interval in increasing order starting from the first phase, p_{start}. After each addition of a phase, it checks if the interval is valid using the condition in (3) (line 10). Once the algorithm finds a valid interval, with the last phase of the interval denoted by p_{end}, it uses Box Elimination algorithm to find the schedule, *prevSched*, for the interval using the cumulative function of all performance model functions for the phases in the interval, the problem size and resource characteristics (lines 7 and 14). It then tries to form a lager interval by adding the next phase, $p_{\text{end}+1}$, to the interval (line 14) and again invoking Box Elimination algorithm to determine a new schedule, *curSched* for this larger interval (line 16). It predicts the execution time of the larger interval when executed on the schedules, *prevSched* and *curSched*, as $t_{prevSched}$ and $t_{curSched}$, respectively (lines 17–20). The algorithm compares the cost of continuing on *prevSched*, $t_{\text{prevSched}}$, and the cost of rescheduling to *curSched*, $(t_{\text{curSched}} + rcost)$. Rescheduling to *curSched* involves a rescheduling cost, *rcost*. If $t_{\text{prevSched}} < (t_{\text{curSched}} + rcost)$ (line 21), then the algorithm decides that the application behavior has not changed significantly in the newly added phase. This is because the earlier schedule, *prevSched*, determined for executing the phases, $p_{\text{start}} - p_{\text{end}}$, is found to be equivalent to *curSched* for executing the phases, $p_{\text{start}} - p_{\text{end}+1}$, indicating that the application behavior does not significantly change in the phase, $p_{\text{end}+1}$, to necessitate changing the machines considered for execution. In this case, the algorithm considers adding more phases to the interval (line 22). Thus this algorithm uses the quality in schedules to detect significant changes in application behavior.

If $(t_{\text{curSched}} + rcost) < t_{\text{prevSched}}$ (line 24), the algorithm determines that the application behavior

Fig. 2 Incremental Algorithm (IA)

```
1  Algorithm:Incremental Algorithm
2  plan = ∅; start = 1; end = 1 ;
3  while end ≤ phasecount do
       /* Form the next interval                                          */
4      I = ∅ ;
5      while end ≤ phasecount do
           /* For meeting condition in Eq. 3                              */
6          g = f_start + f_start+1 + f_start+2 + ··· + f_end ;
           /* Cumulative function of performance models of phases p_start to
              p_end                                                       */
7          curSched = BE(problemSize, P, availCPU, availBW, g);
8          curP = number of processors corresponding to curSched ;
9          (curAvailCPU, curAvailBW) = average available cpu and bandwidth in
           curSched ;
10         t_curSched = g(problemSize, curP, curAvailCPU, curAvailBW);
11         if t_curSched < t_thres then end = end + 1 ;
12         else break ;
13     end
       /* Found an interval that meets the condition by Equation 3. The
          following attempts to expand the interval.                      */
14     prevSched = curSched; end = end + 1;
15     g = f_start + f_start+1 + f_start+2 + ··· + f_end ;
16     curSched = BE(problemSize, P, availCPU, availBW, g) ;
17     (curP, curAvailCPU, curAvailBW) = parameters corresponding to curSched ;
18     t_curSched = g(problemSize, curP, curAvailCPU, curtAvailBW);
19     (prevP, prevAvailCPU, prevAvailBW) = parameters corresponding to
       prevSched ;
20     t_prevSched = g(problemSize, prevP, prevAvailCPU, prevAvailBW);
21     if t_prevSched < (t_curSched + rcost) then
22         I = I ∪ p_end ; end = end + 1 ; go to line 15 ;
23     end
24     else
25         plan = plan ∪ (I, prevSched) ; start = end ;
26     end
27 end
```

has changed significantly in the phase, $p_{\text{end}+1}$, necessitating a change in the set of machines considered for execution. In this case, the algorithm forms the interval, p_{start}–p_{end}, and adds p_{end} as an interval marker to the rescheduling plan and begins the new interval from the phase, $p_{\text{end}+1}$ (line 25). This process of forming subsequent intervals by incrementally adding phases and using the change in quality of schedules to decide the interval markers continues till the algorithm adds the last phase of application execution to the rescheduling plan. This algorithm tries to find a balance between reducing the rescheduling costs by forming larger intervals and reducing the execution times of the intervals by forming new intervals when it encounters a phase with different application behavior.

5.1.1 Complexity

The total number of steps in the incremental algorithm is equal to the number of phases in the application. For each step, box-elimination scheduling algorithm [11] is invoked for determining the schedule of an interval. Thus the complexity of the incremental algorithm is equal $\mathcal{O}(phases \times complexity\_BE)$ where $complexity\_BE$ is equal to the complexity of the box-elimination algorithm, which is approximately given by:

$$complexity\_BE = \mathcal{O}\left(log_{4/3} points \times \left(proc\_cnt^2 + pr\_limit + cpu\_limit + b\,w\_limit\right)\right) \quad (5)$$

points is the total number of schedules in the search space and is dependent on the minimum and maximum values of number of processors, available CPU and bandwidth values, and the discretization of these ranges. *proc_cnt* is the total number of processors in the worst case. *pr_limit*, *cpu_limit* and *bw_limit* correspond to range of number of processors,available CPU and available bandwidth, respectively. Analysis of the complexity of the box-elimination can be performed from the description of the algorithm given in [11].

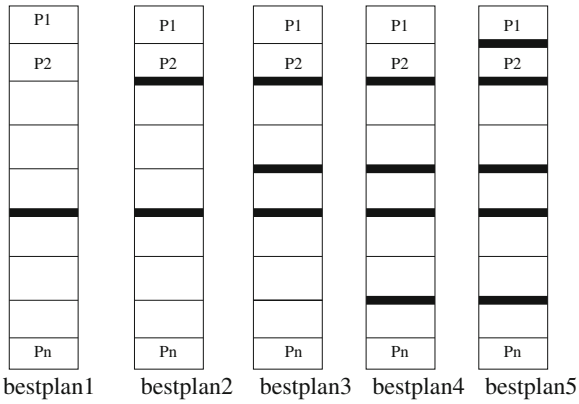


Fig. 3 Illustration of division heuristic

5.2 Division Heuristic

In this scheme, the application execution is progressively divided into increasing number of intervals. The best rescheduling plan is formed for a given number of intervals using the best plans for the lower number of intervals. The best plan for some number of intervals, *inter*, denoted as $bestplan_{\text{inter}}$, is the plan for which the total predicted application execution time is minimum among all the candidate rescheduling plans containing *inter* intervals. The total predicted execution time for a rescheduling plan is calculated by adding the sum of the predicted times for the intervals in the plan and the sum of the rescheduling costs. The predicted time for an interval is calculated as in the incremental algorithm by forming the cumulative function of all the performance model functions for the phases in the interval and using the Box Elimination algorithm with the cumulative function to find the best schedule of machines for the interval and the corresponding execution time.

$bestplan_2$ is formed for two intervals by considering all candidate rescheduling plans with two intervals. Each candidate rescheduling plan corresponds to assigning one of the phases of application execution as an interval marker. $bestplan_3$ is then formed for three intervals using $bestplan_2$ for two intervals. For forming $bestplan_{\text{inter}}$, for *inter* intervals, using $bestplan_{\text{inter}-1}$ for $inter - 1$ intervals, the interval markers in $bestplan_{\text{inter}-1}$ are used and the phases other than these interval markers are considered as candidates for another interval marker. Thus, the interval markers in $bestplan_{\text{inter}-1}$ are contained in the rescheduling plans evaluated for *inter* intervals. This is illustrated in Fig. 3.

This procedure is continued for higher number of intervals until no valid rescheduling plan can be formed for a given number of intervals. In this case, the best rescheduling plans generated for all the lower number of intervals are considered and the overall best plan is chosen. The division heuristic is suitable for applications that exhibit hierarchical behavior of application execution. These applications contain a small set of large phases of execution and each of these large phases can be further divided hierarchically into smaller phases.

5.2.1 Complexity

Since the application execution is progressively divided into increasing number of intervals, the number of phases considered for forming an interval marker in a given iteration of the division heuristic progressively decreases for different iterations. Thus, the worst case complexity of the algorithm is given by $\mathcal{O}(phases^2 \times complexity\_BE)$.

5.3 Genetic Algorithm

In genetic algorithm, a population of chromosomes is initially generated and the chromosomes undergo cross-over and mutations across various generations. The chromosomes with high fitness

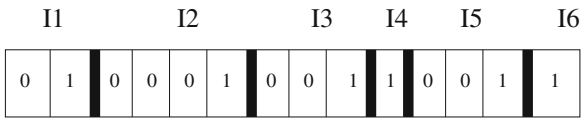


Fig. 4 Chromosome representation of rescheduling plan

values are retained over successive generations. In the problem of determining an efficient rescheduling plan, a chromosome represents a candidate rescheduling plan. The chromosome or rescheduling plan is represented as an array where the number of elements in the array is equal to the number of phases of application execution. The array consists of 0's and 1's with the 1's representing the interval markers. This is illustrated in Fig. 4.

The chromosome fitness function is calculated using (7).

$$reverseFitness = execTime\left(1+\frac{violationCount}{totalIntervals}\right) \times \left(1+\frac{avgViolationExtent}{execTime}\right) \tag{6}$$

$$fitness = \frac{1}{reverseFitness} \tag{7}$$

where $execTime$ is the predicted execution time of the rescheduling plan or chromosome calculated as in the incremental and division algorithms by using Box Elimination schedules for each interval and adding the sum of the predicted execution times of the intervals and the rescheduling costs. A *violating interval* is defined as an interval that does not satisfy the condition specified in (3). $violationCount$ is the number of violating intervals and $avgViolationExtent$ is the average of the extent of the violations in the violating intervals. The violation extent of a violating interval is the difference between the threshold limit, t_{thres} and the execution time of the interval. Thus the fitness metric attempts to minimize the rescheduling time and improve the validity of a rescheduling plan.

The mutation step tries to generate a chromosome with high fitness value from base chromosomes by splitting a large interval into two equal intervals and merging a small interval with the neighboring interval. Splitting a large interval into two small intervals will reduce the value of the first term, $execTime$, of (6) since the sum of the execution times of the two small intervals will be less than the execution time of the large interval. This is because efficient schedules can be formed for small intervals with small number of phases while schedule of machines for a large interval cannot efficiently meet the requirements of the characteristics for the large number of phases in the interval. Merging of two intervals helps to minimize the last two terms of (6) and hence improve the validity of the plan.

After performing cross-over and mutations of chromosomes for a given generation, *selection* of chromosomes for the next generation is performed using *Roulette-wheel* based selection method and *elitism*. By using *Roulette-wheel* based probabilistic selection policy, chromosomes with high fitness values have high probabilities of being retained for the next generation. The algorithm also uses *elitism* in which the chromosomes with fitness values that are within 10% of the highest fitness value in the generation are retained for the next generation. The steps of cross-over, mutations and selection are repeated for a certain number of generations. After performing experiments with the genetic algorithm for different input configurations with different total number of generations, a total of 50 generations was chosen for the algorithm. Executing for 50 generations resulted in convergence to solution or best schedule within reasonable time for most of the experiments. After executing the genetic algorithm for 50 generations, the best valid rescheduling plan with the highest fitness value is chosen.

5.3.1 Complexity

For each generation, a given population of chromosomes representing rescheduling plans are evaluated for fitness. For calculating fitness of a chromosome, the predicted execution cost of the plan is determined for the schedules of the intervals in the chromosome formed using the Box-Elimination algorithm. Thus, the complexity of the genetic algorithm is given by $\mathcal{O}(generations \times populationSize \times complexity\_BE)$.

6 Rescheduling Plans for Multi-Cluster Grids

The algorithms in the previous section generate single-cluster rescheduling plan (SCRP) for a cluster based on the performance models obtained for the cluster. Invoking an algorithm on different clusters with the corresponding performance models on the clusters will lead to formation of different SCRPs for the different clusters. This section discusses the problem formulation and an algorithm for the formation of a coherent rescheduling plan involving multiple clusters of a Grid using the SCRPs generated for the individual clusters by the algorithms described in the previous section.

6.1 Problem Formulation

1. Let us consider a multi-phase parallel application with P contiguous phases of execution. Let C be the total number of available clusters.
2. $f_{i,j}$ $(1 \leq i \leq P, 1 \leq j \leq C)$ denotes the the performance model function of the ith phase on the jth cluster.
3. *intraCost* is the rescheduling cost associated with rescheduling an application to a different set of machines within the same cluster, and *interCost* is the rescheduling cost for rescheduling to a different cluster.
4. $\{SI_{1,j} < SI_{2,j} < \cdots SI_{L_j,j}\}$ are the intervals in the single cluster rescheduling plan (SCRP) of a cluster j $(1 \leq j \leq C)$, formed using one of the three algorithms discussed in the previous section. L_j represents the number of intervals formed in cluster j. Let $Sg_{k,j}$ be the cumulative performance model function for interval $SI_{k,j}$ in cluster j and $St_{k,j}$ be the execution time corresponding to the schedule determined by box elimination heuristic for interval $SI_{k,j}$, using the performance model function $Sg_{k,j}$.

The problem is to find an optimal multi-cluster rescheduling plan (MCRP) containing the intervals $\{MI_1 < MI_2 < \cdots MI_{Lopt}\}$, with the intervals in the plan scheduled on resources $\{MS_1, MS_2, \cdots MS_{Lopt}\}$ in the clusters $\{MC_1, MC_2, \cdots MC_{Lopt}\}$ for execution such that

$$\sum_{i=1}^{Lopt} t_i + (Lopt - 1)rcost \tag{8}$$

is minimized subject to the condition

$$t_i > t_{\text{thres}}, \forall i \in \{1, 2, \cdots Lopt\} \tag{9}$$

The definitions of t_i and *rcost* for a multi-cluster rescheduling plan (MCRP) are slightly different from the corresponding definitions for a single-cluster rescheduling plan (SCRP) given in Section 4 and (2). For a MCRP, t_i is the predicted execution time for schedule MS_i found using the cumulative performance model function, Sg_{i,MC_i}, of the interval MI_i in cluster MC_i. The rescheduling cost, *rcost*, is defined as:

$$rcost = \begin{cases} 0 & \text{if } MS_i = MS_{i-1} \\ intraCost & \text{if } MS_i \neq MS_{i-1} \text{ and } MC_i = MC_{i-1} \\ interCost & \text{if } MS_i \neq MS_{i-1} \text{ and } MC_i \neq MC_{i-1} \end{cases} \tag{10}$$

For forming the intervals in MCRP, one of the three algorithms discussed in the previous section for forming the SCRPs on the individual clusters is repeatedly invoked, and the intervals in the SCRPs are used as follows. For forming a particular interval, MI_i in MCRP, the SCRPs are formed for the phases from the beginning of the interval MI_i to the final phase of of application execution. The smallest of the first intervals in the SCRPs in terms of execution time is then chosen as the interval MI_i. The next interval MI_{i+1} is formed by once again forming SCRPs for the phases from the beginning of the interval MI_{i+1}.

6.2 Algorithm

The algorithm for multi-cluster rescheduling plan (MCRP) takes as input the application problem size, the total number of execution phases for the application, P, the number of clusters, C, a $C \times P$ matrix of performance model functions, f, where $f_{i,j}$ denotes the performance model obtained on the jth cluster for the ith phase, a vector of total

number of processors in each cluster, and the available CPU and available BW values of the processors and links of all the clusters, the cost for rescheduling in the same cluster, *intraClusterCost* and and the cost for rescheduling to a different cluster, *interClusterCost*. The algorithm is shown in Fig. 5.

The MCRP generation algorithm begins by invoking one of the SCRP generation algorithms, *scrpAlgo*, described in the previous section, for

Fig. 5 Algorithm for generating multi cluster rescheduling plan

```
1  Algorithm:MCRPAlgo
2  MCRP = ∅ ; currentInterval = 0;
3  while number of phases in MCRP ≠ phaseCount do
4      for j = 1 to clusterCount do
5          SCRP_j =
           scrpAlgo(problemSize, phaseCount, f_j, P_j, availCPU_j, availBW_j) ;
6      end
7      Sort first intervals of SCRPs to form a vector of sortedFirstIntervals ;
8      for j = 1 to clusterCount do
9          g_j = cumulative function of performance models in sortedFirstIntervals_j ;
10         sched_j = BE(problemSize, P_j, availableCPU_j, availableBW, g_j) ;
11         (currentP, currentAvailCPU, currentAvailBW) = sched_j parameters;
12         t_j = g_j(problemSize, currentP, currentAvailCPU, currentAvailBW) ;
13         if t_j > t_thres then
14             multiClusterInterval_currentInterval = sortedFirstIntervals_j ;
15             break;
16         end
17     end
18     for j = 1 to clusterCount do
19         g_i = cumulative function of models in multiClusterInterval_currentInterval ;
20         newSched_j = BE(problemSize, P_j, availableCPU_j, availableBW, g_j) ;
21         (currentP, currentAvailCPU, currentAvailBW) = newSched_j parameters;
22         newT_j = g_j(problemSize, currentP, currentAvailCPU, currentAvailBW) ;
23     end
24     bestTime = minimum of newT; bestSchedule = corresponding newSched ;
25     if MCRP = ∅ then
26         MCRP = MCRP ∪ (multiClusterInterval_currentInterval, bestSched) ;
27     end
28     else
29         sameShedule = schedule for multiClusterInterval_currentInterval−1 ;
30         timeInPrevSchedule = predicted execution time for
           multiClusterInterval_currentInterval on sameSchedle ;
31         scheduleonSameCluster = scheduled obtained by invoking BE on the
           cluster containing sameSchedule ;
32         timeinPrevCluster = predicted execution time for
           multiClusterInterval_currentInterval on scheduleonSameCluster
33         Compare (timeInPrevSchedule),
           (timeinPrevCluster + intraClusterCost), (bestTime + interClusterCost) ;
34         if timeInPrevSchedule is minimum then
35             MCRP =
               MCRP ∪ (multiClusterInterval_currentInterval, sameSchedule) ;
36         end
37         else if (timeinPrevCluster + intraClusterCost) is minimum then
38             MCRP = MCRP ∪
               (multiClusterInterval_currentInterval, scheduleonSameCluster) ;
39         end
40         else if (bestTime + interClusterCost) is minimum then
41             MCRP = MCRP ∪ (multiClusterInterval_currentInterval, bestSched) ;
42         end
43     end
44     currrentInterval = currentInterval + 1 ;
45 end
```

generating rescheduling plan for each cluster (line 5). It then decides a schedule for executing the first interval by first sorting the first intervals in the rescheduling plans of all the clusters in terms of the number of phases in the intervals (line 7). It chooses the smallest first interval, $multiClusterInterval_1$, that is valid in all the clusters (line 14). The algorithm finds schedules in each cluster for executing $multiClusterInterval_1$ using Box Elimination (lines 18–23) and chooses the cluster and schedule of machines in the cluster with the minimum predicted execution time for $multiClusterInterval_1$ execution (line 24). The algorithm adds the $multiClusterInterval_1$ and the corresponding schedule to MCRP (line 26).

For determining the jth interval of MCRP and schedule for the interval, the algorithm regenerates SCRPs for the remaining phases not contained in MCRP and forms $multiClusterInterval_j$ similar to the formation of $multiClusterInterval_1$ for the first interval. The algorithm then determines three predicted execution times:

1. $timeInPrevSchedule$ which is the predicted execution time for $multiClusterInterval_j$ when executed on the schedule determined for the previous interval, $multiClusterInterval_{j-1}$ (lines 29 and 30). This schedule is denoted as $sameSchedule$. In this case, the application will be continued to execute on the same schedule and will not be rescheduled.
2. $timeinPrevCluster$ which is the predicted execution time for $multiClusterInterval_j$ when executed on the schedule obtained by invoking the Box Elimination algorithm for $multiClusterInterval_j$ for the cluster containing the schedule determined for the previous interval (lines 31 and 32). This schedule is denoted as $scheduleonSameCluster$. In this case, the cost for rescheduling to a different schedule on the same cluster, $intraClusterCost$, is added to $timeinPrevCluster$.
3. $bestTime$ which is the minimum of predicted execution times for $multiClusterInterval_j$ obtained on all the clusters by invoking Box Elimination algorithm for the interval in each of the clusters (line 24). This schedule is denoted as $bestSchedule$. In this case, the cost for rescheduling to a different schedule on a different cluster, $interClusterCost$, is added to $timeinPrevCluster$.

The algorithm then performs a three-way comparison between $timeInPrevSchedule$, ($timeinPrevCluster + intraClusterCost$) and ($bestTime + interClusterCost$) to decide if $multiClusterInterval_j$ has to be executed on $sameSchedule$, $scheduleonSameCluster$ or $bestSchedule$ (lines 33–41), respectively. The algorithm continues performing the above steps until it includes all the phases in MCRP. The algorithm for generating MCRP is illustrated in Fig. 5.

6.2.1 Complexity

The complexity of the MCRP algorithm depends on the SCRP algorithm used for forming rescheduling plans for each cluster. The SCRP algorithm is repeatedly invoked after choosing an interval from the intervals formed in all clusters. In the worst case, the number of invocations of the SCRP algorithm can be equal to the number of application phases. Thus the complexity is given by $\mathcal{O}(phases \times complexity\_SCRP)$ where $complexity\_SCRP$ is the complexity of the SCRP algorithm used. The SCRP algorithm can be one of incremental, division and genetic algorithms. The number of clusters do not impact the complexity of the MCRP algorithm since the SCRP algorithm for each cluster can be invoked independently.

7 Experiments and Results

A large number of simulations were performed using a custom-built simulator to evaluate the rescheduling strategies. The details of the simulator are given in the following subsection. The results corresponding to the simulations shown in this section include both the predicted execution times and rescheduling costs for the applications. Five large scale multi-phase parallel applications were used for evaluating the rescheduling strategies.

1. Molecular dynamics simulation (MD) of Lennard-Jones system systolic algorithm.

2. ChaNGa (Charm N-body GrAvity solver) [37], an application for performing collisionless N-body simulations.
3. Athena [38], a grid-based code for astrophysical gas dynamics with nested and adaptive mesh capabilities.
4. LAMMPS [39] (Large-scale Atomic/Molecular Massively Parallel Simulator), a classical molecular dynamics simulation code for studying crack propagation in 2-D solid materials.
5. MIT Photonic-bands (MPB) [40], a program for computing the band structures (dispersion relations) and electromagnetic modes of periodic dielectric structures.

By using active profiling [34, 35] and manual analysis of source codes, the total number of application execution phases was determined as 30 for MD, 50 for ChaNGa, 100 for Athena, 26 for LAMPPS and 32 for MPB. The performance model equations for the application phases were obtained on a 48-core AMD Opteron cluster consisting of 12 2-way dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers connected by Gigabit Ethernet. In the experiments, the available CPU values ranged from 0.1–1.0 where a value of 1.0 indicates an unloaded processor. Thus the available CPU of 0.75 for a processor in the simulation setup represents an AMD processor that is one-fourth loaded.

7.1 Details of Simulator

A custom-built simulator was developed for simulating application execution on a single cluster or multi-cluster Grid using a rescheduling plan. The simulator considers different inputs given as input configuration files. The *application configuration file* specifies various application configuration parameters, namely the problem size, the number of computational and communication phases of the application, and the performance model functions of each phase. A performance model for a phase is a function of the application problem size, the number of processors for execution, the CPU speed, the maximum network bandwidth, the CPU load expressed as available CPU which a number between 0–1 (0 for fully loaded and 1 for free CPU), and the network load expressed as available bandwidth which is a percentage of the maximum network bandwidth. The performance model function gives the estimated execution time of the phase when executed on the given number of processors with given CPU and network loads. The performance model functions can be obtained using the performance modeling strategies described in our previous work [10].

The *resource configuration file* input to the simulator specifies the number of clusters for multi-Grid experiments, and for each cluster specifies the total number of processors, the CPU speed of the processors, the maximum intra-cluster network bandwidth of the links connecting the processors in the cluster, the available CPU (inverse of CPU load) for each processor, and the available bandwidth (inverse of network load) for each link in the cluster. For multi-cluster Grids, the file also specifies the maximum inter-cluster bandwidth of the links between the clusters, and the network loads on the links.

The *rescheduling configuration file* input to the simulator specifies the intra and inter cluster costs of rescheduling or rescheduling overheads, and also specifies a period of updating the CPU and network loads in the resource configuration file to simulate the resource load dynamics in the multi-cluster Grid experiments. This configuration file also specifies one of the three algorithms described in Section 5 to generate the single cluster rescheduling plan (SCRP).

The simulator is implemented as two processes, a *main process* and a *resource update* process. The resource update process updates the CPU and network loads in the resource configuration file with the period specified in the rescheduling configuration file. The main process uses the parameters in the different configuration input files, generates a single-cluster rescheduling plan (SCRP) for each cluster specified in the resource configuration file and a multi-cluster rescheduling plan (MCRP) if more than one cluster is specified in the resource configuration file. For generating SCRP, the simulator uses the resource and application parameters specified in the resource and application configuration file, respectively, passes these parameters to the performance model functions for the different application phases in the application configuration files, and obtains the

predicted execution times of the phases. The simulator passes these predicted execution times along with the rescheduling costs specified in the rescheduling configuration file to the algorithm specified in the configuration file to form SCRP for each cluster. In multi-cluster Grids, the simulator uses these SCRPs, the inter-cluster bandwidths specified in the resource configuration file, and the rescheduling costs, and invokes the MCRP algorithm described in Section 6 to form a multi-cluster rescheduling plan.

7.2 Results

In this section, the accuracy of using multiple performance models for the different phases of a single application in predicting the execution time of the application is first shown. The efficiency of the rescheduling plans generated by the rescheduling strategies is then evaluated by comparing the plans with the plans generated by a brute force method. Finally, the multi cluster rescheduling plan generation algorithm is evaluated using simulations of dynamic multi-cluster Grids.

7.2.1 Prediction Accuracy due to Cumulative Performance Models

The efficiency of the rescheduling plans formed by the algorithms for a multi-phase application primarily depends on the accuracy of the performance models of the individual phases of the application. In this section, the accuracy of the performance models of the phases in predicting the execution time of the application is evaluated. A *cumulative performance model* is formed for the complete application using the performance models for the individual phases of the application and the cumulative model is used to predict application execution time. Some existing rescheduling efforts [4] determine points of rescheduling in an application by using a *single performance model* formed for the complete application by observing only the total execution time of the complete application and not the execution times of the individual phases. The accuracy of such *single performance models* in predicting the performance of multi-phase applications is also evaluated. Thus, the usefulness of such models in forming efficient rescheduling plans for the multi-phase applications is evaluated.

Different applications were executed with different problem sizes and number of processors on the non-dedicated AMD cluster, and the execution times of the individual phases and the entire applications were observed for 150 experiments. For each application, a *single performance model* was developed for the complete application executions using the total executed times. Performance models were also developed for the individual phase executions using the phase execution times. A *cumulative performance model* was then formed for the complete application using the performance models for the individual phases of the application. The *single* and *cumulative performance models* were used for predicting execution times for different application and processor configurations. Table 1 compares the percentage prediction errors (PPE) when using single and cumulative performance models for predicting execution times of the applications.

It can be found that using single performance model function gives highly inaccurate predictions or greater than 35% average PPEs for ChaNGa, LAMMPS and MD applications. These applications exhibit highly non-uniform behavior dur-

Table 1 Percentage predictions errors (PPEs) with single and cumulative models for 150 experiments

Application	PPE (single model)		PPE (cumulative model)	
	Avg.	Std. dev.	Avg.	Std. dev.
MD	46.36	35.05	24.62	16.41
Athena	22.18	11.97	18.26	8.84
ChaNGA	86.87	73.37	25.2	20.41
MPB	28.47	16.19	18.37	12.30
LAMMPS	35.34	27.99	21.7	13.21

Cumulative models have lower PPEs or higher prediction accuracy than single models

ing executions and the computation and communication characteristics widely vary between the phases. In comparison, Athena and MPB have fairly uniform computation and communication characteristics throughout application executions. Hence using single performance models for Athena and MPB gave less than 30% average PPEs. In all cases, using cumulative performance models formed from the models of individual phases gave less prediction errors than using single performance models.

Figure 6 shows the percentage prediction errors with single and cumulative models for ChaNGa application for different application and processor configurations for 150 experiments. The x-axis represents the different configurations. The figure illustrates that the PPEs of the cumulative models are on an average 70% lower than the PPEs of the single models in all the experiments. Thus, the performance models of the individual phases of the different applications give reasonably small prediction errors and can be used for formation of efficient rescheduling plans by the algorithms. It is also shown that single performance models for complete applications, formed using the total execution times of the applications, have high prediction inaccuracies for multi-phase applications and cannot be used for formation of efficient rescheduling plans.

7.2.2 Evaluation of Single-Cluster Rescheduling Plans

In this section, the efficiency of the rescheduling plans generated by the algorithms is evaluated by comparing the execution costs of the plans with the costs of the near-optimal rescheduling plans generated by a brute force method. The brute force method determines a near-optimal rescheduling plan for an application with N phases by evaluating all the candidate rescheduling plans and choosing the best rescheduling plan with the minimum execution cost. For determining the execution cost of a rescheduling plan, execution costs of the individual phases are added with the rescheduling costs. By means of simulations with different resource configurations and different multi-phase applications on a single cluster, single-cluster rescheduling plans (SCRPs) are generated using the algorithms and the brute force method and the execution costs of the plans are compared.

The brute force method typically takes about 6-8 hours for execution for an application with 30 phases since it has to evaluate all the $2^{(30-1)}$ rescheduling plans possible with 30 phases. The SCRP generation algorithms in this work take only few minutes for execution. This is because the incremental and division algorithms, after

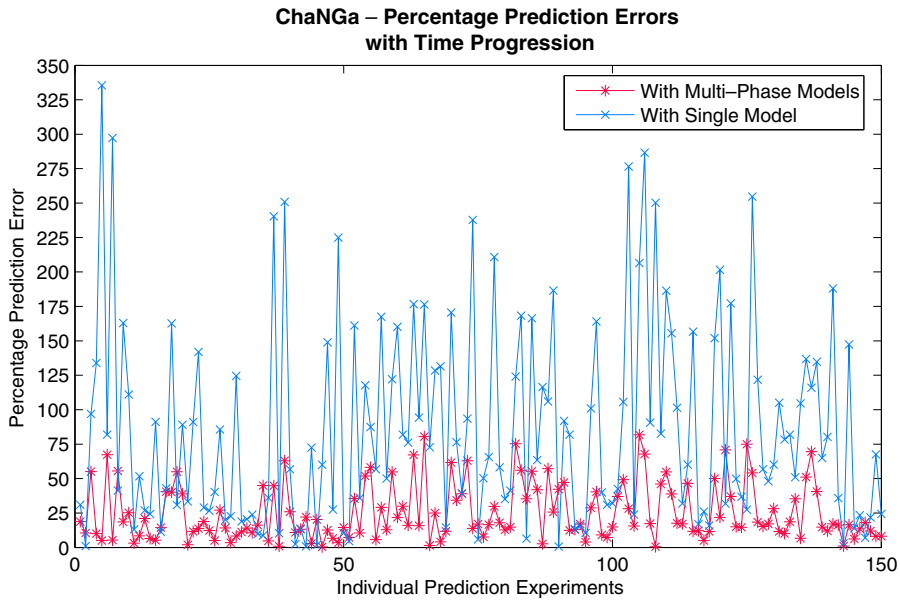


Fig. 6 Percentage prediction errors with single and cumulative models for ChaNGa for 150 experiments. Cumulative models have about 70% lower PPEs than single models

forming an interval with a set of phases, consider only the remaining phases for formation of subsequent intervals in the rescheduling plan. Thus, in the worst case, the total number of candidate rescheduling plans evaluated by these algorithms for an application with 30 phases is approximately $30(30-1)/2$. The genetic algorithm, in the worst case, evaluates approximately ($generations \times populationSize$) candidate rescheduling plans. In the genetic algorithm implementation, a total of 50 generations and 200 chromosomes were considered, leading to a total of 10,000 evaluations. However, in practice, since many chromosomes are retained across generations, the total number of candidate rescheduling plans that are considered is significantly less. Thus, the algorithms take much less time to execute than the brute force method.

Application executions were simulated for different applications on a cluster of 512 processors. For each application, 50 simulation experiments were conducted with different resource load configurations. For each simulation experiment, SCRPs were generated for a resource load configuration using each of the three algorithms and the brute force method. For each SCRP for a simulation experiment, the predicted execution cost was observed.

For obtaining a resource load configuration for a simulation experiment, random values were chosen for maximum intra-cluster bandwidth of links in a cluster, available bandwidth of each link and the available CPU value of each processor. The maximum intra-cluster bandwidth was chosen as one of 100 Mbps, 1 Gbps, 5 Gbps and 10 Gbps. The intra-cluster bandwidths on the links connecting processors of a cluster impacts the cost of rescheduling an application. Higher bandwidths lead to smaller rescheduling costs due to smaller times incurred in transferring checkpoints of the application across the network for continuation of the application on different number of processors in a cluster. This leads to reduction in overall execution time of the application for a given rescheduling plan, as given by the relationship between the rescheduling cost and execution time in (2). However, as can be seen in the equation, while the execution times of different candidate rescheduling plans change with change in intra-cluster bandwidths, the relative rankings of the candidate plans will remain the same since a rescheduling plan is derived for a single rescheduling cost. Hence the optimal single-cluster rescheduling plan (SCRP) does not change with the change in intra-cluster bandwidths.

The available bandwidth of each link was randomly varied to be within 20–80% of the maximum available bandwidth. The available CPU value was also randomly varied between 0.1–1.0 for each processor. Although existing work use more realistic CPU and network models [24, 41, 42] than the random variations in CPU and network loads used in the experiments, such random large variations are used to simulate systems with extreme load dynamics. The large variations in the random CPU and network loads help to represent systems with high load dynamics and hence help in better demonstration of the robustness of the rescheduling algorithms and the simulator than the realistic models. In our earlier work, it was shown that the random network loads used in the experiments have similar dynamics as the realistic network loads and hence are realistic. In the realistic CPU and network bandwidth environments, as represented by the existing models [24, 41, 42], it is expected that the rescheduling plans will involve smaller number of rescheduling events, and hence incur smaller rescheduling overheads.

Figure 7 shows the average execution times of the rescheduling plans generated by the algorithms and the brute force method for the ChaNGA application. For this comparison, we considered only 15 phases of the ChaNGA application execution obtained by considering the execution with the first 45 phases and merging every three consecutive phases into one single phase. Table 2 shows the average percentage differences between the execution times of the plans generated by the algorithms and the brute force method for different rescheduling overheads for the ChaNGA application. The maximum rescheduling cost of 5 minutes considered in the evaluations corresponds to the overheads in real checkpointing systems [12].

Figure 7 and Table 2 show that the rescheduling plans generated by the algorithms are highly efficient and the execution times of the plans generated by the algorithms are competitive when

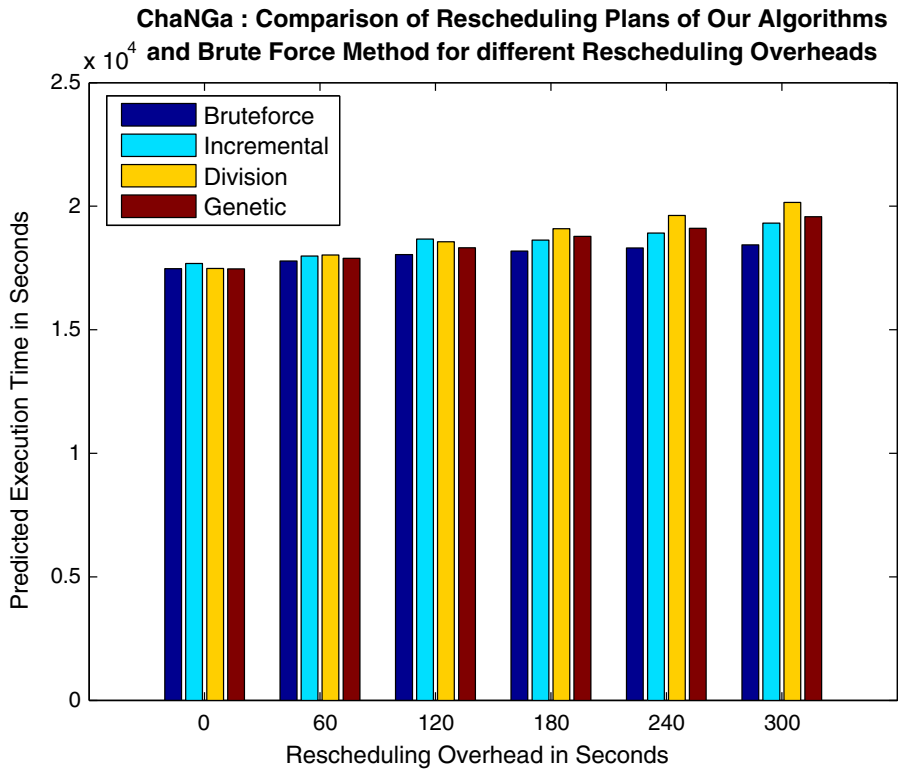


Fig. 7 Comparison of the algorithms with brute force method (ChaNGA). The *y*-axis shows the average execution times of the rescheduling plans for 50 experiments

compared to the execution times of the plans generated by the brute force method. The maximum percentage difference between the execution times of the plans of the brute force method and the algorithms is less than 10%. Thus, the algorithms result in efficient rescheduling plans whose execution times are comparable to the execution times of the near-optimal rescheduling plans generated by the brute force method.

Table 2 Comparison of the algorithms with brute force method (ChaNGA)

Rescheduling overhead (s)	Incremental	Division	Genetic
0	1.23	0.06	0.01
60	1.15	1.38	0.63
120	3.54	2.86	1.53
180	2.42	4.96	3.24
240	3.28	7.18	4.33
300	4.78	9.32	6.17

The numbers show the average percentage differences in execution times of the rescheduling plans of the brute force method and the algorithms for 50 experiments and for different rescheduling overheads

It is also found that the average execution times of the rescheduling plans generated by the genetic algorithm are lesser than the times corresponding to incremental and division algorithms in most cases. This is because the genetic algorithm explores widely varying candidate rescheduling plans due to initial random population generation, and cross-overs and mutations for each generation. However, incremental and division algorithms evolve the intervals in a rescheduling plan based on the intervals that are already formed in the plan. Thus, the candidate rescheduling plans considered in the incremental and division algorithms are not as widely varying as in the genetic algorithm. Hence genetic algorithm attempts to obtain globally efficient rescheduling plans resulting in smaller execution times.

7.2.3 Rescheduling Plans for Multi-Cluster Grids

In this section, the efficiency of Multi-Cluster Rescheduling Plans (MCRPs) generated by the algorithm, described in Section 6, is evaluated for application executions on multi-cluster Grid

environments with load dynamics. The MCRPs are formed using the Single-Cluster Rescheduler Plans (SCRPs) generated for each cluster by the three algorithms. A number of simulation experiments was conducted with different multi-cluster Grids and applications. For each simulation experiment with an application, a multi-cluster Grid is randomly generated with a specific number of clusters and processors in a cluster, maximum inter and intra-cluster bandwidths, and network and processor loads. One of the three algorithms, namely, incremental, division and genetic algorithms, was used to generate an SCRP for each cluster in the Grid. The SCRPs of the clusters were used to generate an MCRP for multi-cluster application execution using the algorithm described in Section 6. In a simulation experiment, the change in load characteristics of the clusters during application execution was also simulated, resulting in change of MCRP during execution. Thus application executions on typical Grid systems involving resource load dynamics were simulated. For the simulation experiment, the total execution time of the MCRP was calculated using the execution times of the individual application phases on different clusters and intra and inter rescheduling costs. The execution costs of the MCRPs were compared with the costs of executing an application on a single schedule in a cluster. The single schedule was determined using the scheduling algorithm based on the resource characteristics that existed at the beginning of the application execution. By this comparison, we attempt to show the benefits of rescheduling in general, and using the algorithms for rescheduling in particular, over using a single schedule for executions involving application and resource dynamics.

For the simulation experiments, 250 multi-cluster Grid setups were randomly generated, the performance model functions of the different phases of MD application were used, and the MCRPs developed using SCRPs generated by different algorithms were compared for each setup. $U[x, y]$ is used to denote uniform probability distribution in the interval (x, y). For randomly generating each multi-cluster setup Grid setup, U[5, 12] number of clusters and U[32, 512] number of processors in each cluster were generated. In order to simulate heterogeneity of processors in different clusters, a random *cpu scaling factor* is used from the set $(0.6, 0.8, 1.0, 1.2, 1.4)$[1] for each cluster, and multiplied with the coefficients of computational complexity of the performance model equations.

The maximum intra-cluster bandwidth of links in a cluster was chosen to be one of 100 Mbps, 1 Gbps, 5 Gbps and 10 Gbps. The maximum inter-cluster bandwidth of links connecting two clusters was chosen to be one of 0.6, 0.8, 1, 5, and 10 Mbps. These bandwidths are commonly observed on the links connecting two clusters located at two different sites in many Grid systems. While the change in intra-cluster bandwidths impacts only the rescheduling cost and not the solution for a SCRP as discussed in Section 7.2.2, the change in inter-cluster bandwidths and the relative difference between the intra and inter-cluster bandwidths can change the solution for a multi-cluster rescheduling plan (MCRP). This is because the algorithm for MCRP, discussed in Section 6, chooses an interval in a plan and the schedule for the interval based on the comparison between the intra and inter rescheduling costs, associated respectively with executing the interval in the same cluster as the previous interval and executing in a different cluster. The comparisons using the intra and inter rescheduling costs are given in (10). These intra and inter rescheduling costs in turn depend on intra and intra bandwidths, respectively, since rescheduling involves transfer of application checkpoints on the links for continuation of application on a different set of machines. Thus, slower inter-cluster links can cause the algorithm to form a MCRP containing more intra-cluster rescheduling than inter-cluster rescheduling.

The available bandwidth of each link was randomly varied to be within 20–80% of the maximum available bandwidth. The available CPU value was also randomly varied between 0.1–1.0 for each processor. To simulate load dynamics in the Grid during application execution, the available CPU and bandwidth values of the processors

[1] The scaling factors are chosen from a small range since the CPU speeds of modern processors vary by small amounts.

and links, respectively, are varied after addition of every interval to the multi-cluster rescheduling plan (MCRP) in the MCRP generation algorithm. The MCRP for the subsequent intervals is formed based on the updated processor and network loads. Hence the rescheduling plans involving the different intervals are dynamically updated based on the resource dynamics.

Table 3 summarizes the execution times of the MCRPs generated using SCRPs formed by different algorithms. The table also shows the execution times when the application is executed on a single schedule, determined at the beginning of the application execution, for the entire duration of the application. The large standard deviations, shown in the table, are due to the differences between the simulation setups used for the experiments in terms of the number of clusters, number of processors in a cluster, heterogeneity of the processors, inter and intra cluster bandwidths and load dynamics.

It is found that rescheduling in response to application and resource dynamics, using the rescheduling plans generated by the algorithms, gives large reductions in execution times when compared to execution of the applications on a single schedule throughout application execution. Application execution on a single schedule is not able to adapt to high resource and application dynamics leading to huge execution times. It is also found that MCRPs based on genetic algorithm give 9–12% smaller average execution times than MCRPs based on incremental and division heuristics. This is because genetic algorithm explores different rescheduling plans at all stages using cross-over and mutation and attempts to achieve globally efficient rescheduling plans while in the incremental algorithm, the intervals are added to the rescheduling plans without the knowledge about the formation of the subsequent intervals. The assumption in the division heuristic about the hierarchical execution behavior of the applications is not applicable to molecular dynamics application. This results in large standard deviation values in the division heuristic although the average execution times in the division heuristic is lesser than in the incremental algorithm. Thus, using the multi-cluster Grid simulations, it is seen that rescheduling plans by the algorithms give much smaller execution costs than single schedule executions and the genetic algorithm that explores diverse rescheduling plans gives best average performance.

Table 3 Comparison of algorithms on multi cluster Grid setups (MD)

Rescheduling method	Avg. (h)	Std. dev. (h)
Incremental	6.8	4.72
Division	6.58	5.30
GA	5.97	4.05
Single schedule	68.77	75.38

Average and standard deviations of execution times of rescheduling plans by the algorithms and execution on single schedule for 250 experiments

7.2.4 Impact of Cluster Heterogeneity on Efficiency of Rescheduling Plans

In this section, the effect of heterogeneity of different clusters in a multi-cluster Grid setup on the efficiency of the algorithms is determined. The *weight* of a cluster was calculated as a product of number of machines in the cluster and the cpu scaling factor for the cluster. Clusters with larger weights are expected to yield better schedules. The ratio of the maximum and minimum weight was calculated for a multi-cluster setup. Higher values of this ratio indicate greater heterogeneity between the clusters. The 250 multi-cluster setups was divided into different groups with different ranges of ratios. The groups that had less than 5 setups were eliminated. The execution costs of the rescheduling plans by the algorithms for various groups of cluster heterogeneities was then analyzed.

Figure 8 shows the average performance of the algorithms for different groups of ratio values. It is found that with increasing ratio of maximum to minimum cluster weight or increasing heterogeneity, the application execution times decrease. This is because with increasing differences in capacities of the clusters, the schedules for most of the phases of application execution will be chosen from the cluster with largest weight or highest capacity. Hence, with increasing heterogeneity, applications incur more intra-cluster rescheduling costs and less inter-cluster reschedul-

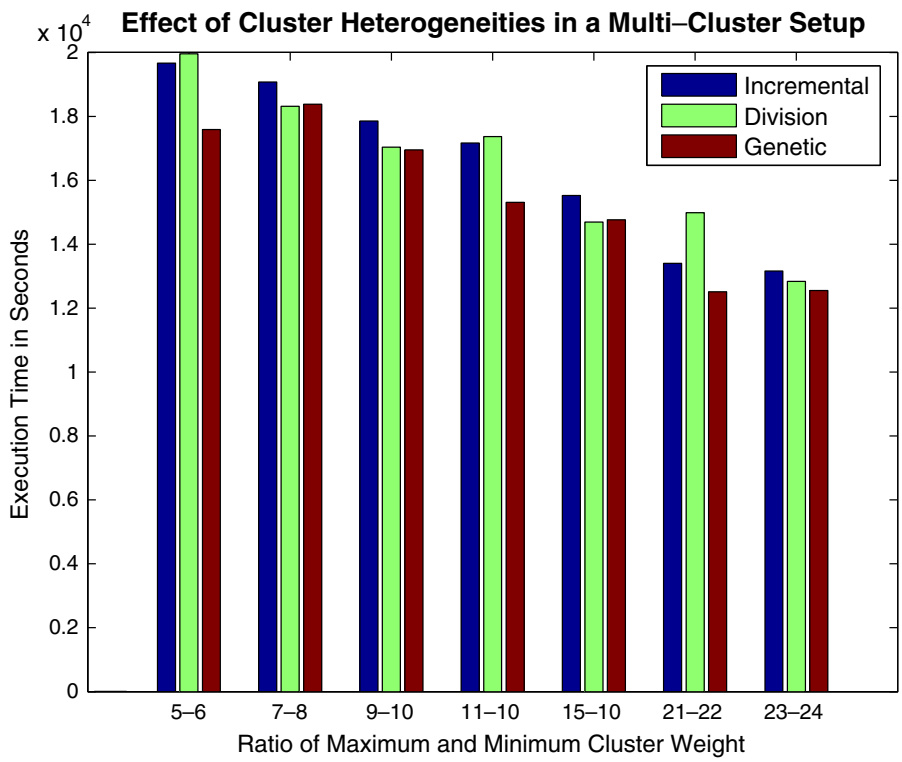


Fig. 8 Effect of cluster heterogeneity on execution times multi-cluster Grid setup

ing costs. Since the intra-cluster rescheduling costs are smaller than the inter-cluster rescheduling costs, the execution times of the applications decrease with increasing heterogeneity. It is also found that genetic algorithm gives better average execution times than incremental and division heuristics for all cases.

8 Discussion: Application to Workflows

The strategies in this paper for rescheduling using rescheduling plans can also be applied to workflow applications consists of both serial and parallel jobs [43–48]. A workflow application is represented as a Directed Acyclic Graph (DAG) where each node of the graph represents a parallel or sequential job/application that forms a component of the overall application and an edge denotes the control and data dependency between two application components. Most of the practical workflow applications have a well defined *start* node/application that initializes and sets up the data and execution environment for a workflow. Such a workflow application can be scheduled and rescheduled on a Grid consisting of multiple clusters using the algorithm for generating multi-cluster rescheduling plan (MCRP) described in Section 6. A meta heuristic can be devised that first forms the BFS (breadth-first-search) tree from the workflow graph with the *start* node as the root of the tree. The meta heuristic will traverse the tree top-down, trying to minimize the execution time of each level of a tree. For a given level, the meta heuristic will invoke in parallel the algorithm for forming MCRP, described in Section 6, for each node/application in the level. Since the MCRP algorithm is invoked in parallel and independently for the applications in the level, this can result in "scheduling conflicts" where a single resource can be contained in the schedules, formed by the MCRP algorithm, for multiple applications in the level of the tree. To resolve these conflicts, the MCRP algorithm can be extended by which, after the algorithm forms an interval of the rescheduling plan for an application in the level, it sends the schedule of the interval to the meta heuristic. The meta heuristic, after receiving the

schedules of the intervals of all the applications in the level, can resolve scheduling conflicts by giving scheduling preference to the critical application with the highest execution time. The meta heuristic then removes the resources scheduled for the application from the available resources, and invokes the MCRP algorithm for the other applications in the level to form schedules for the intervals from the remaining resources. The meta heuristic, by iterating across different intervals of the different applications of a level, coordinating with the MCRP algorithm invocations for the applications, and by traversing the different levels of the tree, can thus form a *global* multi-cluster rescheduling plan for the entire workflow application. A more complicated strategy would be to extend the MCRP algorithm to consider multiple applications instead of a single application, where it uses the performance model functions of the applications in a level to form a set of a non-conflicting intervals for the applications, instead of a single interval for an application. Note that the above strategies can also consider sequential jobs of a workflow since the performance models, scheduling strategies, and rescheduling plans, described in this paper, can also be applied for multi-phase sequential applications.

9 Conclusions and Future Work

This paper described strategies for deciding when and where to reschedule executing tightly-coupled multi-phase parallel applications on multi-cluster Grids. While our earlier efforts discussed performance modeling strategies for predicting execution times of single-phase parallel applications [10], scheduling algorithms for allocating a set of resources for single-phase parallel applications on a single cluster [11], and techniques for malleability of parallel application [12], the current work proposes algorithms for deriving rescheduling plans for adaptive execution of multi-phase parallel applications on single and multiple clusters.

Using large number of simulations of large-scale applications on multi-cluster Grids, it is shown that the rescheduling plans can greatly reduce the application execution times when compared to executions on a single schedule. The genetic algorithm, by exploring diverse rescheduling plans, yielded the most efficient rescheduling plans with 9–12% smaller average execution times than the other algorithms. It is planned to develop a practical rescheduling framework that uses the rescheduling decisions for improving the efficiency of tightly coupled applications on multi-cluster Grids.

References

1. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., Robert, Y.: Centralized versus distributed schedulers for multiple bag-of-task applications. In: 20th International Parallel and Distributed Processing Symposium (2006)
2. Allen, G., Dramlitsch, T., Foster, I., Karonis, N., Ripeanu, M., Seidel, E., Toonen, B. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In: Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM) (2001)
3. Sudarsan, R., Ribbens, C.: ReSHAPE: a framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. In: ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing, p. 44 (2007)
4. Vadhiyar, S., Dongarra, J.: A performance oriented migration framework for the Grid. In: CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, p. 130 (2003)
5. Huang, C., Zheng, G., Kalé, L., Kumar, S.: Performance evaluation of adaptive MPI. In: PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 12–21 (2006)
6. Maghraoui, K., Desell, T., Szymanski, B., Varela, C.: The Internet operating system: middleware for adaptive distributed computing. Int. J. High Perform. Comput. Appl. **20**(4), 467–480 (2006)
7. Wrzesinska, G., Maassen, J., Bal, H.: Self-adaptive applications on the Grid. In: PPoPP '07: Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 121–129 (2007)
8. Hussein, M., Mayes, K., Luján, M., Gurd, J.: Adaptive performance control for distributed scientific coupled models. In: ICS '07: Proceedings of the 21st Annual International Conference on Supercomputing, pp. 274–283 (2007)
9. Desell, T., Maghraoui, K., Varela, C.: Malleable applications for scalable high performance computing. Cluster Comput. **10**(3), pp. 323–337 (2007)
10. Sanjay, H.A., Vadhiyar, S.: Performance modeling of parallel applications for Grid scheduling. J. Parallel Distrib. Comput. **68**(8), 1135–1145 (2008)

11. Sanjay, H., Vadhiyar, S.: Strategies for scheduling tightly-coupled parallel applications on clusters and Grids. Concurr. Comput. **21**(18), 2491–2517 (2009)
12. Vadhiyar, S., Dongarra, J.: SRS–a framework for developing malleable and migratable parallel applications for distributed systems. Parallel Process. Lett. **13**(2), 291–312 (2003)
13. Fernandes, R., Pingali, K., Stodghill, P.: Mobile MPI programs in computational Grids. In: PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 22–31 (2006)
14. Zhang, Y., Koelbel, C., Cooper, K.: Hybrid re-scheduling mechanisms for workflow applications on multi-cluster Grid. In: CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 116–123 (2009)
15. Gong, Y., Pierce, M., Fox, G.: Dynamic resource-critical workflow scheduling in heterogeneous environments. In: Job Scheduling Strategies for Parallel Processing: 14th International Workshop, JSSPP 2009, Rome, Italy, 29 May 2009. Revised Papers, pp. 1–15 (2009)
16. Huedo, E., Montero, R., Llorente, I.: A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. Future Gener. Comput. Syst. **23**(2), 252–261 (2007)
17. Vadhiyar, S., Dongarra, J.: GrADSolve: a Grid-based RPC system for parallel computing with application-level scheduling. J. Parallel Distrib. Comput. **64**(6), 774–783 (2004)
18. Zhang, Y., Koelbel, C., Kennedy, K.: Relative performance of scheduling algorithms in Grid environments. In: CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, pp. 521–528 (2007)
19. Zhang, Y., Koelbel, C., Cooper, K.: Cluster-based hybrid scheduling mechanisms for workflow applications on the Grid. In: IEEE Fourth International Conference on eScience, pp. 390–391 (2008)
20. Sakellariou, R., Zhao, H.: A low-cost rescheduling policy for efficient mapping of workflows on Grid systems. Sci. Program. **12**(4), 253–262 (2004)
21. Elmroth, E., Tordsson, J.: A standards-based Grid resource brokering service supporting advance reservations, coallocation, and cross-Grid interoperability. Concurr. Comput. **21**(18), 2298–2335 (2009)
22. Dumitrescu, C., Raicu, I., Foster, I.: The design, usage, and performance of GRUBER: a Grid usage service level agreement based BrokERing infrastructure. J. Grid Computing **5**(1), 99–126 (2007)
23. Moltó, G., Hernández, V., Alonso, J.: A service-oriented WSRF-based architecture for metascheduling on computational Grids. Future Gener. Comput. Syst. **24**(4), 317–328 (2008)
24. Adzigogov, L., Soldatos, J., Polymenakos, L.: EMPEROR: an OGSA Grid meta-scheduler based on dynamic resource predictions. J. Grid Computing **3**(1–2), 19–37 (2005)
25. Foster, I.: Globus toolkit version 4: software for service-oriented systems. In: IFIP International Conference on Network and Parallel Computing. LNCS, vol. 3779, pp. 2–13. Springer, Berlin (2006)
26. WS Resource Framework. http://www.globus.org/wsrf
27. Czajkowski, K., Foster, I., Kesselman, C.: Agreement-based resource management. Proc. IEEE **93**(3), 631–643 (2005)
28. Zhang, X., Freschl, J., Schopf, J.: A performance study of monitoring and information services for distributed systems. In: HPDC '03: Proceedings of the 12th IEEE International Symposiumon High Performance Distributed Computing, p. 270 (2003)
29. Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I.: The globus striped GridFTP framework and server. In: Proceedings of Super Computing 2005 (SC05) (2005)
30. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S.: Security for Grid services. In: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, p. 48 (2003)
31. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G., Good, J., Laity, A., Jacob, J., Katz, D.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**(3), 219–237 (2005)
32. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia (1997)
33. Wolski, R., Spring, N., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**(5–6), 757–768 (1999)
34. Shen, X., Zhong, Y., Ding, C.: Predicting locality phases for dynamic memory optimization. J. Parallel Distrib. Comput. **67**(7), 783–796 (2007)
35. Shen, X., Scott, M., Zhang, C., Dwarkadas, S., Ding, C., Ogihara, M: Analysis of input-dependent program behavior using active profiling. In: ExpCS '07: Proceedings of the 2007 Workshop on Experimental Computer Science, p. 5 (2007)
36. Ding, C., Dwarkadas, S., Huang, M., Shen, K., Carter, J.: Program phase detection and exploitation. In: 20th International Parallel and Distributed Processing Symposium (2006)
37. ChaNGa (Charm N-body GrAvity Solver). http://librarian.phys.washington.edu/astro/index.php/Research:ChaNGa
38. Athena Code Home Page. http://www.astro.princeton.edu/ jstone/athena.html
39. LAMMPS Molecular Dynamics Simulator. http://lammps.sandia.gov
40. MIT Photonic-Bands (MPB). http://ab-initio.mit.edu/wiki/index.php/MIT_Photonic_Bands

41. Dinda, P., O'Hallaron, D.: Host load prediction using linear models. Cluster Comput. **3**(4), 265–280 (2000)
42. Dinda, P.: Online prediction of the running time of tasks. Cluster Comput. **5**(3), 225–236 (2002)
43. Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling strategies for mapping application workflows onto the Grid. In: HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium, pp. 125–134 (2005)
44. Fox, G., Gannon, D.: Workflow in Grid systems. Concurr. Comput. **18**(10), 1009–1019 (2006)
45. Montagnat, J., Glatard, T., Plasencia, I., Castejn, F., Pennec, X., Taffoni, G., Voznesensky, V., Vuerli, C.: Workflow-based data parallel applications on the EGEE production Grid infrastructure. J. Grid Computing **6**(4), 369–383 (2008)
46. Ramakrishnan, L., Koelbel, C., Kee, Y.-S., Wolski, R., Nurmi, D., Gannon, D., Obertelli, G., YarKhan, A., Mandal, A., Huang, T., Thyagaraja, K., Zagorodnov, D.: VGrADS: enabling e-science workflows on Grids and clouds with fault tolerance. In: SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–12 (2009)
47. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**(5), 528–540 (2009)
48. Yu, J., Buyya, R.: A taxonomy of workflow management systems for Grid computing. J. Grid Computing **3**(3–4), 171–200 (2005)