J Supercomput (2010) 53: 163–181
DOI 10.1007/s11227-009-0326-1

Robust task scheduling for volunteer computing systems

Young Choon Lee · Albert Y. Zomaya · Howard Jay Siegel

Published online: 4 September 2009
© Springer Science+Business Media, LLC 2009

Abstract Performance perturbations are a natural phenomenon in volunteer computing systems. Scheduling parallel applications with precedence-constraints is emerging as a new challenge in these systems. In this paper, we propose two novel robust task scheduling heuristics, which identify best task-resource matches in terms of makespan and robustness. Our approach for both heuristics is based on a proactive reallocation (or schedule expansion) scheme enabling output schedules to tolerate a certain degree of performance degradation. Schedules are initially generated by focusing on their makespan. These schedules are scrutinized for possible rescheduling using additional volunteer computing resources to increase their robustness. Specifically, their robustness is improved by maximizing either the total allowable delay time or the minimum relative allowable delay time over all allocated volunteer resources. Allowable delay times may occur due to precedence constraints. In this paper, two proposed heuristics are evaluated with an extensive set of simulations. Based on simulation results, our approach significantly contributes to improving the robustness of the resulting schedules.

Keywords Scheduling · Volunteer computing · Robustness · Resource allocation · Distributed systems

Y.C. Lee · A.Y. Zomaya (✉)
Centre for Distributed and High Performance Computing, School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia
e-mail: zomaya@it.usyd.edu.au

Y.C. Lee
e-mail: yclee@it.usyd.edu.au

H.J. Siegel
Department of Electrical and Computer Engineering and Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1373, USA
e-mail: HJ@ColoState.edu

1 Introduction

A volunteer computing system (VCS) is a distributed computing platform in which a large number of individually owned computing resources are voluntarily donated to science projects. Many large-scale science projects (e.g., SETI@home [1], Folding@home [2], and Einstein@Home [3]) have been successfully deployed and tackled on these systems. Computing resources participating in VCSs are characterized by their essentially autonomous, heterogeneous, distributed, and dynamic nature. Unlike resources in grids, those in VCSs are typically not tied to any managed organization—most of them are personal computers connected via the internet.

As this new computing paradigm gains more recognition for its high-throughput and high-performance potential, many science and engineering problems are being deployed onto VCSs. While applications in most current volunteer computing projects are independent or bag-of-tasks/parameter sweep applications (a series of independent tasks with different input parameters), a considerable group of scientific and engineering applications in practice are precedence-constrained parallel applications. The problem of scheduling applications in the latter model both on homogeneous and heterogeneous computing systems has been studied extensively over the past few years (e.g., [4–11]). However, most efforts in task scheduling have focused primarily on two issues, the minimization of application completion time (*makespan*/*schedule length*), and reduction of time complexity; in other words, the main objective of a task scheduling algorithm is the generation of the optimal schedule for a given application with the minimal amount of scheduling time. It is only recently that much attention has been paid to robustness in scheduling, particularly on parallel and distributed computing systems [12–18].

Broadly, robustness can be defined as the capacity to function properly in the presence of variable conditions. In the context of task scheduling particularly for VCSs, the definition may be narrowed down to a guarantee that the quality of a schedule is assured in spite of a certain degree of performance fluctuation, such as incorrect estimates of task completion times and resource performance degradation. This performance characteristic has a particular importance in the case of scheduling for VCSs in which resources behave in a quite unmanaged fashion. For example, a volunteered computer may be turned off or returned to personal use. In such circumstances, VC project tasks on such a computer will be either suspended or terminated, or lowered in priority (resulting in slower execution).

Robustness is closely related to fault tolerance. Fault tolerance is mostly associated with task/resource failures. Rescheduling and task replication are two most frequently used techniques to deal with those failures. While these techniques can be applied for robust scheduling, robustness in "output schedules" (i.e., the assignment and ordering of tasks on computers produced by a resource allocation heuristic) can also be significantly improved if scheduling decisions are made taking into account performance perturbations.

In this paper, we address the task scheduling problem in VCSs and propose two novel scheduling heuristics (*RMAX* and *RMXMN*) that take into account robustness as well as makespan. Our approach in the case of both heuristics is based on a proactive reallocation (or schedule expansion) scheme enabling output schedules to tolerate a

certain degree of performance degradation. Schedules are initially generated focusing on their makespan; this is used as the lower bound. These schedules are scrutinized for possible rescheduling using additional VC resources to increase their robustness. For a given schedule, rescheduling occurs only if a new task-resource match improves the robustness without increasing the makespan. Let the "allowable delay time" of a task in a schedule refer to the amount of time that can be delayed to start or complete the task without incurring an increase in makespan; this may occur due to precedence constraints. Specifically, rescheduling decisions in *RMAX* and *RMXMN* are made aiming to maximize the total allowable delay time of all tasks in that schedule, or maximize the minimum allowable delay time to computation time ratio over all allocated VC resources. The two heuristics have been evaluated with an extensive set of simulations. Based on simulation results, our approach significantly contributes to improving the robustness of schedules.

The remainder of the paper is organized as follows. Section 2 describes the application, system, robustness, and scheduling models used in this paper. Section 3 overviews related work. The *RMAX* and *RMXMN* algorithms are presented in Sect. 4 followed by results in Sect. 5. Finally, conclusions are provided in Sect. 6. We have provided a table of acronyms used in this paper in Appendix.

2 Models

In this section, we describe the system, application, robustness, and scheduling models employed in this work.

2.1 System model

The target system used in this work consists of a set R of r heterogeneous computing resources (VC resources) that are fully interconnected in the sense that a route exists between any two individual resources (Fig. 1). Each VC resource is in an autonomous administrative domain that has its own local users who access the resources provided by it. Typically, the number of available VC resources at a given time is far larger than the number of tasks to be scheduled. These resources are not entirely dedicated to the VCS. In other words, they are used for both local and VC jobs. These resources are connected to each other through a wide area network. The interresource communications are heterogeneous. All network links are assumed to work without substantial contentions. It is also assumed that a message can be transmitted from one resource to another while a task is being executed on the recipient resource, which is possible in many systems.

The availability and capacity of resources, such as, VC resources and network links, fluctuates. Therefore, the accurate completion time of a job on a particular resource is difficult, if not impossible, to determine a priori. Moreover, the job may fail to complete due to a failure of the resource on which it is running. However, resource failures are not considered in this study.

2.2 Application model

Parallel programs, in general, can be represented by a directed acyclic graph (DAG). A DAG, $G = (N, E)$, consists of a set N of n nodes and a set E of e edges. A DAG is

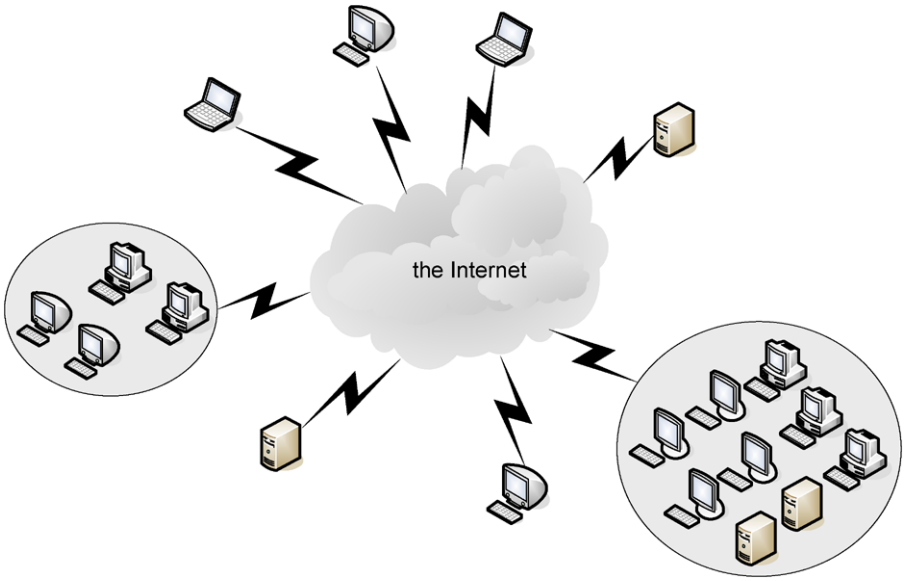


Fig. 1 VCS model

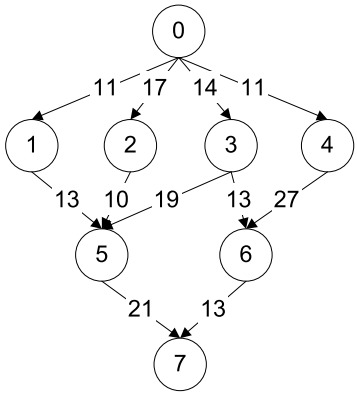
also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application, and the edges represent precedence constraints. An edge $(i, j) \in E$ between task n_i and task n_j also represents intertask communication. In other words, the output of task n_i has to be transmitted to task n_j in order for task n_j to start its execution. A task with no predecessors is called an *entry* task, n_{entry}, whereas an *exit* task, n_{exit}, is one that does not have any successors. Among the predecessors of a task n_i, the predecessor which completes the communication at the latest time is called the most influential parent (*MIP*) of the task denoted as $MIP(n_i)$. The longest path of a task graph is the critical path (CP).

The weight on a task n_i denoted as w_i represents the computation cost of the task. In addition, the computation cost of the task on a VC resource r_j, is denoted as $w_{i,j}$ and its average computation cost is denoted as $\overline{w_i}$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication time between two tasks, n_i and n_j. For the purposes of this study, it is assumed that the bandwidths between any pairs of resources are virtually the same. However, the time to send a data set between any two tasks will depend on the size of the data sets, which can vary. Thus, the time $c_{i,j}$ is based on the common bandwidth between all pairs of resources (the same for all pairs) the size of the data set being moved between the two tasks (which can differ for each pair of communicating tasks). However, a communication cost is only required when two tasks are assigned to different resources. In other words, the communication cost when tasks are assigned to the same VC resource can be ignored, i.e., 0.

The earliest start time and the earliest finish time of a task n_i on a VC resource r_j are defined as follows, where $MIP(n_i)$ is executed on machine r_k:

Fig. 2 A simple task graph



$$EST(n_i, r_j) = \begin{cases} 0 & \text{if } n_i = n_{\text{entry}} \\ EFT(MIP(n_i), r_k) + c_{MIP(n_i),i} & \text{otherwise,} \end{cases} \quad (1)$$

$$EFT(n_i, r_j) = EST(n_i, r_j) + w_{i,j} \quad (2)$$

Note that, the actual start and finish times of a task n_i on a resource r_j, denoted as $AST(n_i, r_j)$ and $AFT(n_i, r_j)$ can be different from its earliest start and finish times, $EST(n_i, r_j)$ and $EFT(n_i, r_j)$, if the actual finish time of another task scheduled on the same resource is later than $EST(n_i, r_j)$.

In the case of adopting task insertion, the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the resource as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing resource utilization for a communication intensive task graph with fine-grain tasks.

A simple task graph is shown in Fig. 2 with its details in Tables 1 and 2. The nodes are the tasks and the edges are data set transfers between nodes. The values presented in Table 1 are computed using two task prioritization methods, *t-level* and *b-level*, frequently used in many list scheduling heuristics, e.g., [6, 8]. The weight on the edge from node i to node j is the time $c_{i,j}$. Note that computation costs are averaged over all nodes and links, as shown in Table 2. The t-level of a task is defined as the summation of the computation and communication costs along the longest (in terms of time) path of the node from the entry task in the task graph. The task itself is excluded from the computation. In contrast, the b-level of a task is computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph (including the task). The b-level is used in this study.

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication time between all pairs of communicating tasks divided by the average computation time for all tasks on a target system.

Table 1 Task priorities

Task	b-level	t-level
0	101.33	0.00
1	66.67	22.00
2	63.33	28.00
3	73.00	25.00
4	79.33	22.00
5	41.67	56.33
6	37.33	64.00
7	12.00	89.33

Table 2 Computation cost

Task	r_0	r_1	r_2	Average
0	11	13	9	11
1	10	15	11	12
2	9	12	14	12
3	11	16	10	12
4	15	11	19	15
5	12	9	5	9
6	10	14	13	12
7	11	15	10	12

2.3 Robustness model

In essence, our robustness metric is derived from latest start/finish times of tasks. The latest start and finish times of a task n_i on a VC resource r_j are defined as:

$$LST(n_i, r_j) = LFT(n_i, r_j) - w_{i,j}, \tag{3}$$

$$LFT(n_i, r_j) = \begin{Bmatrix} EFT(n_i) & \text{if } n_i = n_{\text{exit}} \\ \min\limits_{n_k \in succ(n_i)} \{LST(n_k, r_m) - c_{i,k}\} & \text{otherwise} \end{Bmatrix} \tag{4}$$

where $succ(n_i)$ is the set of successor tasks of task n_i, and r_m is the VC resource executing task n_k. For a given task, its *LST*/*LFT* may differ from *EST*/*EFT* if the minimum *EST* in all of its successor tasks is later than the time it finishes its communication (data transfer); once again, this may occur due to precedence constraints. Note that, *LST* and *LFT* of a task have a cumulative effect on those of its parent tasks. Similar to the case for *AST* and *AFT*, the actual latest start and finish times of a task n_i on a VC resource r_j are defined as follows, where $ALST(n_{\text{next}}, r_j)$ is the actual latest start time of the next task scheduled after n_i on the same VC resource r_j:

$$ALST(n_i, r_j) = ALFT(n_i, r_j) - w_{i,j}, \tag{5}$$

$$ALFT(n_i, r_j) = \left\{ \begin{array}{l} AFT(n_i) \quad \text{if } n_i = n_{\text{exit}} \\ \min\left\{ \min\limits_{n_k \in succ(n_i)} (ALST(n_k, r_m) - c_{i,k}), ALST(n_{\text{next}}, r_j) \right\} \\ \quad \text{otherwise} \end{array} \right\} \quad (6)$$

The degree of robustness for a particular task-resource match can be measured based on *ALST* and/or *ALFT*, since they enable the quantification of "allowable" delay in the completion of the task. The late completion of a task does not affect the makespan of a given precedence-constrained parallel application as long as the time of the completion is no later than the *ALFT* of the task; hence, the current task-resource match in the schedule for this application is robust to a certain degree.

For a given schedule, the average robustness ratio (*RR*) associated with a particular VC resource r_j is defined as follows, where N_j is the set of tasks assigned on r_j:

$$RR(r_j) = \frac{\sum_{n_i \in N_j} ALST(n_i, r_j) - AST(n_i, r_j)}{\sum_{n_i \in N_j} w_{i,j}} \quad (7)$$

The average robustness ratio of the schedule is then as follows, where R^* is the set of resources used in the schedule:

$$RR = \frac{\sum_{r_j^* \in R^*} RR(r_j^*)}{|R^*|} \quad (8)$$

The maximization of *RR* is directly related to the improvement in robustness; however, this should be achieved without an increase in makespan.

2.4 Scheduling model

The task scheduling problem in this study is the process of assigning a set N of n tasks to a set R of r heterogeneous VC resources—without violating precedence constraints—aiming to minimize makespan with a robustness factor as large as possible. Traditionally, the former (the minimization of makespan) is the single most important objective in the task scheduling problem.

In VCSs, this sole focus of makespan minimization is neither sufficient nor appropriate due to performance perturbations of resources. Rather, the robustness of output schedules should be explicitly taken into account. However, these two objectives conflict with each other. In other words, the minimization of makespan is often achieved by "compacting tasks" whereas the maximization of robustness is obtained by increasing slack times between tasks. The scheduling model in this study can be seen as a multi-pass (two-pass) model—the schedule generation pass and the robustness improvement pass. Each of the two objectives in this study is attempted to be optimized through the relevant pass. Specifically, the first pass is dedicated to the generation of schedules focusing solely on their makespan. In the second and last pass, these schedules are inspected with an additional set of VC resources for possible rescheduling to improve their robustness.

The makespan is defined as $M = \max\{AFT(n_{\text{exit}})\}$ after the scheduling of n tasks in a task graph G is completed. Although the minimization of makespan is crucial, tasks of a DAG in our study are not associated with deadlines as in real-time systems. The improvement of robustness is measured based on *RR* as defined in Sect. 2.3.

3 Related work

Some of the common approaches that deal with performance fluctuations in dynamic environment, such as, VCSs and grids include robust scheduling, task replication/migration, and rescheduling. In general, the first approach implicitly and proactively handles uncertainties in task/resource performance by generating robust schedules that are insensitive to a certain degree of performance perturbations. On the contrary, the rest are reactive approaches; that is, when performance degradation (e.g., task/resource failures) is detected, they react by taking an appropriate step to minimize negative effects from such an event.

Probably, the closest works to our study are static heuristics proposed in [15]. The target system is a rather specialized platform (e.g., a military system) consisting of heterogeneous sets of sensors, applications, machines, and actuators. Although such a system model is specific to a particular environment, it can be easily generalized for heterogeneous computing systems with dynamic performance fluctuations. Those two heuristics are devised with the robustness metric developed in [12]. The metric is used to identify the maximum amount of performance changes that a given schedule can tolerate without violating performance constraints, e.g., the makespan of the schedule is within 120% of the initial makespan. Initial schedules (chromosome or state) for those two heuristics are generated using a mixture of another two heuristics—most critical task first (*MCTF*) and most critical path first (*MCPF*)—also proposed in [15]. The evaluation was conducted based on a number of performance criteria including robustness and failure rate (i.e., how often schedules cannot meet or fail to meet performance criteria).

The authors in [19] introduced two different robust multiobjective optimization procedures that can be applied to the task scheduling problem in our study. While most other approaches developed for (evolutionary) multiobjective optimization problems focus on finding the global Pareto-optimal solutions (frontier), these procedures were intended to identify a robust solution/frontier. This means that one or more solutions that are relatively more capable of tolerating variable perturbations compared with other neighboring solutions. The first procedure achieves its performance goal by minimizing the mean effective objectives; they are obtained by averaging a finite set of neighboring solutions. The second procedure performs its search for robust solutions with a user-defined limit (constraint) that restricts the change in objective values within a predefined threshold.

There have been a large number of fault-tolerant scheduling algorithms—for heterogeneous computing systems—proposed in the literature; however, most of them are for applications consisting of independent tasks (e.g., [20]). Significant previous efforts on scheduling precedence-constraint task graphs with fault-tolerance/reliability include *RHEFT* in [21], *RDSL* and its variations in [22], and *FTSA* and *MC-FTSA* in [23]. In the first two works, failure models (more precisely, failure rates) were incorporated into their scheduling to ensure the successful completion of applications by increasing the reliability of output schedules. On the other hand, *FTSA* and *MC-FTSA* adopt task replication to deal with resource failures.

Another interesting approach to note is the scheduling scheme based on dedication rate (*SSDR*) in [24]. Its scheduling decisions are made taking into account both temporal-availability and spatial-availability rates of VC resources for VC jobs.

4 Robust scheduling heuristics

As in most multiobjective optimization problems, the goal in our task scheduling problem is to find Pareto-optimal solutions since the performance objectives of the problem most likely to be in conflict with each other. In other words, for a given task graph, the heuristics presented in this study aim to minimize makespan, while the robustness of the schedule is maximized with the makespan as a constraint. More formally, a multiobjective optimization problem can be defined as

$$\min_{x \in S}[f_1(x), f_2(x), \ldots, f_n(x)] \tag{9}$$

where S is the feasible search space and $n \geq 2$.

Next, we present two robust scheduling heuristics, *RMAX* and *RMXMN* (Figs. 3 and 4), for which the main objective is to maximize the robustness of schedules to minimize the increase—caused by resource and/or task performance fluctuations—in makespans of those schedules.

4.1 *RMAX*

Since the number of VC resources typically far exceeds the number of tasks for a given parallel application, it is quite necessary to select a certain number of resources (possibly much fewer than the number of available resources) for scheduling. The resource selection process in *RMAX* is enabled using a greedy method. Specifically, for each available resource, the summation of execution times of all tasks is identified,

Algorithm *RMAX*
Input: A task graph *G(N, E)*, and a set R of r VC resources
Output: A schedule of G onto R
1. Compute *b-level* of $\forall n_i \in N$
2. Sort N in decreasing order by *b-level* value
3. Select a set R^* of best VC resources
4. Group R^* into sets R' and R'' based on resource performance
5. **for** $\forall n_i \in N$ **do**
6. Select $r_j^* \in R'$ on which $AFT(n_i, r_j^*)$ is minimized
7. **end for**
8. Compute *ALST*s for $\forall n_i \in N$
9. Let S = the sum of *ALST*s
10. **for** $\forall n_i \in N$ **do**
11. Remove n_i from its current resource
12. **if** there exists $r_j^* \in R''$ on which S is increased without increasing M or M is shortened **then**
13. Assign n_i to r_j^*
14. **else**
15. Restore the previous resource allocation for n_i
16. **end if**
17. **end for**

Fig. 3 The *RMAX* algorithm

Algorithm *RMXMN*
Input: A task graph *G(N, E)*, and a set *R* of *r* VC resources
Output: A schedule of *G* onto *R*
1. Compute *b-level* of $\forall n_i \in N$
2. Sort *N* in decreasing order by *b-level* value
3. Select a set *R** of best VC resources
4. Group *R** into sets *R′* and *R″* based on resource performance
5. **for** $\forall n_i \in N$ **do**
6. Select $r_j^* \in R'$ on which $AFT(n_i, r_j^*)$ is minimized
7. **end for**
8. Compute *ALST*s for $\forall n_i \in N$
9. Compute $RR(r'_j)$ for $\forall r'_j \in R'$
10. Let $MRR = \min_{r_j' \in R'} (RR(r_j'))$
11. Let r^m = the *MRR* resource
12. Let N^m = the set of tasks on r^m
13. **for** $\forall n_i^m \in N^m$ **do**
14. Remove n_i^m from r^m
15. **if** there exists $r_j^* \in R''$ on which *MRR* is increased without increasing *M* or *M* is shortened **then**
16. Assign n_i^m to r_j^*
17. **else**
18. Reassign n_i^m to r^m
19. **end if**
20. **end for**
21. Repeat Steps 10 to 20 if the *MRR* resource has changed

Fig. 4 The *RMXMN* algorithm

and a predefined number of best resources are selected based on completion time. These selected resources are divided into two sets, such that each set contains the same number of resources. Although the number of resources to select is much larger and they can be divided into more than two sets, it is impractical in terms of the time complexity of the algorithm. Besides, the performance of selected resources may vary significantly resulting in some resources are never used, but again they just contribute to increase the time complexity. Note that resources in the first set might be better in terms of processing speed than those in the second set. However, this performance characteristic is measured based on the overall completion time; that is, for a particular task, some resources in the second set may deliver superior performance.

RMAX starts its scheduling with the first set only, concentrating on the minimization of makespan (Steps 5–7). The first set is then expanded using the second set. Now, this aggregated set is used to rearrange the current schedule to improve its robustness by increasing the total amount of allowable delay time (Steps 11–18).

The most significant effect of resource expansion and task rearrangement processes using the second set is the recovery of *LST*s or an increase of *ALST*s. Specifically, for a particular task with its *ALST* earlier than its *LST* due to one or more subsequent tasks on the same resource, its *ALST* can be recovered or at least can be increased by rearranging those subsequent tasks using the resources of the second set.

4.2 *RMXMN*

RMXMN adopts the same initial makespan minimization and resource expansion processes as those used in *RMAX*. The main difference between *RMXMN* and *RMAX* is in *RMXMN*'s task rearrangement method (or robustness improvement method). The rationale behind this method is to maximize the minimum robustness rate (*MRR*) over all allocated VC resources. Clearly, the resource with the *MRR* (or *MRR* resource) is most vulnerable to performance perturbations.

It can be said that a schedule is guaranteed on average to tolerate performance perturbations to the limit of its *MRR*. *RMXMN* applies its robustness improvement method to each task on the *MRR* resource (Steps 13–20). If the *MRR* resource has changed after some tasks on the previous *MRR* resource are rearranged, *RMXMN* performs the method with tasks on the new *MRR* resource; this repeats until no further improvement on *MRR* is possible.

5 Performance evaluation

In this section, we describe experimental methods and settings including task graph characteristics and their generation, and resource performance settings. Experimental results are then presented based on the makespan increase rate (*MIR*), i.e., the percentage of makespan increase. More formally,

$$MIR = \frac{M^a - M^e}{M^e} \tag{10}$$

where M^a is the actual makespan after the execution of a given application with a certain degree of performance perturbation, and M^e is the makespan estimated with an assumption that the performance information is accurate.

5.1 Experimental settings

The performance of *RMAX* and *RMXMN* was thoroughly evaluated with a large number of simulations using both random task graphs and real-world application task graphs. Table 3 summarizes the parameters used in our experiments. These parameters were chosen to ensure that the simulations are extensive enough and cover a wide range of scenarios.

The number of experiments performed using different random task graphs on the three different resource sizes with three resource heterogeneity settings is 9,000. Specifically, the random task graph set consisted of 50 base task graphs generated with combinations of 10 graph sizes and five CCRs. For each combination, 20 task graphs were randomly generated, retaining the characteristics of the base task graph. These 1,000 graphs were investigated with three different resource settings and three resource heterogeneity settings; hence, the figure 9,000.

The computation and communication times of the tasks in each task graph were randomly selected from a uniform distribution with the mean equal to the chosen average computation and communication times. The out degree of each node in a

Table 3 Experimental parameters

Parameter	Value
The number of tasks	U(10, 400)
CCR	{0.1, 0.2, 1.0, 5.0, 10.0}
The number of resources	{16, 32, 64}
Out degree of a node	U(1, 10)
Resource heterogeneity	{100, 200, random}
Avg. performance degradation rate	{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7}

task graph was random and uniformly distributed between 1 and 10. Specifically, for a node, an out degree of one indicates that it has only one successor task. A resource heterogeneity value of 100 was defined to be the percentage of the speed difference between the fastest resource and the slowest resource in a given system.

The three real-world parallel applications used for our experiments were the Laplace equation solver [25], the LU-decomposition [26], and Fast Fourier Transformation [27]. A large number of variations (i.e., 1,350 task graphs for each application) were made on these task graphs for more comprehensive experiments. These variations were made using experimental parameters in Table 3. In addition, the matrix sizes and the number of input points were varied, so that the number of tasks can range from about 10 to 400.

To introduce resource performance perturbations (performance estimation errors), we used a Gaussian random number generator with different average performance fluctuation rates. An average performance degradation rate of 0.3 (or 30%) indicates that resource performance is on average at 70% of its full capacity.

5.2 Results

Since our main focus is the robustness of schedules, the performance evaluation of our heuristics is solely based on *MIR*. This metric is straightforward, yet important for identifying and comparing the impact of performance fluctuation on schedules generated by different scheduling algorithms accounting for robustness. Our simulation results are summarized and presented in Figs. 5, 6, and 7. Note that, *HEFT* (a well-known scheduling algorithm that does not consider robustness) [6] was used in our comparisons to identify the extent of the contribution of our heuristics to the improvement in robustness.

The overall performance comparisons for random task graphs and real-world application task graphs are presented in Figs. 5 and 6, respectively. The results in each of these two figures are a consolidation of results obtained with five different CCRs (0.1, 0.2, 1.0, 5.0, and 10.0). Clearly, our heuristics show their capability to generate schedules that are relatively more robust compared to those produced by *HEFT*. Specifically, schedules generated by *RMAX* and *RMXMN* were on average 5% and 7% more robust than those generated by *HEFT*, respectively. It is noted that robustness of schedules for real-world application task graphs was slightly worse than that for random task graphs due to the regularity of real-world application task graphs. That is, (1) tasks in the same level (or with the same height) are often homogeneous

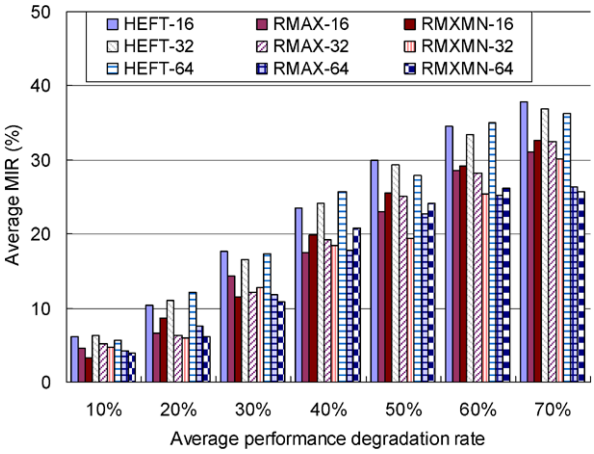


Fig. 5 Robustness of schedules generated by different heuristics with respect to different resource sizes

in terms of both their computation and communication costs, and (2) the width of task graphs is typically much larger than that of random task graphs. These two factors tend to make tasks to be compacted. In many cases, it is also not quite possible to reschedule tasks without increasing the original makespan.

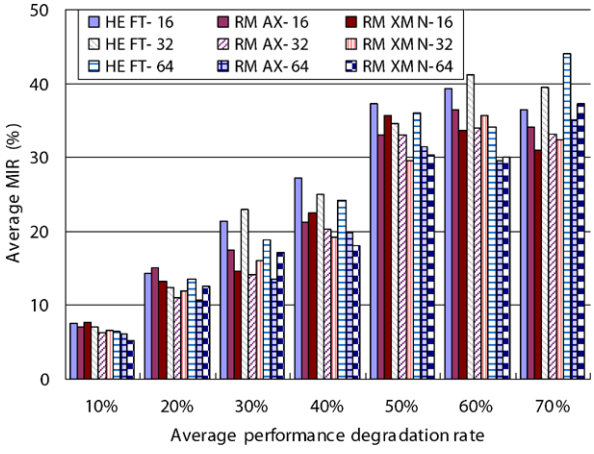
Since the number of tasks to be scheduled at any one time tends not to overwhelm the number of available resources in our simulation environments, the results obtained from experiments with different numbers of resources do not significantly differ; this is particularly true for *HEFT*. For example, the difference between average *MIR*s of *HEFT* with 16 or with 32 resources is marginal.

Although the experiments were carried out with five different CCRs as stated in Table 3, only experimental results obtained with three significant CCRs (0.1, 1.0 and 10.0) are explicitly presented in Fig. 7. This is because these results were sufficient to represent the performance of our heuristics for three fundamental task graph types (computationally intensive, moderate, and communication-intensive). The rest of the test results obtained from the task graphs with CCRs of 0.2 and 5.0 tend to be similar to those obtained from the task graphs with close CCRs. For instance, the test result acquired from the task graphs with a CCR of 5.0 does not show significant difference from the test result acquired from the task graphs with a CCR of 10.0. Note that Fig. 7 is plotted using results obtained with random task graphs, since results obtained with real-world application task graphs showed similar patterns.

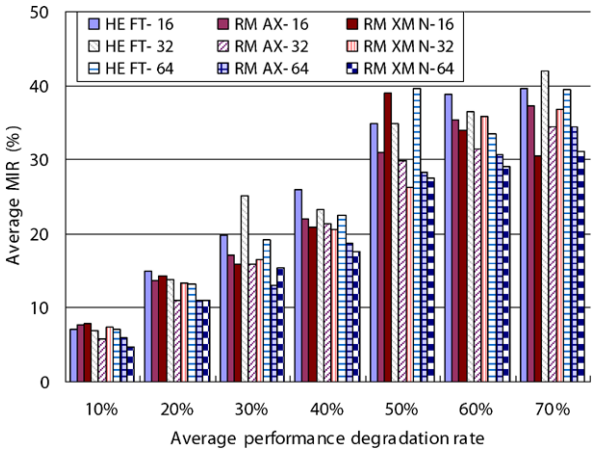
Because allowable delay times often occur due to precedence constraints, communication intensive applications tend to be scheduled with better robustness as shown in Fig. 7c.

6 Conclusions

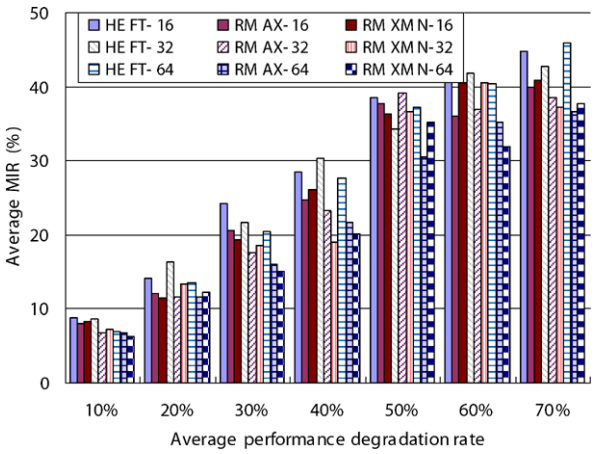
Resources in VCSs cannot be assumed to be reliable particularly in terms of performance. Therefore, the robustness of output schedules is an important quality of service consideration. We have addressed this issue in the context of task scheduling for VCSs, and presented *RMAX* and *RMXMN*, robust scheduling heuristics for dynamic



(a)

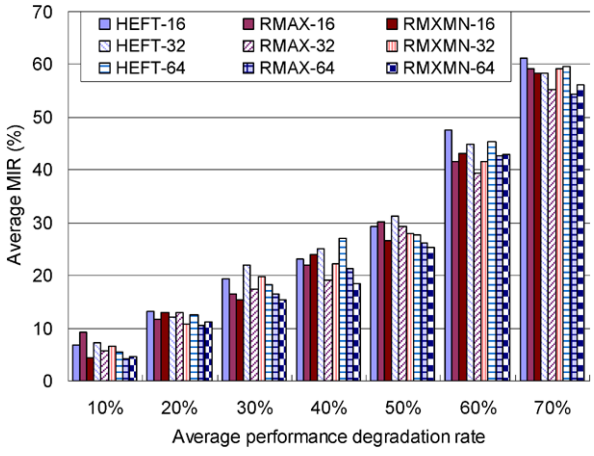


(b)

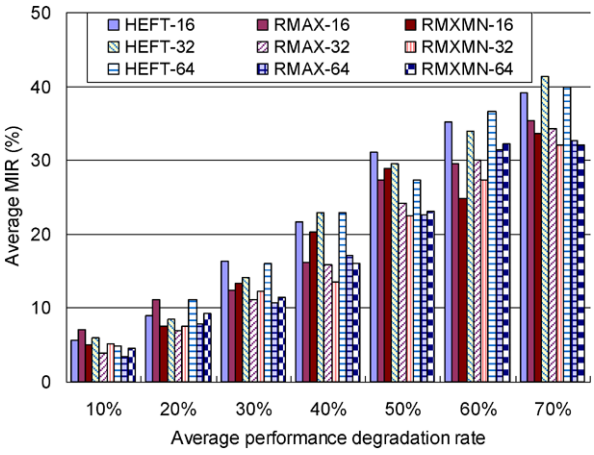


(c)

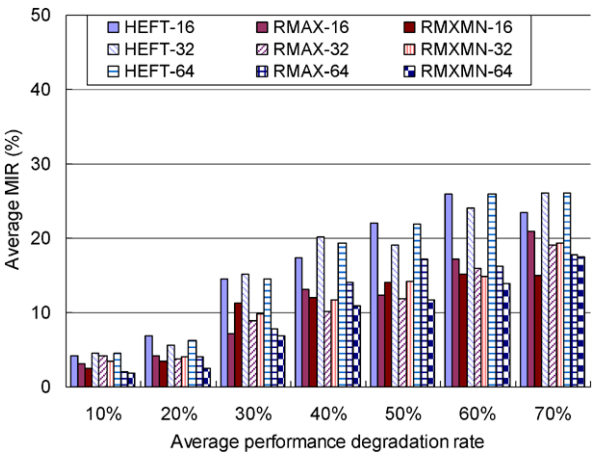
Fig. 6 Robustness of schedules generated by different heuristics with respect to different application types. (**a**) Laplace. (**b**) FFT. (**c**) LU-decomposition



(a)



(b)



(c)

Fig. 7 Robustness of schedules generated by different heuristics with respect to different CCRs. (**a**) CCR = 0.1. (**b**) CCR = 1.0. (**c**) CCR = 10.0

VCSs based on a proactive reallocation scheme. The maximization of allowable delay time for a given output schedule can be an effective technique to improve the robustness of the schedule in the presence of performance perturbations. Our extensive experiments verified this claim. Another significant characteristic of *RMAX* and *RMXMN* is that their performance in terms of makespan is also quite competitive, especially compared with *HEFT*.

We plan to extend the work in this study to more comprehensive robust scheduling taking into account the availability of resources (e.g., resource failures), as resources in VCSs can join and disconnect at any time without advance notice. Task replication can be an effective technique to deal with this resource reliability issue and also to ensure successful completion of applications. However, there are a number of complex problems involved with this technique, such as identifying candidate tasks for replication and determining the number of replicas.

Acknowledgements Professor H.J. Siegel's work is supported by the USA National Science Foundation (NSF) under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment. Professor A.Y. Zomaya's work is supported by an Australian Research Grant DP0667266.

Appendix

Table 4 Table of acronyms

Acronym	Full term
VC	volunteer computing
VCS	volunteer computing system
DAG	directed acyclic graph
MIP	most influential parent
CP	critical path
EST	earliest start time
EFT	earliest finish time
AST	actual start time
AFT	actual finish time
CCR	communication to computation ratio
LST	latest start time
LFT	latest finish time
ALST	actual latest start time
ALFT	actual latest finish time
RR	robustness ratio
MCTF	most critical task first
MCPF	most critical path first
SSDR	scheduling scheme based on dedication rate
MRR	minimum robustness rate
MIR	makespan increase rate
HEFT	heterogeneous earliest finish time

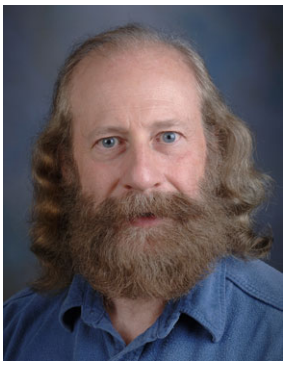
References

1. Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) SETI@home: an experiment in public-resource computing. Commun ACM 45(11):56–61
2. Folding@home (2009) http://folding.stanford.edu/
3. Einstein@Home (2009) http://einstein.phys.uwm.edu/
4. Darbha S, Agrawal DP (1998) Optimal scheduling algorithm for distributed-memory machines. IEEE Trans Parallel Distrib Syst 9(1):87–95
5. Zomaya AY, Ward C, Macey BS (1999) Genetic scheduling for parallel processor systems: comparative studies and performance issues. IEEE Trans Parallel Distrib Syst 10(8):795–812
6. Topcuouglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
7. Lee YC, Zomaya AY (2008) A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 19(9):1215–1223
8. Lee YC, Subrata R, Zomaya AY (2009) On the performance of a dual-objective optimization model for workflow applications on grid platforms. IEEE Trans Parallel Distrib Syst 20(9):1273–1284
9. Shivle S, Sugavanam P, Siegel HJ, Maciejewski AA, Banka T, Chindam K, Dussinger S, Kutruff A, Penumarthy P, Pichumani P, Satyasekaran P, Sendek D, Smith J, Sousa J, Sridharan J, Velazco J (2005) Mapping subtasks with multiple versions on an *ad hoc* grid. Parallel Comput 31(7):671–690. Special Issue on Heterogeneous Computing
10. Shivle S, Siegel HJ, Maciejewski AA, Sugavanam P, Banka T, Castain R, Chindam K, Dussinger S, Pichumani P, Satyasekaran P, Saylor W, Sendek D, Sousa J, Sridharan J, Velazco J (2006) Static allocation of resources to communicating subtasks in a heterogeneous *ad hoc* grid environment. J Parallel Distrib Comput 66(4):600–611. Special Issue on Algorithms for Wireless and Ad-hoc Networks
11. Braun TD, Siegel HJ, Maciejewski AA, Hong Y (2008) Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. J Parallel Distrib Comput 68(11):1504–1516
12. Ali S, Maciejewski AA, Siegel HJ, Kim J-K (2004) Measuring the robustness of a resource allocation. IEEE Trans Parallel Distrib Syst 15(7):630–641
13. Smith J, Briceño LD, Maciejewski AA, Siegel HJ, Renner T, Shestak V, Ladd J, Sutton A, Janovy D, Govindasamy S, Alqudah A, Dewri R, Prakash P (2007) Measuring the robustness of resource allocations in a stochastic dynamic environment. In: Proc international parallel and distributed processing symposium (IPDPS 2007), Mar 2007
14. Chtepen M, Claeys FHA, Dhoedt B, De Turck F, Demeester P, Vanrolleghem PA (2009) Adaptive task checkpointing and replication: toward efficient fault-tolerant grids. IEEE Trans Parallel Distrib Syst 20(2):180–190
15. Ali S, Kim J-K, Siegel HJ, Maciejewski AA (2008) Static heuristics for robust resource allocation of continuously executing applications. J Parallel Distrib Comput 68(8):1070–1080
16. Sugavanam P, Siegel HJ, Maciejewski AA, Oltikar M, Mehta A, Pichel R, Horiuchi A, Shestak V, Al-Otaibi M, Krishnamurthy Y, Ali S, Zhang J, Aydin M, Lee P, Guru K, Raskey M, Pippin A (2007) Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. J Parallel Distrib Comput 67(4):400–416
17. Mehta AM, Smith J, Siegel HJ, Maciejewski AA, Jayaseelan A, Ye B (2007) Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. J Supercomput 42(1):33–58. Special Issue on Grid Technology
18. Shestak V, Smith J, Maciejewski AA, Siegel HJ (2008) Stochastic robustness metric and its use for static resource allocations. J Parallel Distrib Comput 68(8):1157–1173
19. Deb K, Gupta H (2006) Introducing robustness in multi-objective optimization. Evol Comput 14(4):463–494
20. Qin X, Jiang H (2005) A dynamic and reliability driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. J Parallel Distrib Comput 65(8):885–900
21. Dongarra J, Jeannot E, Saule E, Shi Z (2007) Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proc 19th annual ACM symposium on parallel algorithms and architectures (SPAA'07), 2007, pp 280–288
22. Dogan A, Ozguner F (2002) Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):308–323

23. Benoit A, Hakem M, Robert Y (2008) Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In: Proc international parallel and distributed processing symposium (IPDPS), 2008
24. Byun E, Choi S, Baik M, Hwang C, Park C, Jung SY (2005) Scheduling scheme based on dedication rate in volunteer computing environment. In: Proc 4th international symposium on parallel and distributed computing (ISPDC), 2005, pp 234–241
25. Wu M-Y, Gajski DD (1990) Hypertool: a programming aid for message-passing systems. IEEE Trans Parallel Distrib Syst 1(3):330–343
26. Lord RE, Kowalik JS, Kumar SP (1983) Solving linear algebraic equations on an MIMD computer. J ACM 30(1):103–117
27. Cormen TH, Leiserson CE, Rivest RL (1990) Introduction to algorithms. MIT Press, Cambridge

Young Choon Lee received the Ph.D. degree in problem-centric scheduling in heterogeneous computing systems from the University of Sydney in 2008. He is currently with the Centre for Distributed and High Performance Computing, School of Information Technologies. His current research interests include scheduling strategies for heterogeneous computing systems, nature-inspired techniques, and parallel and distributed algorithms. He is a member of the IEEE and the IEEE Computer Society.

Albert Y. Zomaya is currently the *Chair Professor of High Performance Computing and Networking* in the School of Information Technologies, The University of Sydney. He is also the Director for the newly established Sydney University Centre for Distributed and High Performance Computing. Prior to joining Sydney University he was a Full Professor in the Electrical and Electronic Engineering Department at the University of Western Australia, where he also led the Parallel Computing Research Laboratory during the period 1990–2002. He is the author/co-author of seven books, more than 350 publications in technical journals and conferences, and the editor of eight books and eight conference volumes. He is currently an associate editor for 16 journals, the Founding Editor of the *Wiley Book Series on Parallel and Distributed Computing* and a Founding Co-Editor of the *Wiley Book Series on Bioinformatics*. Professor Zomaya was the Chair the *IEEE Technical Committee on Parallel Processing* (1999–2003) and currently serves on its executive committee. He also serves on the Advisory Board of the *IEEE Technical Committee on Scalable Computing* and *IEEE Systems, Man, and Cybernetics Society Technical Committee on Self-Organization and Cybernetics for Informatics*, and is a Scientific Council Member of the *Institute for Computer Sciences, Social–Informatics, and Telecommunications Engineering* (in Brussels). He received the *1997 Edgeworth David Medal* from the Royal Society of New South Wales for outstanding contributions to Australian Science. Professor Zomaya is also the recipient of the *Meritorious Service Award* (in 2000) and the *Golden Core Recognition* (in 2006), both from the *IEEE Computer Society*. He is a Chartered Engineer (CEng), a Fellow of the American Association for the Advancement of Science, the IEEE, the Institution of Electrical Engineers (UK), and a Distinguished Engineer of the ACM. His research interests are in the areas of high performance computing, parallel algorithms, mobile computing, and bioinformatics.



Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in 2001, where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC), a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. From 1976 to 2001, he was a professor at Purdue University. Professor Siegel is a Fellow of the IEEE and a Fellow of the ACM. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 370 technical papers. His research interests include robust computing systems, resource allocation in computing systems, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of seven international conferences, and Chair/Co-Chair of five workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/~hj.