Scientific Programming 18 (2010) 51–76
DOI 10.3233/SPR-2010-0299
IOS Press

Supporting fault-tolerance for time-critical events in distributed environments

Qian Zhu * and Gagan Agrawal
Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Abstract. In this paper, we consider the problem of supporting fault tolerance for *adaptive* and *time-critical* applications in heterogeneous and unreliable grid computing environments. Our goal for this class of applications is to optimize a user-specified *benefit function* while meeting the time deadline. Our first contribution in this paper is a multi-objective optimization algorithm for scheduling the application onto the most efficient and reliable resources. In this way, the processing can achieve the maximum benefit while also maximizing the *success-rate*, which is the probability of finishing execution without failures. However, for the cases where failures do occur, we have developed a *hybrid failure recovery* scheme to ensure that the application can complete within the pre-specified time interval. Our experimental results show that our scheduling algorithm can achieve better benefit when compared to several heuristics-based greedy scheduling algorithms, while still having a negligible overhead. Benefit is further improved when we apply the hybrid failure recovery scheme, and the success-rate becomes 100%.

Keywords: Fault tolerance, time-critical event, adaptive application, grid computing

1. Introduction

Grid or utility based computing models allow flexible use of resources by applications. Resource discovery and resource allocation are among the problems most widely studied in grid computing [3,21,27,29]. A key problem faced by applications executing in a grid computing environment is the inherent unreliability of the resources. As one considers a variety of commodity resources as part of a grid, resource failures could occur during the execution of an application. Thus, resource allocation in a grid environment should consider both the processing power and the reliability of the resources. However, there is only a very limited amount of work that has considered both factors [6,12,28]. Another related problem in grid environments is of *failure recovery*, i.e., continuing execution of an application when some of the resources may fail [15].

This paper considers these two problems in context of *adaptive* applications that perform *time-critical* event handling. These are the applications where a timely response to an important event is needed. One example could be real-time medical image processing, where images have to be rendered at a certain level of resolution from as many different angles as possible [24]. Another example is severe weather forecasting on the Great Lakes [5] for various meteorological information. Such time-critical and adaptive applications involve complex set of distributedly deployed processing stages in the grid computing environment and they are subject to a time limit. For such cases, there could be a user provided *benefit function*, which captures what is most desirable to compute. We have been developing a middleware to support these applications [35].

Reliability is a very important factor while considering resource allocation for such adaptive applications. Because of the strict time-limit over which a response is needed, resource failures could lead to a significant degradation in the application benefit, and/or a high risk of missing the time deadline. However, addressing this problem involves several new challenges. The overall goal in these applications is to maximize the benefit within the pre-specified time interval. More specifically, we want to first achieve a *baseline benefit* and then maximize it. For this, we want to process the adaptive application on the resources that are not only efficient but also reliable. Choosing the right trade-off between resource efficiency and reliability is extremely important. Furthermore, because the application needs

*Corresponding author: Qian Zhu, Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, USA. Tel.: +1 614 292 4634; Fax: +1 614 292 2911; E-mail: zhuq@cse.ohio-state.edu.

to meet the time deadline even when a failure occurs, we need failure recovery mechanisms that are fast, effective and have low overheads.

This paper presents the design and implementation of a fault-tolerance approach for supporting our target class of time-critical applications. We first propose a reliability-aware scheduling algorithm based on the concept of *Multi-objective Optimization* (MOO) to select the resources that are both efficient and reliable. The purpose of applying this scheduling algorithm is to minimize the possibility of resource failures during the event processing. However, in case that the failures do occur, we propose a *hybrid failure recovery scheme* that allows us to recover from failures with limited overhead. Although there exists work on using bi-objective optimization in grid and distributed computing [12,32,33], we consider correlated failures in our reliability model and we focus on *adaptive* applications with maximizing the application benefit while satisfying the time constraint. As a result, our work is quite distinct from the existing efforts.

Fault tolerance has been an active topic in grid computing, particularly with reliability-aware scheduling [1,6,19,28] and failure recovery [15,26]. With our goal of achieving a baseline value of the application specific benefit function and further maximizing it within a time interval, our fault tolerance problem is different. Furthermore, we target Directed Acyclic Graph (DAG) based applications, as compared to the bag of task applications that have been the focus for much of the existing work. Our work is also related to fault-tolerant DAG based real-time scheduling, which has been studied by several researchers [8,11,20]. Both our work and these efforts have to consider the dependency between services or tasks for resource assignment. However, the existing work on DAG scheduling with fault-tolerance focuses on minimizing the make-span of the application. In comparison, the problem we want to solve in this paper is more challenging which includes both maximum application benefit and maximum system reliability. In achieving such a goal, we consider the match between the resource heterogeneity and the resource usage pattern of individual services, as well as the reliability of the resource assignment.

The contributions we make in this paper are as the follows. (1) A scheduling algorithm for unreliable environments is presented, which is based on *Multi-objective Optimization* (MOO). We assign application components onto resources with goals of first achieving the baseline value of the benefit function and then maximizing it while satisfying the time constraints, and achieving highest success rate. (2) An effective failure recovery scheme has been designed to recover from resource failures and further improve the benefit without missing the time deadline and (3) an inference mechanism for time-critical event handling is presented to ensure that our proposed fault tolerance approach can always reach the baseline benefit with a success-rate of 100%.

The rest of the paper is organized as follows. We motivate our work by two real applications in Section 2. The system model and assumptions we make in our models are presented in Section 3. In Section 4, we describe our scheduling algorithm for unreliable environments and the hybrid failure recovery scheme. Results from experimental evaluation are reported in Section 5. We compare our work with related research efforts in Section 6 and conclude in Section 7.

2. Motivating applications

The adaptive applications we target in this work comprise a set of dependent *services*. Each of these services could have one or more *service parameters*, which can be modified within the pre-specified ranges. Examples of such parameters could be the time step that decides the temporal granularity of a model, or image resolution that decides the computation's spatial granularity. Adapting these service parameters could impact the application benefit as well as the event handling time. Our goal is to achieve the maximum benefit, as per the user-defined benefit function, while satisfying the time constraint. We have given a formal model for the adaptation process and based on this formulation, we have developed an autonomic adaptation algorithm [35].

This section describes two adaptive applications that require time-critical response to certain events. These applications can be adapted by tuning values of multiple service parameters, with the goal of achieving an baseline value of an application-specific *benefit function* and further maximizing it within the pre-specified time interval. We now describe these applications and the adaptable service parameters they have.

Volume Rendering: This application interactively creates a 2D projection of a large time-varying 3D data set (volume data) [13]. The volume data can be streaming in nature, e.g., it may be generated by a long running simulation, or captured continuously by an instrument. An example of the application is rendering tissue volumes obtained from clinical instruments in real-

time to aid a surgery. Under normal circumstances, the system invokes services for processing and outputs images to the user at a certain *frame-rate*. In cases where a *notable event* is detected in a particular portion of the image, the user may want to obtain detailed information on that area as soon as possible. For example, if an abnormality emerges in a part of the rendered tissue image, the doctor will like to do a detailed diagnosis in a timely fashion.

Time may be of essence, because of the need for altering parameters of the simulation or the positioning of the instrument. In obtaining the detailed information, there is flexibility with respect to parameters such as the *error tolerance*, the *image size* and also the *number of angles* at which the new projections are done. Details for the definition of a benefit function for this application can be found in our previous work [35,36]. We can formally define a *benefit function*, which needs to be maximized in the given amount of time and with given resources. Let the set of all possible view directions be denoted as Δ. Let N_b be the total number of data blocks in the dataset. For any given data block i, the importance value [30] and the likelihood of being visited are denoted as $I(i)$ and $L(i)$, respectively. Another set of parameters includes the spatial error (*SE*) and the temporal error (*TE*) [31]. Both of them should be close to a pre-defined level, (SE_0, TE_0).

Then, the benefit function can be stated as

$$Ben_{VR} = \sum_{\delta \in \Delta} \frac{\sum_{i=1}^{N_b} I(i) \times L(i)}{p} \mathrm{e}^{-(SE-SE_0)(TE-TE_0)}. \quad (1)$$

Intuitively, Eq. (1) implies that the user wants to view high-quality images from all possible view directions. For each view angle δ, the first factor impacting the quality of the final image is captured by the sum of contribution of each data block over the penalty of choosing non-beneficial nodes (p). We further calculate the contribution of data block i from its importance value and the likelihood of being visited. The second part is related to the image quality, involving the spatial and temporal errors, respectively. Although none of the tunable parameters *error tolerance* or *image size* is directly a variable in the benefit function, different choices of values from them would significantly impact the benefit we can obtain.

As demonstrated in our previous work [36], assigning the *Unit Image Rendering* service of this application to a processing node N_1 can adjust both the parameters *error tolerance* and *image size* to larger values, which leads to more application benefit within 20 min time limit, than executing it on node N_2. Thus, in a reliable computing environment, an effective resource allocation scheme is critical in achieving the maximum benefit within the time interval. However, failures of computing nodes and network links are inevitable in the grids. In a test, we assign services from the VolumeRendering application using the scheduling algorithm that we previously proposed [36], to the most efficient resources regardless of their reliability values. One or more resource failures occur 8 times out of 10 runs for a 20 min event processing in a moderately reliable environment. As a result, the application benefit for a failed execution could drop dramatically to 50% of the benefit from a successful run.

Great Lake Forecasting System (GLFS): This application monitors meteorological conditions of the Lake Erie for nowcasting (for the next hour) and forecasting (for the next day). Every second, data comes into the system from the sensors planted along the coastal line, or from the satellites supervising this particular coastal district. Normally, Lake Erie is divided into multiple coarse grids, each of which is assigned available resources for model calculation and prediction. In a situation where particular areas in Lake Erie encounters severe weather condition, such as a storm or continuous rain, the experts may want to predict additional factors, by executing other models. One possible goal may be managing sewage disposal in this area in view of the severe weather.

There is a strict time constraint on when the solution to these models is needed. However, while it is desirable to run the models with high spatial and temporal granularity, clearly there is some flexibility in this regard. For example, such flexibility could be the *resolution of grids* assigned to the model from a spatial view, or the *internal and external time steps* deciding the temporal granularity of model prediction. Furthermore, if computing resources available are limited, running new models in certain areas may be more critical than running those at other areas. We have defined a benefit function for this application that is explained in our earlier publications [35,36].

Thus, a we can formalize a benefit function as follows.

$$Ben_{POM} = \left(w \times R + N_w \times \frac{1}{4} R\right) \times \sum_{i=1}^{M} \frac{P(i)}{C(i)}, \quad (2)$$

$$w = \begin{cases} 1 & \text{if } \textit{water level} \text{ is predicted,} \\ 0 & \text{otherwise.} \end{cases}$$

Equation (2) specifies that the *water level* has to be predicted by the model within the time constraint since it is the most important meteorological information. R is a constant value for reward if this criterion is satisfied. It also gives credits to other outputs and the number of outputs N_w has to be maximized. Besides outputting useful results, the user also wants the resources to be allocated to the models with high priority. This is captured by getting the ratio of the model priority $P(i)$ and its cost $C(i)$. We also applied our previously proposed scheduling algorithm to the GLFS application in a testing experiment. The 1 h event handling failed 7 out of 10 runs in a moderately reliable environment and we only obtained 45% in terms of the application benefit from failed runs comparing to that of a successful run.

Therefore, we need a scheduling heuristic that considers both the efficiency value and the reliability of the resources. Furthermore, if a failure cannot be avoided during time-critical event handling, there should be an efficient failure recovery scheme which can guarantee to achieve the baseline benefit and further improve it within the time interval.

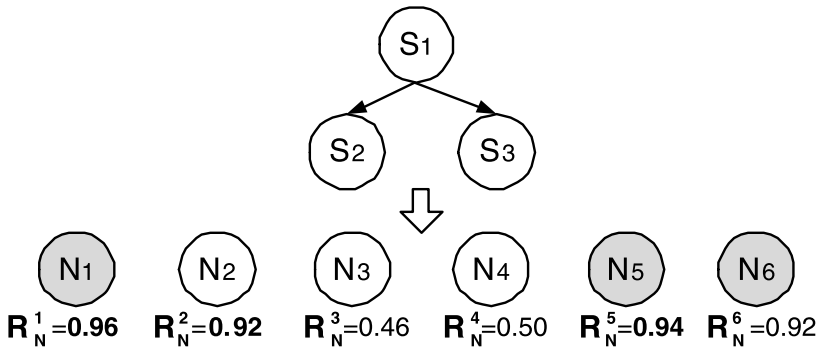
3. System model and assumptions

In this section, we present our system model and the assumptions that we use in the problem formulation. Our discussion is divided into descriptions of the application, target environment and reliability computations.

Application model: An adaptive application we target comprises a series of interacting services, which are denoted as $S_1, S_2, \ldots, S_n$. One service could be *data-dependent* and/or *control-dependent* on another service. Not considering the possibility of circular dependencies, we assume that the application has a DAG structure, as illustrated in Fig. 1. Each service is assumed to be deployed on a single node. The application initiates one *initial* service, which then initiates, directly or indirectly, all other invoked services. We assume that we have at least as many nodes available as the services, and a separate node could be used for each service.

Each time-critical event is associated with a prespecified time constraint, denoted as T_c, and a prespecified benefit function, denoted as **B**. A benefit function is a mathematical function that takes certain application parameters as input and outputs a real number. Examples of these functions were shown earlier in Eqs (1) and (2). In this work, we also propose a *baseline benefit*, which is required to be obtained from the processing. We refer to it as B_0.

Now, our goal is to achieve the baseline benefit B_0 within T_c and further maximize the value of application benefit function **B**. Recall that each service, S_i, could have one or more adaptive service parameters, which are referred to as $\mathbf{X}_{S_i}$. Such parameters could



Services	Efficiency Values					
	N1	N2	N3	N4	N5	N6
S1	0.82	0.46	**0.98**	0.55	0.72	0.53
S2	0.68	0.72	0.56	**0.96**	0.55	0.92
S3	0.54	0.56	0.71	0.68	**0.93**	0.48

Fig. 1. Running example: Application DAG, resource reliability values and efficiency values.

be tuned at runtime and their values significantly impact resource usage and execution time of service S_i, as well as the overall application benefit.

Environment model: We assume that the grid computing environment is composed of m heterogeneous computing nodes, which we denote as $N_1, N_2, \ldots, N_m$. The latency and bandwidth between these nodes are assumed to be known. Assigning a service S_i to a processing node N_j is associated with an *efficiency value* $E_{i,j}$ [36], which is calculated as follows. Primarily, it represents how efficient it is to process the service S_i on the node N_j in terms of benefit maximization. The other part considers the possibility of satisfying the time constraint T_c. $E_{i,j}$ is between 0 and 1, with 1 denoting the most efficient resource for a service. Given a set of selected resources Θ, we use $B(\Theta)$ to represent the application benefit and $T(\Theta)$ to represent the execution time. Intuitively, selecting a resource with a high $E_{i,j}$ for service S_i could help achieve the baseline B_0 and further help maximize **B** within the time constraint T_c.

However, due to the inherent unreliability of grid resources, each node N_i or network link $L_{i,j}$ is associated with a reliability value. Furthermore, we are considering the possibility that the processing node with a high efficiency value can have a low reliability value, and vice versa. The detailed reliability model will be discussed next. In our previous work, we have studied how to schedule service components based on our efficiency value definition in a perfectly reliable grid [36]. However, grid resources are often unreliable, and a resource failure could jeopardize time-critical event handling. Reliability-aware scheduling is the focus of this paper.

Reliability model: The bi-objective scheduling reported in this paper needs a model for computing reliability associated with a particular selection of resources. We describe the method used in our implementation here, though it should be noted that our scheduling algorithm is independent of the specific method for computing the reliability.

The model we use is quite general. It considers the possibility that in a large-scale, heterogeneous and complex grid computing environment, resource failures are often correlated. Specifically, there could be *temporal correlation*, implying there could be several failures occurring on multiple processing nodes over a short time interval, or a failure could appear multiple times on the same node. Similarly, we also capture *spatial correlation*, i.e., failures could occur simultaneously on multiple nodes, or a failure of a processing node may cause another failure. In contrast, most of the work in the literature simply assumes independent failures [12,16].

We assume that the probability of a resource failure could increase with system uptime and the application workload. Thus, we assume failures are Poisson processes [12]. Finally, we consider *fail-silent* (or *fail-stop*) failures on processing nodes and network links, and assume the failure can be detected in a timely manner.

Each processing node N_i (network link $L_{i,j}$) is associated with a reliability value, denoted as R_N^i ($R_L^{i,j}$). The reliability value R_N^i or $R_L^{i,j}$ is defined as the probability that the node N_i or the link $L_{i,j}$ could perform its intended function in a unit time. As stated previously, we assume this value for each resource follows a Poisson distribution dependent of time t. It has a range of [0, 1], with 1 denoting that the resource never fails.

To be able to capture temporal and spatial correlations, we use Dynamic Bayesian Networks (DBN) [22]. Since a DBN expands the basic Bayesian network with temporal dependency, we first discuss how to apply a Bayesian network to represent the spatial correlation of failures in the system reliability model. Then DBN is used to represent the temporal correlation of failures.

A *node* in the Bayesian network represents a resource, i.e., a processing node or a network link. A *link* connecting two nodes represents spatial reliability dependency relations among those resources. Figure 2(a) illustrates such a Bayesian network. The spatial correlation between resource failures are represented as solid lines. In the example, if both the nodes N_1 and N_2 fail, a failure of the link $L_{1,2}$, which joins these two nodes, is also likely. Now we consider the failures with temporal correlation in the reliability model. We expand the Bayesian network to a Dynamic Bayesian network by representing a *node* i with multiple *states*. In our model, the DBN is unrolled for two time-steps, which is referred to as a discrete-time, two-slice, temporal Bayes net (2TBN) [23]. Thus, N_1^{t-1} and N_1^t represent the two states for the node N_1, as illustrated in Fig. 2(a). The dotted line denotes the temporal correlation between resource failures. For example, a failure occurring on the node N_1 at the time-step t could lead to another failure on the link $L_{1,2}$, at the time-step t.

Given the topology of a Bayesian network and the probability distribution values at some of the nodes, the probability distribution value of some other nodes may be deduced. This is known as *inference* in Bayesian networks. We use $R(\Theta, T_c)$ to represent the probability

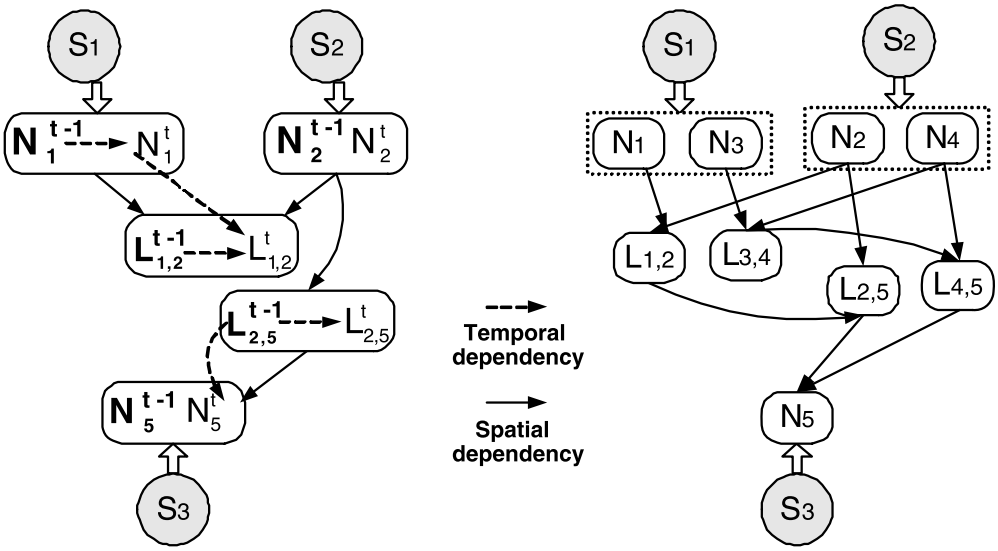


Fig. 2. Example of DBN: (a) serial; (b) parallel.

of finishing the event handling with time constraints T_c on a set of selected resources Θ, without occurring a single resource failure. The value of $R(\Theta, T_c)$ can be inferred from the Dynamic Bayesian network that we previously discussed. We refer to it as *reliability inference*. Note that $R(\Theta, T_c)$ is impacted by the reliability value of an individual resource in Θ and the time constraints T_c. We have applied an *likelihood weighting* inference algorithm [23] to estimate the value of $R(\Theta, T_c)$, based on individual resource reliability values and temporal and spatial correlations. Note that we do not assume the underlying failure distribution of the grid computing environment has to be known a priori. The method we use allows us to learn temporally and spatially correlated failures.

When assigning each service component to a single node, as demonstrated in Fig. 2(a), we refer to it as scheduling with a serial structure. We infer the value of $R(\langle N_1, N_2, N_5\rangle, 20) = P(N_5^{T_c=20}|L_{2,5}^{T_c=20}, L_{2,5}^{T_c=19})\times P(L_{2,5}^{T_c=20}|L_{2,5}^{T_c=19}) \times \cdots \times P(N_1^{T_c=20}|N_1^{T_c=19}) \times P(N_1^{T_c=19}) = 0.86$. However, in order to achieve redundancy, a service could be assigned to multiple nodes and we refer to it as scheduling with a parallel structure. Such an example is illustrated in Fig. 2(b). Note that we omit the temporal correlation between nodes for the simplicity of presentation. We schedule two copies of S_1 to nodes N_1 and N_3 and two copies of S_2 to nodes N_2 and N_4 in the example. The value of $R(\langle N_1, N_2, N_3, N_4, N_5\rangle, 20) = 1-(1-P(N_5^{T_c=20}|L_{2,5}^{T_c=20}, L_{4,5}^{T_c=20}, \ldots)\times\cdots\times(1-P(N_1^{T_c=20}|N_1^{T_c=19}) = 0.96$.

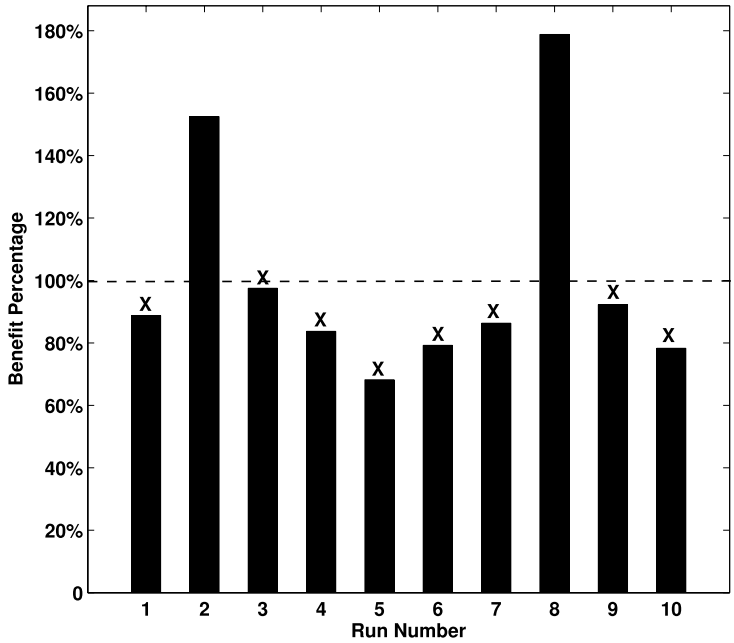
4. Fault-tolerance approach

This section presents the solution we propose to support fault-tolerance in handling time-critical events. We first present two initial solutions. Then, the fault-tolerance problem is formally formulated, and based on this formulation, we present our scheduling algorithm for unreliable resources. Finally, an efficient failure recovery scheme is proposed which cooperates with the scheduling algorithm and targets faults occurring during the event processing.

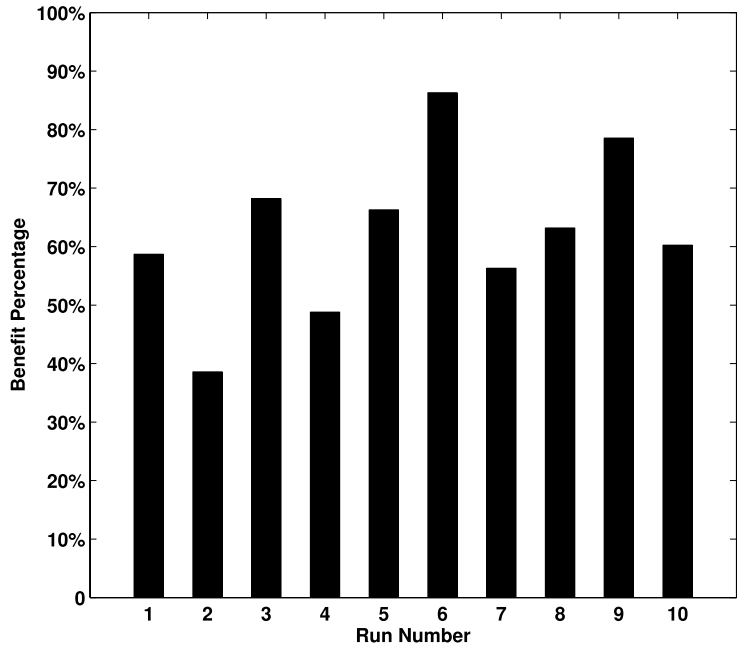
Initial solutions: As straightforward solutions to achieve our goal, we consider the following two heuristics. First, we could schedule the application onto the most efficient resources, i.e., perform *efficiency-value based scheduling*. Alternatively, we could select the most reliable resources, i.e., perform *reliability-value based scheduling*. Both these methods proceed in a greedy way. However, it turns out that neither of these solutions is adequate for our needs. In an experiment using these two scheduling heuristics, we trigger a 20 min event for 10 times for the VolumeRendering application. For example, as illustrated in Fig. 1, the scheduling heuristic based on the efficiency value would assign S_1, S_2 and S_3 to

$\Theta_1 = \langle N_3, N_4, N_5 \rangle$, with $R(\Theta_1, 20) = 0.28$ and $B(\Theta_1)/B_0 = 178\%$. While the scheduling heuristic based on the reliability value would assign those services to $\Theta_2 = \langle N_1, N_2, N_5 \rangle$, with $R(\Theta_2, 20) = 0.85$ and $B(\Theta_2)/B_0 = 72\%$.

We demonstrate the obtained benefit percentage from these two approaches in Fig. 3. The event processing stops if there is a resource failure and the current benefit is taken as the final application benefit. Note that the benefit percentage is $B(\Theta)/B_0$ and



(a)



(b)

Fig. 3. Benefit percentage of VolumeRendering application with different scheduling heuristics: (a) efficiency value; (b) reliability.

failed runs are marked with X in both figures. We can observe that when scheduling only based on the efficiency value, the application can achieve up to 180% of the baseline benefit, if there are no resource failures during the processing. However, there are only two successful runs from among the ten runs, and the benefit from the failed ones can drop to 68% of the baseline benefit, which is roughly $\frac{1}{3}$ of the benefit from the successful runs. By scheduling based on the reliability value, sub-figure (b) illustrates that although 9 out of 10 runs are successful, the average benefit percentage is only 70%.

The results from this experiment demonstrate that there are usually conflicting requirements between maximizing the benefit function and the reliability of the application processing, and it may not be possible to simultaneously optimize both. However, the scheduling algorithm should be capable of balancing the application benefit value and reliability maximization. We propose such a solution in this paper.

4.1. Problem formulation

To present our solution, we first need to present a formal problem description. In order to achieve the pre-defined baseline benefit B_0 with maximum further improvement, while satisfying the time constraint T_c, we have to select a set of resources Θ in a unreliable computing environment for the processing. That is, both the application benefit $B(\Theta)$ and reliability $R(\Theta, T_c)$ from the selected resources should be maximized. We apply the concept of *Multi-objective Optimization* (MOO) to formulate this problem. It should be emphasized that the MOO based scheduling problem with goals including both reliability and benefit maximization, while meeting the time constraint is new in the literature. Particularly, there is no direct tradeoff between reliability and application benefit. Therefore, both the resource reliability and efficiency values could affect the application benefit.

$$\max[B(\Theta), R(\Theta, T_c)] \tag{3}$$

satisfying the following inequality constraint

$$B(\Theta) \geqslant B_0 \tag{4}$$

and the following equality constraint

$$T(\Theta) = T_c. \tag{5}$$

In the case of MOO, two different solutions cannot always be directly compared to each other. In the running example, as we previously discussed, by assigning services S_1, S_2 and S_3 to Θ_1, we have $[B(\Theta_1)/B_0 = 178\%, R(\Theta_1, 20) = 0.28]$. While with the selected resources in Θ_2, we have $[B(\Theta_2)/B_0 = 72\%, R(\Theta_2, 20) = 0.85]$. We cannot say Θ_1 is a better resource configuration than Θ_2 or vice versa. Thus, we use the concept of *domination* in order to compare two resource plans in the context of our optimization problem. A resource plan Θ_1 dominates another resource plan Θ_2, if and only if Θ_1 is partially larger than $\Theta_2 (\Theta_1 >_p \Theta_2)$

$$B(\Theta_1) \geqslant B(\Theta_2) \wedge R(\Theta_1, T_c) \geqslant R(\Theta_2, T_c) \tag{6}$$

and

$$B(\Theta_1) > B(\Theta_2) \vee R(\Theta_1, T_c) > R(\Theta_2, T_c). \tag{7}$$

In the absence of any preference information, a set of solutions for Θ is obtained, where each solution is equally significant. This is because in the obtained set of solutions, no solution is dominated by any other solution. Such a set of solutions is referred to as the *Pareto-optimal* (*PO*) *set* [10]. Usually, we need to choose a single solution from the Pareto set, as required for the implementation. We define the following objective function as weighted sum of benefit and reliability with a trade-off factor α.

$$\max_{\Theta \in PO} \alpha \times (B(\Theta)/B_0) + (1 - \alpha) \times R(\Theta, T_c). \tag{8}$$

The trade-off factor, α, can be tuned to best fit to the characteristics of the computing environment. We use the Eq. (8) interactively during the search process to find the best candidate from the Pareto-optimal set. The detailed algorithm is presented in the following subsection.

4.2. Scheduling algorithm for unreliable resources

In this subsection, we first present our scheduling algorithm for unreliable resources which has a serial scheduling structure. Then, we discuss scheduling with redundancy and failure recovery, which is based on the parallel structure. We argue that our proposed scheduling algorithm is independent of the reliability model that is used.

The determination of a complete Pareto-optimal set is a very difficult task, due to the computational complexity caused by the presence of a large number of suboptimal Pareto sets. There has been a tremendous amount of work on Multi-objective Optimization with the goal of finding the Pareto-optimal set [10]. In

this paper, we adopt a recently proposed *metaheuristic* called *Particle-swarm Optimization* (PSO) [18] as the search mechanism. The reason is that the algorithm has a high speed of convergence and it allows us to iteratively interact with the single objective function, which we defined in Eq. (8), to find the best solution from the approximate Pareto-optimal set [2]. Furthermore, it is easy to balance the scheduling time and the quality of the resource plan generated by the algorithm by adjusting the convergence criteria.

One of the issues is choosing the appropriate value for the parameter α. This parameter decides the weight of the benefit in the overall objective. As we stated previously, the choice of this value should depend on the characteristics of the underlying environment. We now briefly describe a heuristic to automatically choose the value of α.

The heuristic has two main steps. In the first step, we decide if the environment could be considered *reliable* or not. If yes, the value of α we choose is higher than 0.5, since less weight needs to be given to reliability. If not, the value of α we choose is less than 0.5. In the second step, we further refine the value of α. To enable these steps, we generate two sets of initial resource configurations using greedy scheduling, with the efficiency value and reliability value as the criteria for each. These two sets are denoted as Θ_E and Θ_R, respectively. For both the sets, we calculate the mean of the reliability values. If the difference between the mean reliability of the two sets is less than a threshold, we conclude that the environment is reliable. In our implementation, we used 0.1 as the threshold. Otherwise, we conclude that the environment is unreliable. The reason is that in a reliable environment, greedy scheduling based only on efficiency will still lead to high reliability.

The next step refines the value of α. If the environment is considered reliable, we increase the value of α, starting from 0.5. After each increment, we calculate the objective function value based on Eq. (8), for each configuration in the set Θ_R. The goal is to see how we can maximize the benefit, within the set of configurations that maximize reliability. This procedure stops when there is no further increase in the value of the objective function. If the environment is considered unreliable, we decrease the value of α, starting from 0.5, and work with the configurations in the set Θ_E.

We next present the scheduling algorithm in Fig. 4. A resource configuration is referred to as a *particle* in the algorithm description. The *position* of the particle is defined as the objective function value calculated from the Eq. (8). The *velocity* of the particle is

Algorithm VII.1: (RELIABILITYAWARESCHEDULING(obj, S, T_c))

INPUT *obj*: the objective function
S: set of service components
T_c: Time constraint for the event
OUTPUT $\hat{\Theta}$: the optimal resource plan

```
CalculateEfficiencyValue(S);
P = InitializeParticles(S);   // initialization of particles
while (true)
  for each Θ_i ∈ P
    //calculate B(Θ) and R(Θ, T_c)
    p_i = CalculateObj(obj, Θ_i);
    if p_i > pBest_i
      pBest_i = p_i;   //update local optima
  gBest = max_{Θ_i ∈ P}(pBest_i);   //update the global optima
  for each Θ_i ∈ P
    //update resource assignment
    v_i = v_i + C_1 × r_1 × (pBest_i − p_i)
          + C_2 × r_2 × (gBest − p_i);
    Θ_i = Θ_i + v_i;
  if (convergence criteria is met)
    break;
```

Fig. 4. Fault tolerant scheduling algorithm.

defined as change to the current resource configuration by assigning one of the service components to anther node. The algorithm begins with calculating the efficiency values, and proceeds with searching for optima by updating generations. In every iteration, we first update each resource configuration by following two *best* values. One is the optimal objective function value achieved by a particular particle, which is referred to as *pBest* in the algorithm. The other is the optimal objective value achieved by any particle and we denote it as *gBest*. After both of them are updated, we try to explore more resource configurations by changing resource assignments based on the two formulas we present in Fig. 4. Note that r_1 and r_2 are random numbers between 0 and 1 and $c_1 = c_2 = 2$, which are the learning factors in our approach. This iterative procedure stops when there is no significant gain with regard to either benefit or reliability. We return the particle with the *gBest* as the optimal resource configuration.

Example: Applying the proposed scheduling algorithm to the example in Fig. 1, we decide to choose the resource plan $\Theta_3 = \langle N_1, N_6, N_5 \rangle$, which leads to $[B(\Theta_3)/B_0 = 186\%, R(\Theta_3, 20) = 0.85]$. It is obvious that Θ_3 dominates both Θ_1 and Θ_2, which we had obtained earlier from our two simple heuristics. In other words, the resource configuration generated by our proposed scheduling algorithm can achieve better benefit and success-rate. The reason our MOO based scheduling algorithm outperforms the two initial solutions is as the following. Recall that the efficiency-based heuristic selects the nodes with the highest efficiency values. In comparison, our proposed algorithm is able to select the resources with efficiency values that are very close to the highest possible, while achieving much higher reliability. That is why we choose node N_1 over N_3 in resource configuration Θ_3 for service S_1 assignment. Although $E_{1,3} = 0.96$ is larger than $E_{1,1} = 0.82$, node N_1 is more reliable than N_3 (0.96 *vs.* 0.46). Similar reason could apply to the reliability-based heuristic. We argue that the real grid computing environments can have resources that vary in capacity, and/or reliability based on either heterogeneity or workloads/usage. Our MOO based algorithm will not apply to the case where there are either highly efficient resources with very low reliability values, or there are highly reliable resources but very inefficient.

We repeated the experiment using our proposed scheduling algorithm and observed that due to the unreliability of resources, there are still 2 failures in the 10 runs. The benefit percentage for the failed runs is 86% on average. We discuss how to successfully complete the event handling when the resource failures do occur in the next subsection.

4.3. Discussion

We now discuss the following three characteristics of the algorithm. First, the value of the parameter α in Eq. (8) is automatically chosen from the algorithm we proposed and it is not based on any priori knowledge of the reliability distribution of the underlying environment. In a highly reliable environment, this value should be close to 1, in order to favor the application benefit. Whereas, if most resources are unreliable, α should be closer to 0, to favor the reliability. Second, our scheduling algorithm generates resource mappings where the pre-specified baseline benefit is guaranteed to be achieved within the time interval, provided the run is successful, i.e., no failures occur. This is achieved by inferring the benefit that could be obtained from a resource plan based on the efficiency value of the individual nodes. We refer to it as *benefit inference*. Finally, we have to balance the scheduling overhead and actual processing time so that the time-critical event can be successfully processed within the pre-specified time constraint, even in presence of failures. This is referred to as *time inference*. We now present *benefit inference* and *time inference* in detail.

Benefit inference: Recall that every time-critical event is associated with a baseline benefit, which is required to be achieved for event processing during the specified time constraint. In order to guarantee such a baseline benefit, our proposed scheduling algorithm should be able to estimate the benefit that could be obtained given the resource configuration generated from our algorithm. Particularly, the value of each adaptive service parameter within a certain amount of time is first estimated. Then, based on the learned relationship ($f_B(x)$) between these parameters and the application benefit [35], we can infer the benefit that could be obtained from the selected resources. Specifically, the value of each adaptive service parameter is estimated as follows. For each of the services, we collect multiple tuples, where each tuple is of the format $d_m = \langle E_m, t_m, x_m \rangle$. E_m is the efficiency value associated with executing this service on a particular node and is formally defined in [36]; t_m is the execution time and x_m are the values which the adaptive service parameters converge to at the end of execution. Now we regress the relationship between the efficiency value

and the values of the parameters. Such relationship is denoted as $f_P(E, t)$. Note that we consider the execution time in this relationship for two reasons. First, the efficiency value is dependent on the time constraint. Thus, same value of the efficiency value with various time constraints will cause the adaptive service parameters to converge to different values. Second, in the case where the service is hosted on the same processing node with different time constraints, we will have the parameters converging to different values. Therefore, given a time-critical event with the baseline benefit B_0 and the time constraint T_c, the application benefit can be estimated based on the resource configuration generated from our scheduling algorithm as the following (assume service S_i is hosted on node N_j):

$$\begin{aligned} &B_{est} = f_B(X_{S_i}), \quad \text{where } X_{S_i} \in S_i \quad \text{and} \\ &X_{S_i} = f_P^i(E_{i,j}, T_c) \quad \text{for } i = 1, 2, \ldots, n. \end{aligned} \tag{9}$$

The resource configuration will be discarded if $B_{est} < B_0$. As we demonstrate in the experimental evaluation, the benefit inference is accurate and our scheduling algorithm can always generate the resource configuration to achieve the required baseline benefit.

Time inference: The time constraint associated with an event is divided into two parts, i.e. the scheduling overhead, denoted as t_s, and the actual event processing time, denoted as t_p. Intuitively, a longer scheduling time could generate a better resource configuration. However, we need to make sure the benefit we gain is worth the price we pay for increasing the scheduling overhead, and thus reducing the actual time available for processing. Furthermore, more time for failure recovery will be reserved if the estimated system reliability value is small. Effectively distributing the available event processing time T_c to t_s and t_p is quite challenging. We propose the following heuristic to solve the problem.

Recall that our reliability-aware scheduling algorithm stops when there is no significant gain with regard to either benefit or reliability, as illustrated in Fig. 4. Such *convergence criteria* can be varied to trade-off the scheduling time and *quality* of generated resource configuration. A small value of the convergence criteria could be significantly compute-intensive, thus causing a large scheduling overhead. Meanwhile, the solution would also be close to the optimal to the MOO problem, i.e. the adaptive application executed on such selected resources will lead to a large value of benefit with a high probability of avoiding resource failures during processing. On the other hand, we can reduce the scheduling overhead by setting the convergence criteria for the algorithm to a large value. However, the generated resources could be less efficient and/or unreliable. Therefore, during the training phase, we vary the convergence threshold and record the corresponding scheduling time and application benefit obtained from the generated resource configurations. Note that we have a fixed set of candidate values for the convergence criteria and each of them is associated with a significantly different level of application benefit that can be achieved.

Next, we need to take into account the failure recovery time if a resource failure occurs during the event processing. In this work, we consider the failures on processing nodes and network links. During the experiments, we observe that the time to recover from a node/link failure is consistent. Thus, we believe the average statistics will be a good estimate for such recovery time. Specifically, T_r refers to the estimated time to recover a network link or a processing node failure. Finally, we relate the reliability value of the selected resources with the number of failures that occurs during the event processing. Such relationship is denoted as $f_R(r)$, where $r = R(\Theta, T_c)$ and it is the reliability value from the generated resource configuration Θ given time constraint T_c. Based on the learned relationship ($f_T(x)$) between adaptive service parameters and the application execution time [35] and Eq. (9) to estimate parameter values, we distribute T_c to scheduling overhead and actual processing time as follows. For each candidate convergence criteria, we assign t_s to the recorded scheduling time. Then $t_p = T_c - t_s$ and it has to satisfy the following constraint:

$$\begin{aligned} &t_p > f_T(X_{S_i}) + m \times T_r, \\ &\quad \text{where } m = f_R(r) \text{ and } i = 1, 2, \ldots, n. \end{aligned} \tag{10}$$

Then the candidate with the largest benefit while meeting the constraint in Eq. (10) will be taken.

Together, reliability, benefit and time inferences facilitate a synergy of the failure recovery scheme with our reliability-aware scheduling algorithm. In the future, we will work on how to automatically tradeoff the scheduling overhead and the quality of the generated resource configuration.

4.4. Failure recovery scheme

Although our scheduling algorithm chooses resources so as to reduce the possibility of resource

failures during the processing, it is still possible that a failure may occur and interrupt the processing. Thus, an effective failure recovery scheme is necessary.

A simple solution could be based on replication. We can create multiple copies of the entire application and schedule all of the copies. To evaluate how this approach will work, we trigger an event requiring a 20 min response, using the VolumeRendering application. We use 4 copies of all services associated with the application. The results from repeating this experiment 10 times are shown in Fig. 5. As we can see, all 10 runs are successful. However, the obtained benefit percentage is an average of 96%. This is due to the significant overhead of maintaining and switching between multiple copies.

We propose a *hybrid failure recovery scheme* as an enhancement to our proposed scheduling algorithm. A key observation we use is that services that are not the initial service, as often invoked repeatedly by their parent(s) in the DAG, there is often only a very small amount of state that needs to be preserved between these invocations. Such a small amount of state can be easily checkpointed. Such checkpoints are first updated locally, and then they are transferred to a reliable node for storage and retrieval. For services where such low-cost checkpointing is not possible, we have to schedule multiple copies. In our implementation, we use checkpointing for the service where the size of its state is less than 3% of the memory consumed by the service. Furthermore, all copies start processing when the service is invoked by another service, and the copy that finishes processing first will be considered as the primary. In the running example, both service S_1 and S_2 are replicated with two copies each, while S_3 will be checkpointed during its execution. We set the reliability value of the service with checkpointing as 0.95.

Such a combination of checkpointing and replication of services can allow failure recovery. Furthermore, the point at which the failure occurs impacts the decision of choosing the recovery strategy, as suggested by the reference mechanism. We consider the following three cases.

Close-to-start: If the failure occurs shortly after the processing begins, we simply ignore what has been done up to the failure point. This is because it is very likely that the processing performed so far is not very useful, and the loss of such a short period of time will not significantly impact the benefit achieved.

Middle-of-processing: We want to resume from the failure point since useful processing has been done. We can resume the services using the stored checkpoints or switch to another copy, depending upon the service. The failure recovery may cause overhead, however, re-

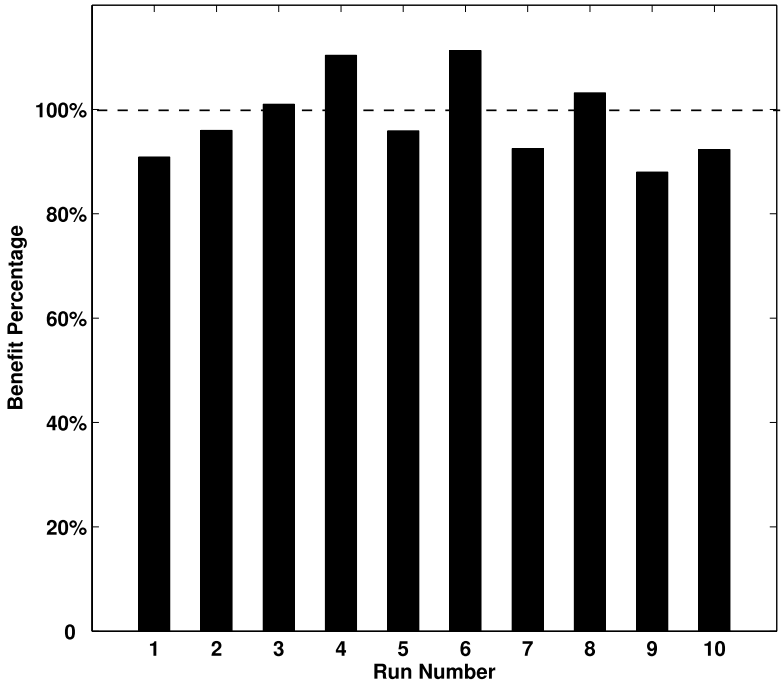


Fig. 5. Benefit percentage of VolumeRendering: Multiple application copies.

trieving the past processing information helps us improve the achieved benefit.

Close-to-end: If the failure occurs very close to the end of time-period for processing, we simply stop processing at this point. This is because any recovery may not help improve the benefit any further.

5. Experimental evaluation

This section presents results from a number of experiments we conducted to evaluate our reliability-aware scheduling algorithm and the hybrid failure recovery scheme.

5.1. Algorithms compared, metrics and goals

For comparison, we used three alternative scheduling heuristics. Greedy-E chooses the resource assignment only based on the efficiency value we discussed in Section 4. The Greedy-R chooses the resources only based on their reliability values. The third heuristic takes both the efficiency value and the reliability value into account and resources are ranked by the product of these two values. We refer to it as Greedy-$E \times R$.

In order to evaluate our proposed failure recovery scheme, we compared it with a simple redundancy based approach where r copies of the entire applications are scheduled and each copy uses a different adaptation strategy. We vary the value of parameter r from 2 to 5 for different grid environments. The highest benefit obtained from copies that can complete successfully within the time interval is taken as the result. We refer to this approach as With Application Redundancy.

To evaluate the performance of our approach against Greedy-E, Greedy-R and Greedy-$E \times R$, we use the following two metrics: (1) *Benefit Percentage*: This shows the benefit obtained from the scheduling algorithm, as a percentage of the pre-defined baseline benefit. Recall that the baseline benefit is specified as being a requirement from the user. (2) *Success-Rate*: The success-rate is defined as the percentage of the time-critical events successfully handled within the time interval. While the goal of each scheduling algorithm is to have a 100% success-rate, it does not always happen because of the limitations of the scheduling algorithm and resource failures that occur during the event processing.

Using the three scheduling heuristics and the above two metrics, we designed the experiments with the following goals: (1) Demonstrate that our proposed reliability-aware scheduling algorithm can generate an efficient resource configuration, which leads to high benefit and success rate, (2) the scheduling overhead of our algorithm is negligible, and the algorithm is scalable, (3) demonstrate that when there are one or more failures during the event handling, our proposed hybrid failure recovery scheme can improve the benefit over what is achieved by our scheduling algorithm.

5.2. Experimental setup

We emulated grid environments using two Linux clusters, each of which consists of 64 computing nodes. One cluster has dual Opteron 250 (2.4 GHz) processors and the other has dual Opteron 254 (2.4 GHz). Each processing node has 8 GB of main memory and 500 GB local disk space, and is interconnected with switched 1 Gb/s Ethernet. The two clusters are located in different buildings, about 0.5 miles apart, within the Ohio State university campus and are connected using two 10 Gb/s optical fibers. We emulated two Grid sites with each representing a 64 node cluster. The emulator we used is GridSim [7] with *time-shared round robin* scheduling for each processor.

In order to emulate various real-world resources in computational grids, we emulated the following three grid environments in terms of resource reliability:

- *HighReliability*: A highly reliable environment where most of the resources do not fail during the application processing. This was emulated using the complement of a normal distribution ($\mu = 1$, $\delta = 0.05$).
- *LowReliability*: A highly unreliable environment where most of the resources fail frequently during the application processing. This was emulated using the a heavy-tailed distribution (1-Pareto(a, b) with parameters $a = 1$, $b = 0.2$).
- *ModReliability*: A moderately reliable environment with a mix of reliable and unreliable resources. This was emulated using a uniform distribution with mean of 0.5.

Furthermore, in order to emulate the temporally and spatially correlated failures, we have applied the temporal/spatial correlation model of failures studied in high performance computing systems from [14]. Most of our experiments were performed in a highly heterogeneous environment where, the processor architecture, CPU speed, memory size and network bandwidth

Table 1
Details of the VolumeRendering and GLFS applications

Application	Services		Dataset
	Preprocessing	Rendering	
VolumeRendering	WSTP tree construction service	Unit image rendering service	7.5 GB
	Temporal tree construction service	Decompression service	(30 time steps)
	Compression service	Image composition service	
GLFS	POM model service (2-D mode)	POM model service (3-D mode)	21.0 GB
	Grid resolution service	Linear interpolation service	

vary significantly. We followed the resource models studied in [17].

The experiments we report were conducted using two applications, i.e., VolumeRendering and GLFS, which were described previously in Section 2. The details of both applications are listed in Table 1. There are three adjustable service parameters in the VolumeRendering application. The *wavelet coefficient*, denoted as ω, is from the Compression Service. Both *error tolerance* and *image size*, denoted as τ and ϕ, are parameters in the Unit Image Processing Service. The benefit function is defined using Eq. (1). We observed that a smaller value of τ yields more benefit from Ben_{VR}. The correlation between ϕ and Ben_{VR} is positive. Furthermore, τ impacts Ben_{VR} more significantly than ϕ does. For this VolumeRendering application, if one or more service components encounter resource failures during the time-critical event handling, the value of the adaptive service parameters, i.e. *error tolerance* or *image size*, should be retrieved and the communication with its invoker or invokee services should be recovered. Otherwise, images would not be rendered with the required resolutions or we will miss certain display angles. The second application is referred to as GLFS. The tunable parameters in this application are the number of internal time steps (T_i), number of external time steps (T_e) and grid resolution (θ). They are from POM Model Services and the Grid Resolution Service, respectively. The benefit function for GLFS application is defined in Eq. (2). Through the experiments, we observed that there is a relationship between Ben_{POM} and T_i and T_e. Furthermore, the correlation is negative for T_e and positive for T_i. Similarly, the GLFS application should retain the value of its adaptive parameters, i.e. the *grid resolution*, *internal* and *external* time steps in the presence of resource failures. They are used to predict the most important meteorological information. Therefore, occurrence of failures can cause degradation of application benefit or delay for an adaptive application.

5.3. Performance of scheduling algorithm

In this subsection, we evaluate the performance of our approach against Greedy-E, Greedy-R and Greedy-$E \times R$ with respect to the two metrics, i.e., benefit percentage and success-rate. We also compare the overhead of different approaches.

Benefit percentage comparison: We now compare our proposed scheduling algorithm against the three heuristics. In this experiment, we demonstrate that our approach can assign service components to resources that are both efficient and reliable. Thus, we can always achieve the baseline benefit as well as a high success-rate.

First, we show how the VolumeRendering application could benefit from our reliability-aware scheduling algorithm in terms of minimizing the possibility of resource failures for benefit maximization. During the processing, we invoked multiple time-critical events, with the time constraints being 5, 10, 15, 20, 25, 30, 35 and 40 min, respectively. For each event, we executed 10 runs and report the average value. Note that if a resource fails, we stop the processing and take the current benefit as the final result. We considered the application benefit as defined in Eq. (1). The benefit percentage, which is the ratio of obtained benefit over the baseline benefit, is shown in Fig. 6. The three sub-figures consider cases with different levels of reliability. We make the following observations from the results. First, our approach can always achieve the pre-defined baseline benefit. This demonstrates that the scheduling algorithm can minimize the possibility of failures without sacrificing the application benefit. Note that there could be up to 2 out of 10 runs which fail to achieve the baseline benefit. Later we show how our proposed hybrid failure recovery scheme will help achieve the goal even in presence of failures. Second, the benefit percentage is further improved up to 206%, 168% and 110% in highly reliable, moderately reliable

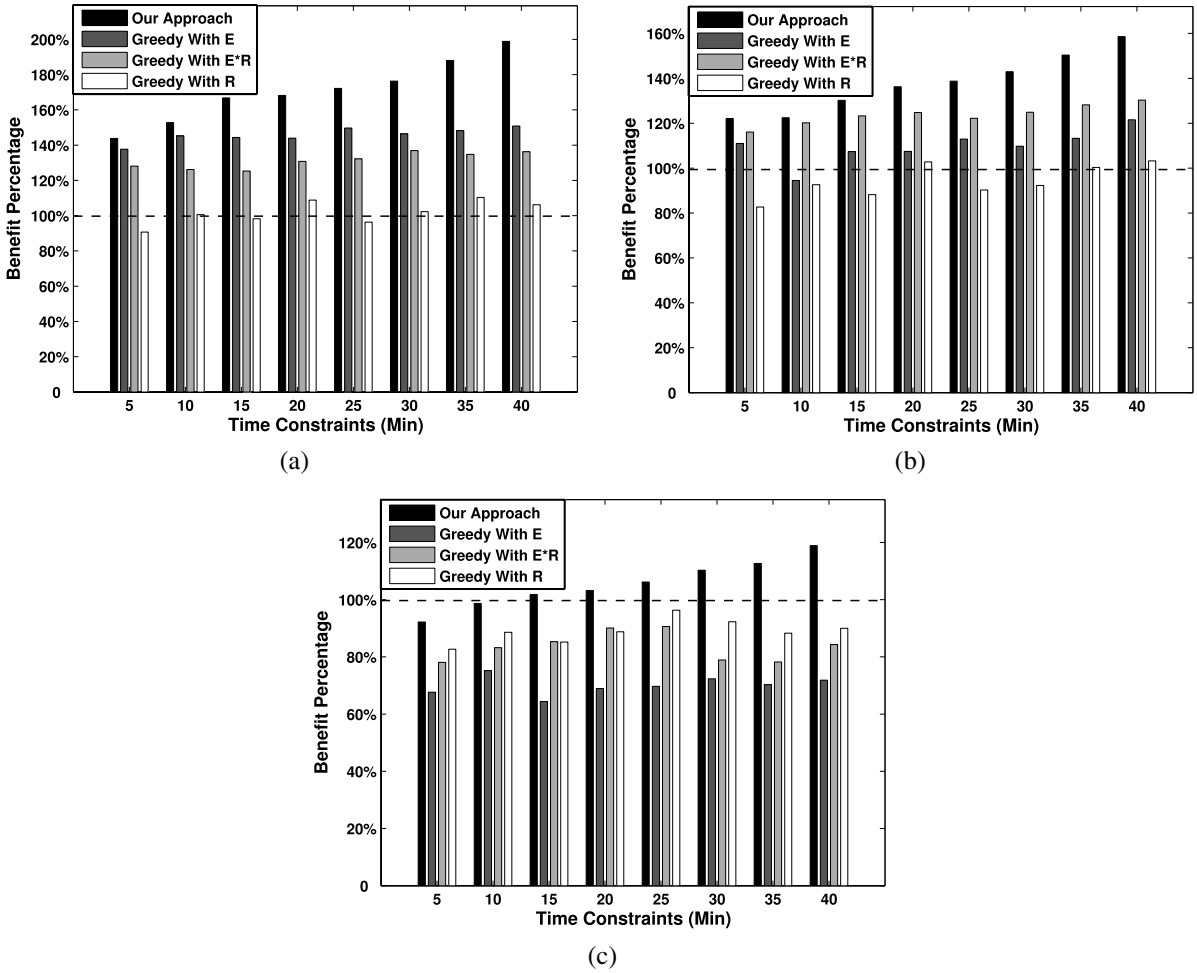


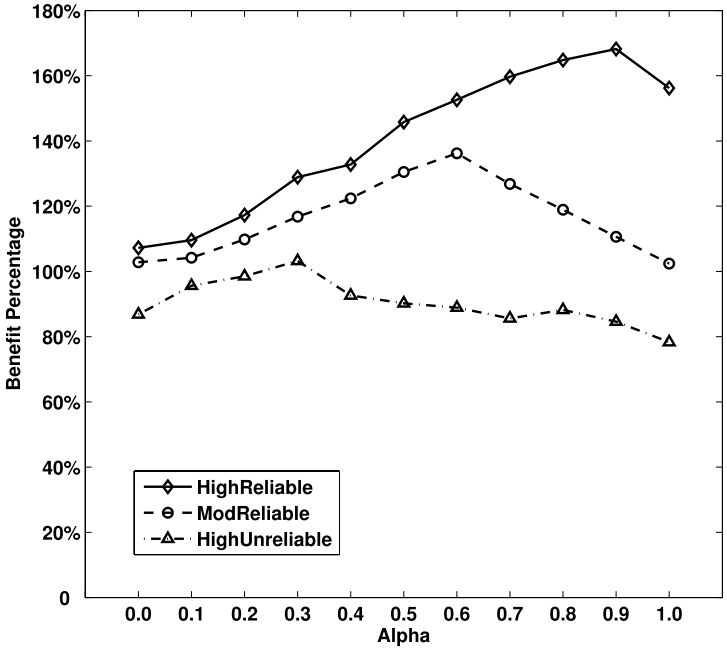
Fig. 6. Benefit percentage comparison of our approach with three scheduling heuristics: VolumeRendering – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

and highly unreliable environments, respectively. We also observed that the obtained benefit decreases in less reliable environments because of a higher possibility of resource failures.

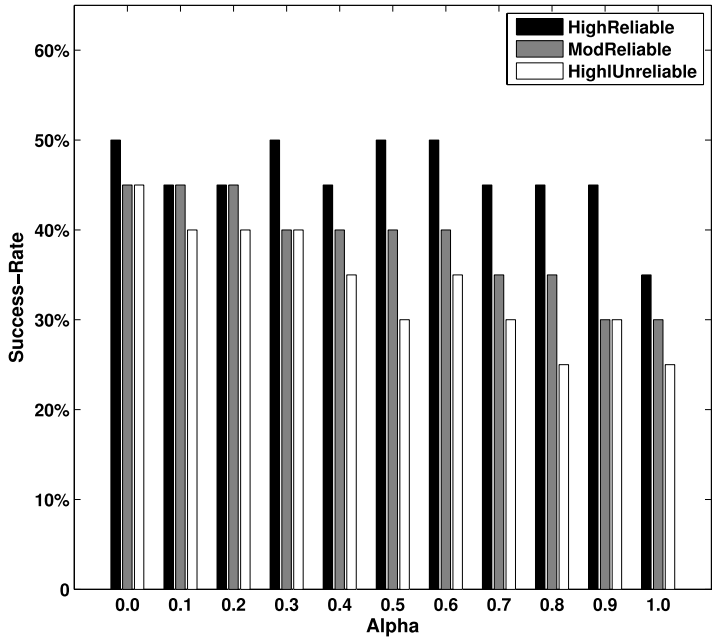
We believe the further improvement over the baseline benefit is achieved because that our scheduling algorithm can interactively adjust the value of the parameter α in Eq. (8). Thus, it can favor nodes with high efficiency values in a highly reliable environment, or focus more on the reliability values when most nodes are unreliable. To further illustrate this issue, the obtained benefit and success rates of a 20 min event are shown as a function of α in Fig. 7(a) and (b). The maximized benefit with a 90% success rate is achieved by tuning the parameter α to be 0.9 in the highly reliable environment. Whereas, $\alpha = 0.3$ yields the best benefit with a success rate of 100% in the highly unreliable environment. In the moderately reliable environment, the benefit peaks when α is 0.6. The values of α we obtained for these three cases were automatically decided by the method we had described earlier.

Finally, an observation can be made that the benefit gain increases as the time constraints associated with the events increase. The reason is that we can use a higher scheduling time to select resources for application performance optimization.

In comparison, the benefit percentage from Greedy-E could achieve 182% and 106% in highly and moderately reliable environments, respectively, but only achieves 62% in the highly unreliable environment. Similar observations can be made for the Greedy-$E \times R$ heuristic. The reason is that when the reliability is not a concern, considering only the effi-



(a)



(b)

Fig. 7. Varying the value of the parameter α: VolumeRendering – (a) benefit percentage; (b) success rate.

ciency value for scheduling can achieve good performance. However, with unreliable resources, the ratio drops dramatically. Even with Greedy-$E \times R$, where both the efficiency value and the reliability value are considered, simply taking the product does not generate a resource assignment with smaller probability of failures. We argue that our proposed scheduling algorithm outperforms the Greedy-$E \times R$ heuristic because

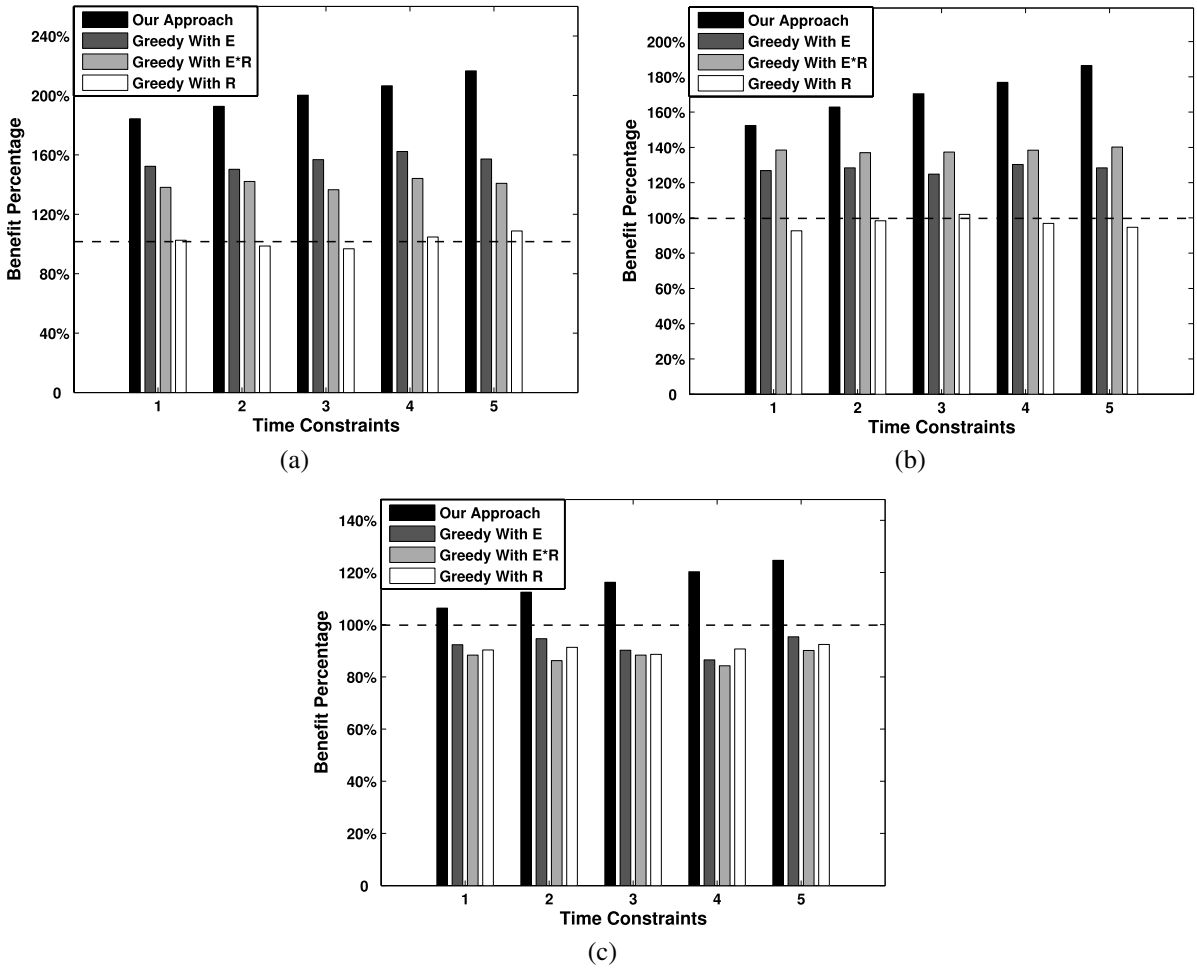


Fig. 8. Benefit percentage comparison of our approach with three scheduling heuristics: GLFS – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

the latter chooses a node with a large value of the product of the efficiency and reliability values. However, this could lead to a problem in the case when the node is relatively inefficient yet with a high reliability value (to make the product value large), and environment is reliable. In such a case, the efficiency value should be favored with regard to benefit maximization, which is what our approach does. Note that the Greedy-$E \times R$ heuristic achieves 18% less benefit compared to our approach in the moderately unreliable case. The third heuristic we compared with, which is Greedy-R, did not reach the baseline benefit in three computing environments. This is due to the fact that some reliable resources could be very inefficient and focusing only on the reliability could degrade the application benefit significantly.

The benefit experiment was repeated using the GLFS application. We invoked time-critical events with 1, 2, 3, 4 and 5 h as the time constraints. Results are demonstrated in Fig. 8(a)–(c), for highly reliable, moderately reliable and highly unreliable environments, respectively. Similar observations can be made for this application. The benefit percentage from our scheduling algorithm is up to 220%, 172% and 117% in the three environments. Whereas, Greedy-E could achieve 176%, 128% and 87% on average and Greedy-$E \times R$ achieves 143%, 158% and 91%. Similarly, Greedy-R can hardly reach the baseline benefit.

Success-rate comparison: Next we compared the performance of the four scheduling algorithms in terms of the success-rate. We first carried out the experiment using the VolumeRendering application. As illustrated in Fig. 9(a), in a highly reliable environment, we can achieve 90–100% from our algorithm. In comparison, the success-rate for Greedy-E and Greedy-$E \times R$ is

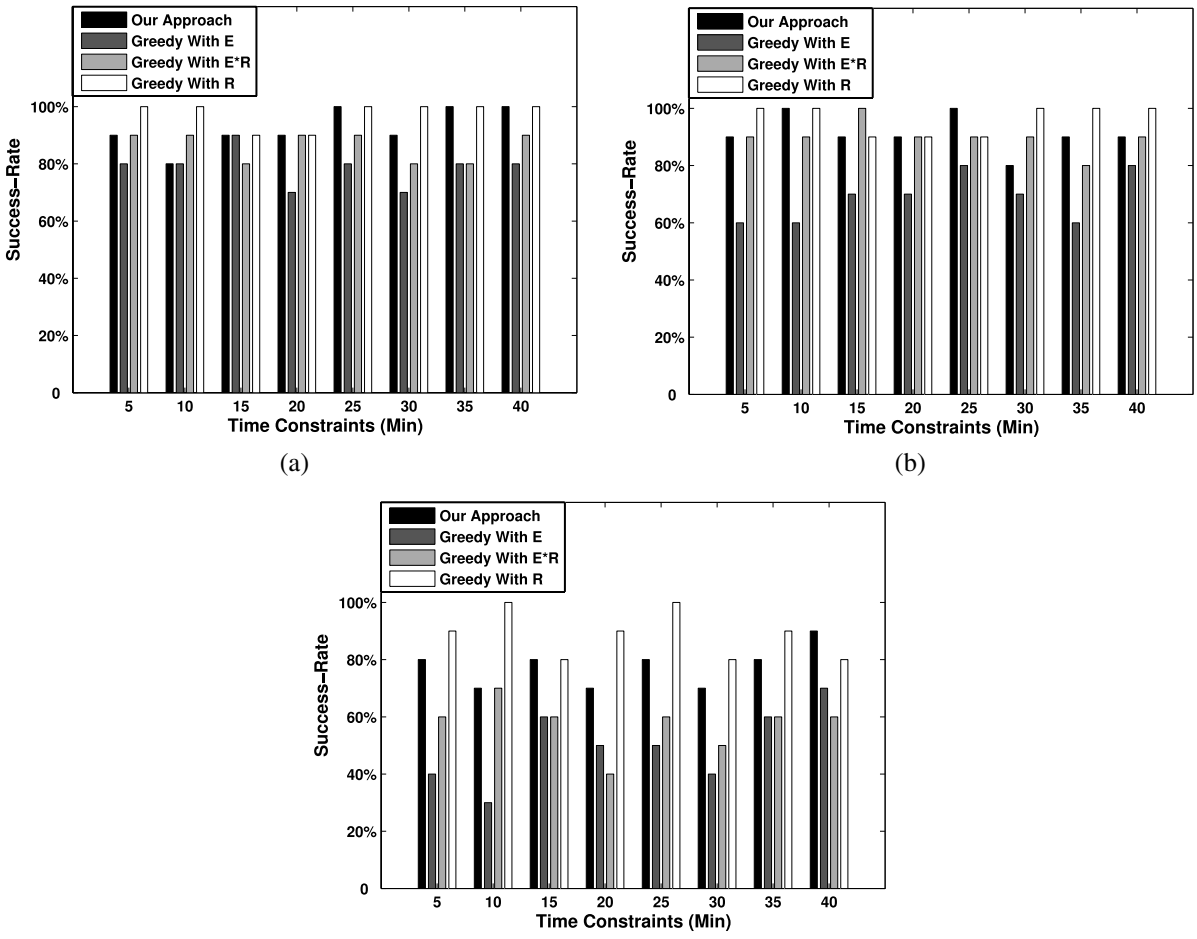


Fig. 9. Success-rate comparison of our approach with three scheduling heuristics: VolumeRendering – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

80% and 90%, respectively. The Greedy-R can achieve 100% success-rate. In an environment where resources rarely fail, selecting the resources regardless of their reliability values will not significantly impact the application performance. In such a case, we tune the weight factor α in Eq. (8) to favor the resource efficiency value. Therefore, we can achieve better benefit while minimizing the possibility of resource failure during the event handling, comparing to the other three heuristics. When the application is executed in a highly unreliable environment, as the shown in Fig. 9(c), the success-rate of Greedy-E and Greedy-$E \times R$ dramatically drop to 40% and 60%, respectively. This also explains the benefit percentage drop we discussed previously. Now our approach tunes the α parameter to favor the resource reliability. Thus, we can still reach the baseline benefit with the success-rate of 80%. Note that failure recovery is not invoked for this experiment, as we consider this in the next subsection. Similar observations can be made from the moderately reliable environment, as illustrated in Fig. 9(b). We also used the GLFS application for the success-rate comparison. Results are demonstrated in Fig. 10. The GLFS application using our algorithm can achieve 100%, 90% and 80% in the three computing environments, outperforming other approaches.

Scheduling overhead and scalability: We now evaluate the overhead of our scheduling algorithm and compare it with the overhead of the other three heuristics. We first used VolumeRendering application and the results are shown in Fig. 11(a). Note that the scheduling overhead is not dependent on the resource reliability. As shown in the figure, when the time constraints associated with events get longer, our algorithm spent

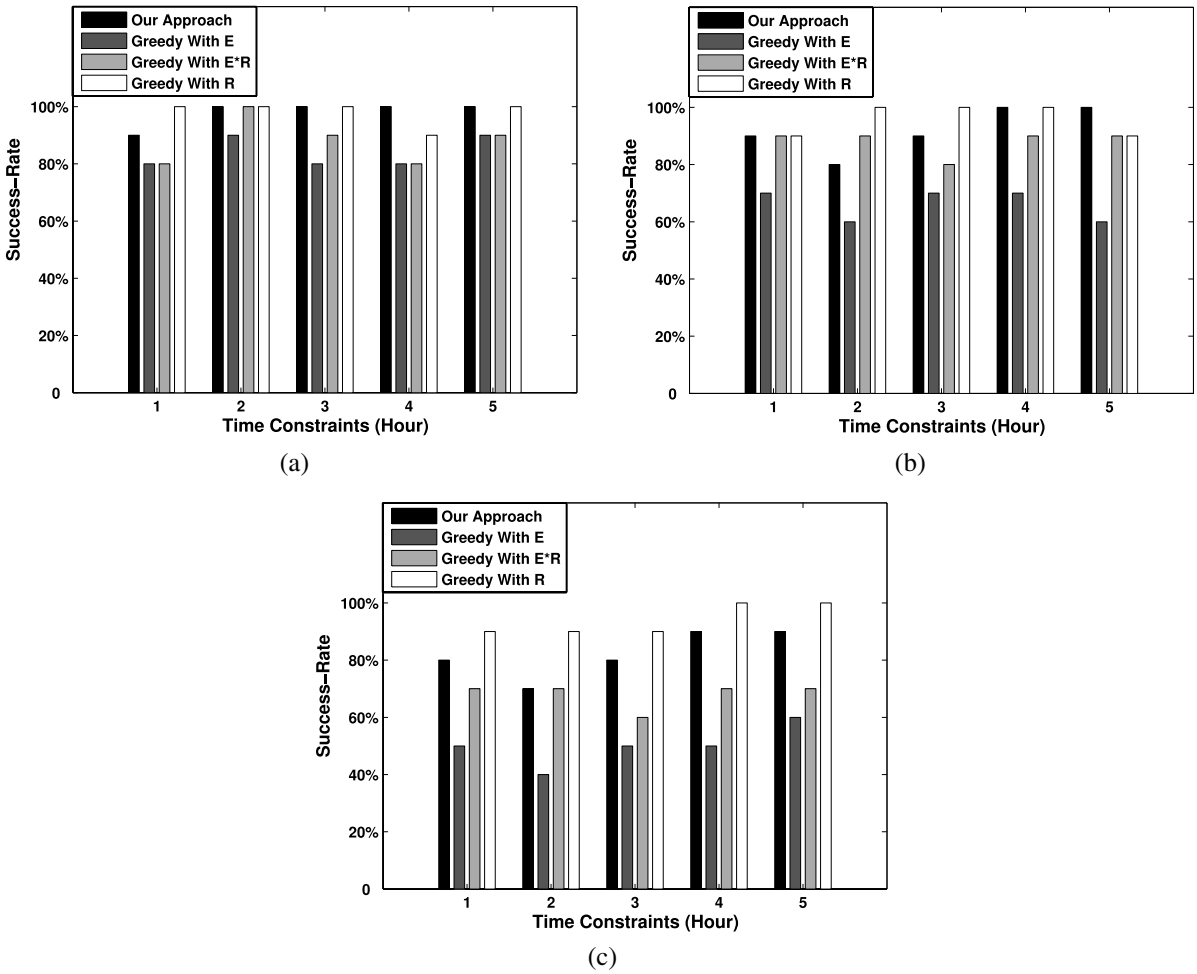


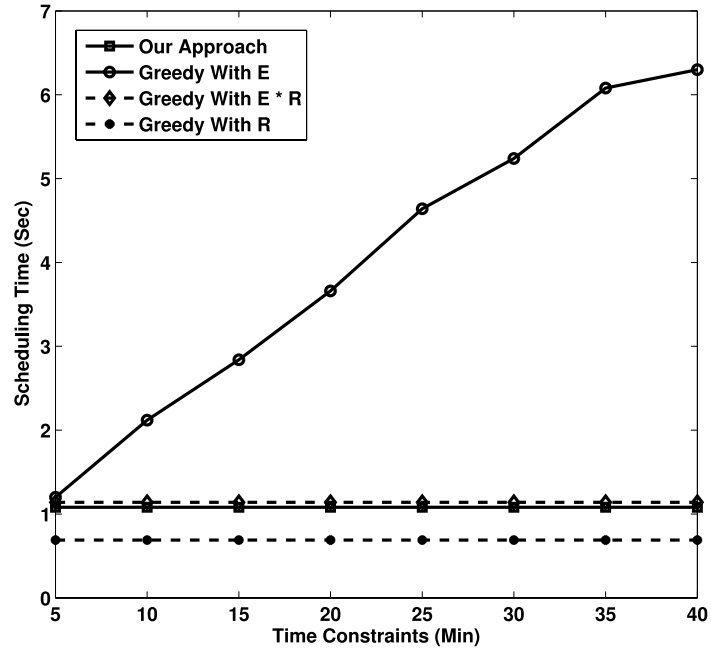
Fig. 10. Success-rate comparison of our approach with three scheduling heuristics: GLFS – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

more time on generating the resource configuration. Our algorithm scheduled the application with six service components onto 2 emulated grid sites, each with 64 nodes, in 6.3 s in the worst case. It is less than 0.3% of the application execution time, which is 40 min. In comparison, the other three heuristics only caused 1 s or less. Although our approach is several times slower, processing applications on the resources selected by our algorithm can achieve much better benefit and success-rate. The overhead of scheduling was also measured for the GLFS application and similar trend was observed.

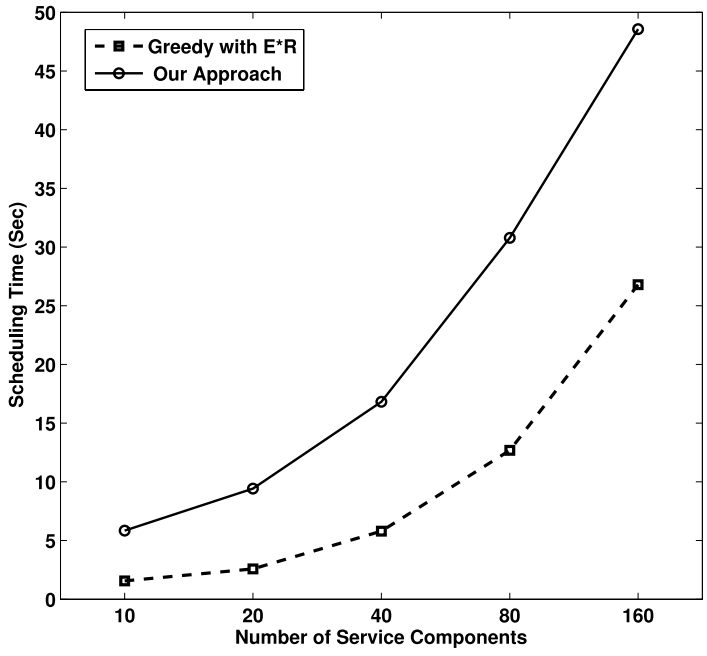
Furthermore, we evaluated the scalability of our scheduling algorithm and we demonstrate the result in Fig. 11(b). For this experiment, we simulated 640 processing nodes for a grid computing environment that is moderately reliable. We generated a synthetic application with the number of service components varying as 10, 20, 40, 80 and 160. Dependencies are involved in each case. We compare our proposed algorithm with the Greedy-$E \times R$ heuristic, since it has the most scheduling overhead among the heuristics we have considered in this paper. We have observed that the scheduling overhead increases linearly as the number of services increases and it takes less than 49 s to schedule 160 service components on 640 nodes. This demonstrates that our scheduling algorithm is scalable.

5.4. Performance of the failure recovery scheme

We now evaluate our proposed failure recovery scheme. We first apply the failure recovery scheme to the three scheduling heuristics, i.e., Greedy-E, Greedy-$E \times R$ and Greedy-R, and evaluate the obtained ben-



(a)



(b)

Fig. 11. (a) Scheduling overhead comparison of our approach with three heuristics: VolumeRendering – (b) scalability.

efit percentage with failure recovery enabled. Then, we show how our fault tolerance approach can achieve the baseline benefit and further maximize it, within the time constraint, by working synergistically with the proposed failure recovery scheme. The results of the first set of experiments from the VolumeRendering application are shown in Fig. 12. The results show that our proposed failure recovery has a very low over-

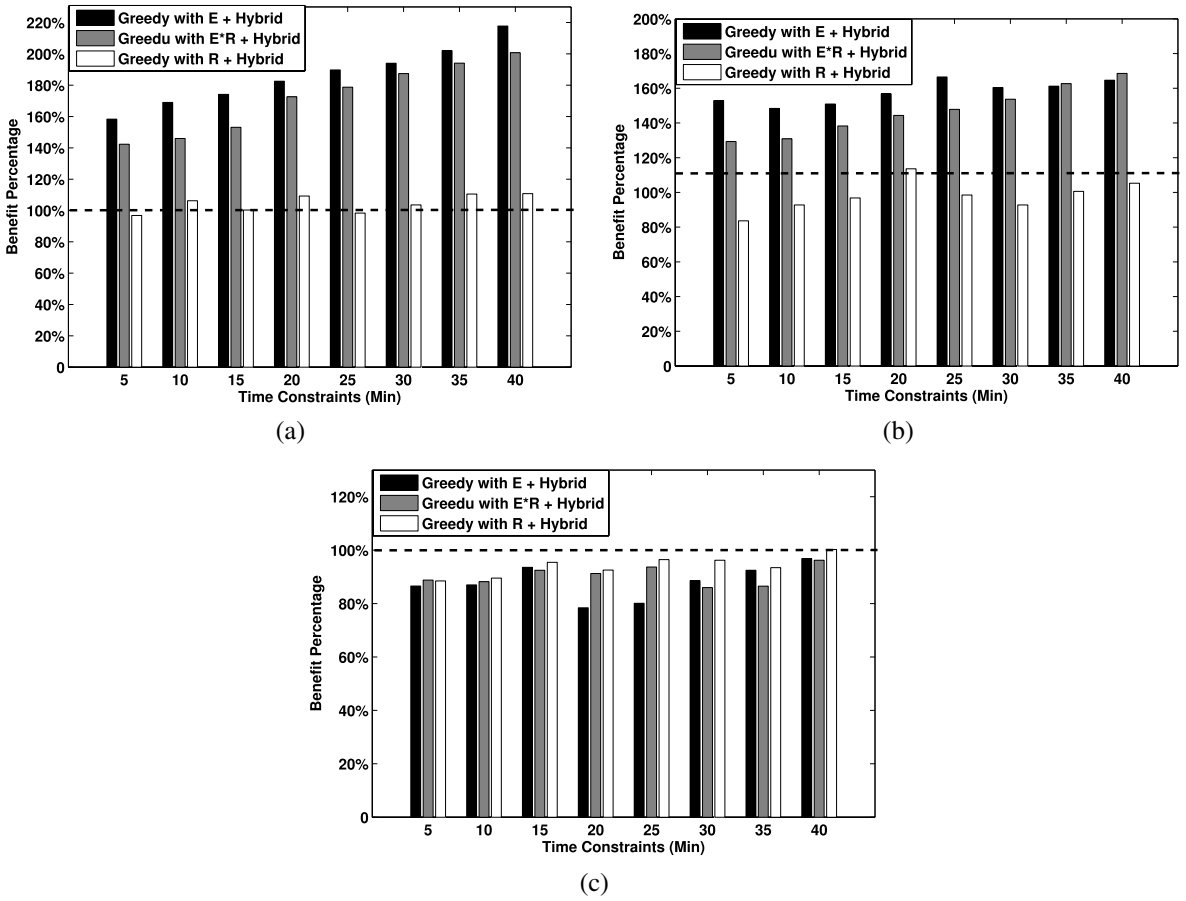


Fig. 12. Benefit percentage comparison of the three scheduling heuristics with our failure recovery scheme: VolumeRendering – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

head. In a highly reliable environment, where resource failures occur very infrequently, the benefit obtained from Greedy-E and Greedy-$E \times R$ increased up to 44% and 47%, respectively, by cooperating with our proposed failure recovery scheme. Such improvement can also be observed in the moderately reliable environment. We are able to achieve up to 38% and 29% for those two scheduling heuristics. The number of resource failures that could occur during event processing is reasonable so that failures can be quickly recovered, without consuming a large portion of the total time. The obtained benefit increased since we are able to achieve 100% success rate. However, in the highly unreliable environment, the achieved benefit is still below the baseline. With a detailed analysis, we found out that failure recovering can take up to 12% of the total processing time, due to a very large number of resource failures. Not considering the resource reliability properly, Greedy-E and Greedy-$E \times R$ select resources that could cause such frequent failures. For the Greedy-R heuristic, the benefit did not improve significantly in the three environments with failure recovery enabled. This is because the success rate of Greedy-R is already very high. Thus, it can barely benefit from using failure recovery schemes. This set of experiments demonstrate the importance of minimizing the probability of incurring resource failures during event processing, as part of the scheduling algorithm.

Next we demonstrate effectiveness of our fault tolerance approach, which is based on a reliability aware scheduling algorithm and a failure recovery scheme. The results of this set of experiments from the VolumeRendering application are presented in Fig. 13. The results show that we could further improve the obtained benefit, while achieving a 100% success-rate, in the presence of resource failures. Note that there are one, three, and five failures that occurred during application processing in the highly reliable, moderately reliable, and highly unreliable environments. We refer to the application execution without invoking any fail-

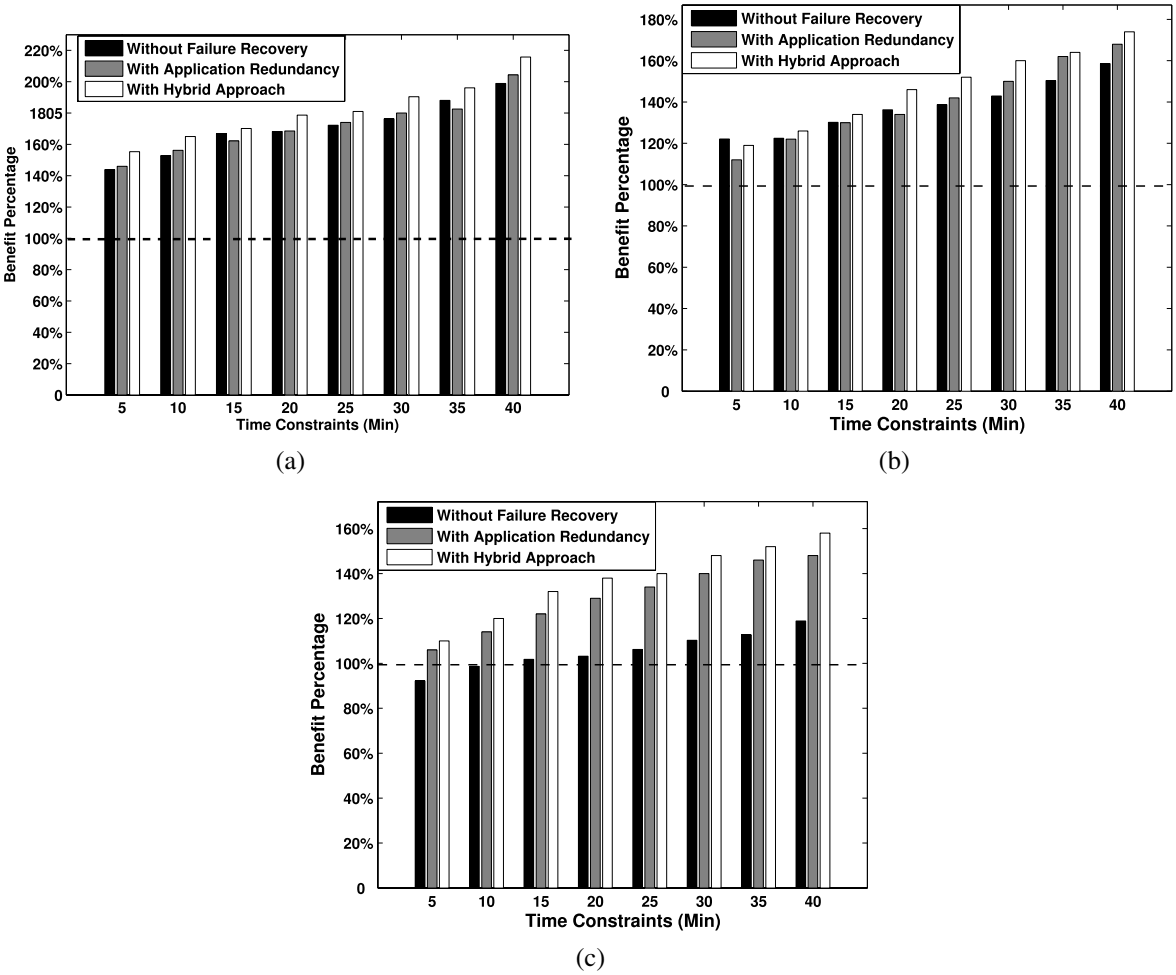


Fig. 13. Benefit percentage comparison of our failure recovery scheme: VolumeRendering – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

ure recovery as Without Recovery. Recall from Section 5.1, the approach that simply schedules multiple copies of the entire application is referred to as With Redundancy. We denote our approach as the Hybrid Approach. When comparing against Without Recovery, our proposed hybrid failure recovery scheme can improve the benefit percentage to 8%, 20% and 33%, in the three computing environments. Furthermore, the success-rate is increased to 100%. The reason is that by applying the failure recovery scheme, we incur the cost of maintaining checkpoints and synchronizing the status of multiple service copies. In the case where failures occur rarely, the relative improvement is small because of this overhead. However, as the failures occur more frequently, the recovery procedure is invoked more frequently. This leads to a more significant benefit gain. Meanwhile, our scheme always achieve 100% success-rate. We also compared our scheme with With Redundancy. An obvious drawback for this approach is the high overhead. Furthermore, it is hard to schedule redundant copies to resources where an successful run with a high benefit could be obtained, in a highly heterogeneous environment. Thus, our proposed failure recovery scheme outperforms the With Redundancy by 6%, 8% and 12% in the three computing environments. We repeat this experiment using the GLFS application and the results are presented in Figs 14 and 15. Similar observations can be made. The benefit obtained from Greedy-E and Greedy-$E \times R$ improved by 46% and 47% in highly reliable and moderately reliable environments, respectively. Our proposed failure recovery scheme can achieve 6%, 18% and 46% more benefit comparing to that from the Without Recovery version. It is 4%, 9% and 12% better when we compare the ob-

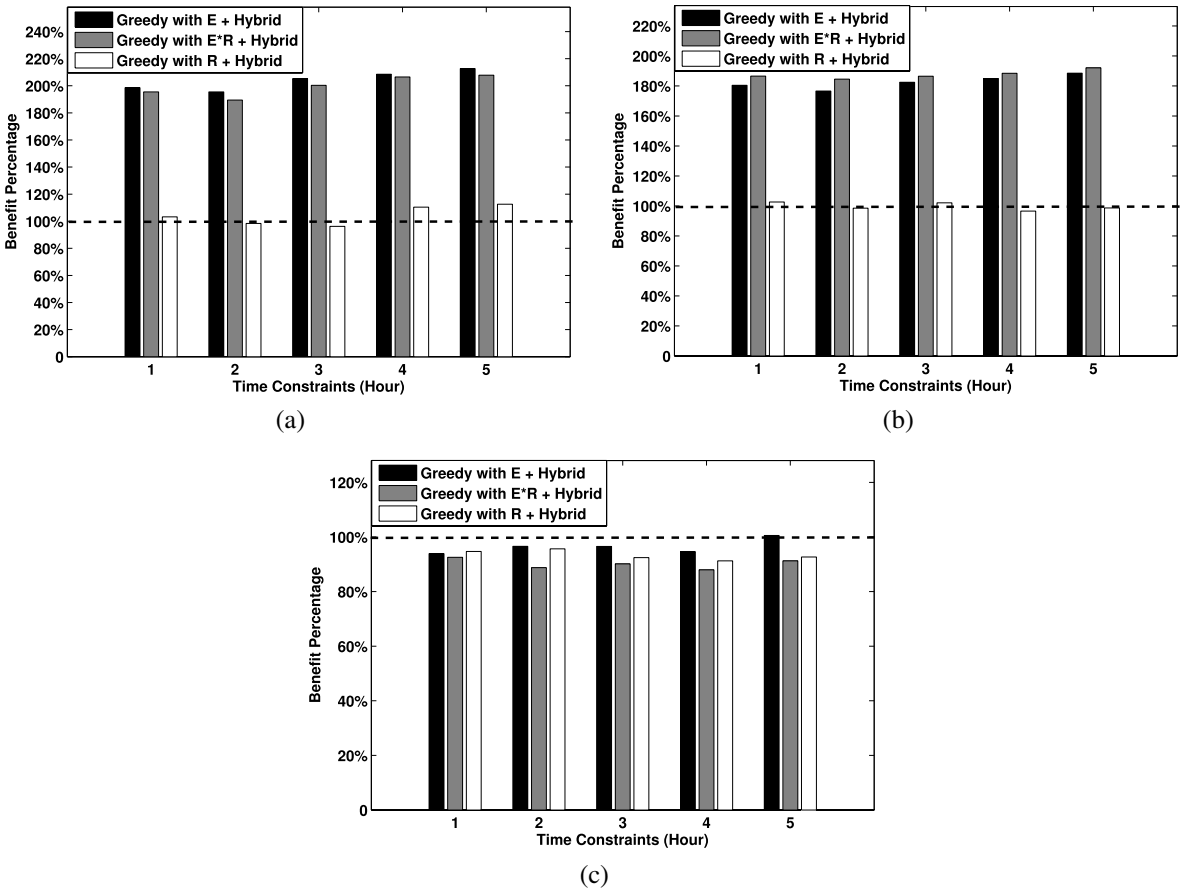


Fig. 14. Benefit percentage comparison of the three scheduling heuristics with our failure recovery scheme: GLFS – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

tained benefit percentage with With Redundancy, for the three cases, respectively.

6. Related work

We now discuss the research efforts relevant to our work from the areas of fault tolerance in grid computing and DAG-based real-time scheduling in the presence of failures.

Fault tolerance in grid computing: We particularly focus on efforts that apply reliability-aware scheduling or perform failure recovery. *Reliability-aware* scheduling has been widely studied [1,6,12,19,28]. Close to our work, Sonnek et al. [28] propose an adaptive algorithm to choose the number of replicas for each task. Our work is different because the adaptive applications we target comprise a DAG of services. Our scheduling algorithm needs to consider the dependence between the services as well as the match between the resource capacity and the resource consumption of individual services.

Many research efforts have contributed to *failure recovery* in a grid computing environment [15,26,34]. Schulz et al. [26] focus on taking checkpoints at the application level for parallel programs, using compiler technology to instrument code and enable self-checkpointing and self-restarting. Zhang et al. [34] propose a primary-backup protocol using Open Grid Services Infrastructure (OGSI) and implement it using Globus Toolkit. Our hybrid strategy chooses between replication and checkpointing, based on the characteristics of a service. The advantage of our proposed approach is lower overhead and a faster recovery time, as is critically required for deadline-driven processing. Migrating tasks to other resources in the presence of failures has also been studied in the grid computing community [9].

DAG-based real-time scheduling in presence of failures: Existing work on real-time scheduling mainly fo-

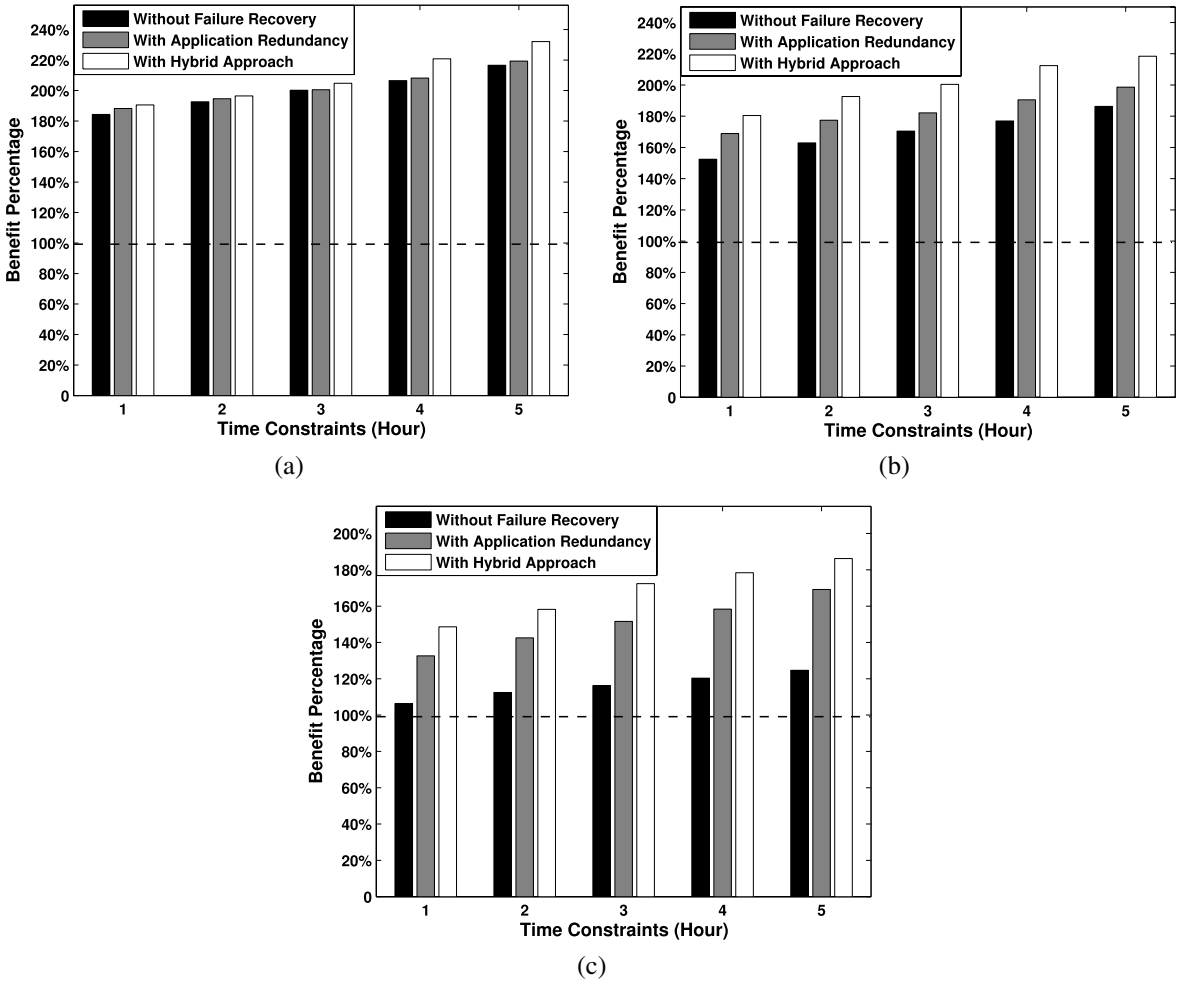


Fig. 15. Benefit percentage comparison of our failure recovery scheme: GLFS – (a) highly reliable environment; (b) moderately reliable environment; (c) highly unreliable environment.

cuses on minimizing the make-span of the application or making it satisfy certain time constraints, in presence of resource failures [8,11,20]. Here we only consider the work that involves DAG-based applications and where both the computing nodes and network links failures are taken into account. Dima et al. [11] propose an off-line scheduling algorithm that uses an active replication of tasks and communications to tolerate a set of failure patterns. Satyanarayana et al. [25] first assign time deadline to individual subtasks without violating their precedence constraint, and then apply the passive replication by scheduling both the primary and backup copies of each sub-task. Our work has the following distinctive aspects. First, we consider heterogeneous resources. Second, besides completing the event within the pre-specified time interval, we also want to achieve the baseline benefit and further improve on it, which is more complex than the minimization of the application make-span.

Since our scheduling task has two objectives, i.e., maximizing application benefit and reliability, to achieve, our work is also related to the problem of bi-criteria scheduling [4,12,27,33]. Assayad et al. [4] proposed a list scheduling heuristic with the goal of minimizing the scheduling length, while maximizing the system reliability. Singh et al. [27] applied the multi-objective optimization concept and used a genetic algorithm to minimize both the resource provisioning cost and application make-span. We have also used the Pareto-optimal set and we applied the Particle-swarm Optimization algorithm to generate the best resource configuration, given the compromise objective function. The advantage of our proposed algorithm is that we can achieve faster convergence by searching the re-

source configuration candidates that lead towards the objective function.

7. Conclusion

A key problem faced by applications executing in a grid computing environment is the inherent unreliability of the resources. In this paper, we have focused on the problem of supporting fault-tolerance for adaptive applications. Our goal is to achieve the baseline benefit and further improve it within the pre-specified time interval for a processing event, when resource failures can occur. We have proposed a reliability-aware scheduling algorithm based on the concept of Multi-objective Optimization (MOO). Furthermore, when a resource failure does occur, we use a hybrid failure recovery scheme for efficient recovery. We have applied our approach to two adaptive applications, namely, VolumeRendering and GLFS. Experimental evaluation has demonstrated that by applying our proposed scheduling algorithm, the obtained benefit can always achieve the baseline and further improve it up to 206%, with a success-rate of 80% . If combined with failure recovery, the obtained benefit improves up to 33% and the success-rate is now 100%. Furthermore, the overhead of the our scheduling algorithm is very low, for example, less than 0.3% for a 40 min event.

Acknowledgements

This work was supported by NSF Grants 0541058 and 0619041. The equipment used for the experiments reported here was purchased under the Grant 0403342.

References

[1] J. Abawajy, Fault-tolerant scheduling policy for grid computing systems, in: *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS04)*, Santa Fe, NW, USA, April 2004, pp. 238–249.

[2] S. Agrawal, Y. Dashora, M.K. Tiwari and Y. Son, Interactive particle swarm: A Pareto-adaptive metaheuristic to multiobjective optimization, *IEEE Transaction on System, Man and Cybernetics* **38**(2) (2008), 258–277.

[3] D. Angulo, I. Foster, C. Liu and L. Yang, Design and evaluation of a resource selection framework for grid applications, in: *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC02)*, Paris, France, June 2002, pp. 63–72.

[4] I. Assayad, A. Girault and H. Kalla, A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints, in: *Proceedings of the International Conference on Dependable Systems and Networks (DSN04)*, Florence, Italy, June 2004, pp. 347–356.

[5] K. Bedford and D. Schwab, The great lakes forecasting system – Lake Erie nowcasts/forecasts, in: *Proceedings of the Marine Technology Society Annual Conference (MTS91)*, Washington, DC, USA, October 1991, pp. 260–264.

[6] K. Budati, A. Chandra and J.B. Weissman, Ridge: Combining reliability and performance in open grid platforms, in: *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC07)*, Monterey, CA, USA, June 2007, pp. 55–64.

[7] R. Buyya and M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, *Concurrency and Computation: Practice and Experience* **14** (2002), 1175–1220.

[8] M. Caccamo and G. Buttazzo, Optimal scheduling for fault-tolerant and firm real-time systems, in: *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA98)*, Hiroshima, Japan, October 1998, pp. 223–231.

[9] A. Chakrabarti, S. Senqupta, A. Upadhyay and A. Damodaran, A systematic approach for application migration in a grid computing environment, in: *Proceedings of the IEEE Asia-Pacific Conference on Services Computing (APSCC06)*, Guangzhou, China, December 2006, pp. 512–519.

[10] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, Wiley, New York, 2001.

[11] C. Dima, A. Girault and Y. Sorel, Static fault-tolerant real-time scheduling with pseudo-topological orders, in: *Journal of Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, LNCS, Vol. 3523, Springer-Verlag, Heidelberg, Germany, 2004, pp. 215–230.

[12] A. Dogan and F. Ozguner, Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems, *The Computer Journal* **48**(3) (2004), 300–314.

[13] R. Drebin, L. Carpenter and P. Hanrahan, Volume rendering, in: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, Atlanta, GA, USA, August 1988, pp. 65–74.

[14] S. Fu and C.Z. Xu, Exploring event correlation for failure prediction in coalitions of clusters, in: *Proceedings of the 21st International Conference on High Performance Computing and Networking (SC07)*, Reno, NV, USA, November 2007, pp. 1–12.

[15] S. Hwang and C. Kesselman, A flexible framework for fault tolerance in the grid, *Journal of Grid Computing* **1**(3) (2002), 251–272.

[16] S. Kartik and C.S.R. Murthy, Improved task-allocation algorithms to maximize reliability ofredundant distributed computing systems, *IEEE Transactions on Reliability* **44** (1997), 575–586.

[17] Y.S. Kee, H. Casanova and A. Chien, Realistic modeling and synthesis of resources for computational grids, in: *Proceedings of the 18th International Conference on High Performance Computing and Networking (SC04)*, Pittsburgh, PA, USA, November 2004, pp. 54–63.

[18] J. Kennedy and R.C. Eberhart, *Swarm Intelligence*, Morgan Kaufmann, San Mateo, CA, USA, 2001.

[19] D. Kondo, A. Chien and H. Casanova, Resource management for rapid application turnaround on enterprise desktop grids, in: *Proceedings of the 18th International Conference on High Performance Computing and Networking (SC04)*, Pittsburgh, PA, USA, November 2004, pp. 180–193.

[20] X. Qin and H. Jiang, A dynamic and reliability driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters, *Journal of Parallel and Distributed Computing* **65**(8) (2005), 885–900.

[21] R. Raman, M. Livny and M. Solomon, Policy driven heterogeneous resource co-allocation with gangmatching, in: *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03)*, Seattle, WA, USA, June 2003, pp. 80–89.

[22] R. Roshandel, N. Medvidovic and L. Golubchik, A Bayesian model for predicting reliability of software systems at the architectural level, in: *Proceedings of 3rd International Conference on Software Architectures, Components, and Applications*, Boston, MA, USA, July 2007.

[23] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, USA, 2003.

[24] M.F. Santarelli, V. Positano and L. Landini, Real-time multimodal medical image processing: a dynamic volume-rendering application, *IEEE Transactions on Information Technology in Biomedicine* **1** (1997), 171–178.

[25] N.V. Satyanarayana, R. Mall and A. Pal, Fault-tolerant scheduling in distributed real-time systems, in: *Proceedings of the International Conference on Computer Networks and Mobile Computing (ICCNMC01)*, Los Alamitos, CA, USA, October 2001, pp. 275–280.

[26] M. Schulz, G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali and P. Stodghill, Implementation and evaluation of a scalable application-level checkpoint-recovery scheme for mpi programs, in: *Proceedings of the 18th International Conference on High Performance Computing and Networking (SC04)*, Pittsburgh, PA, USA, November 2004, pp. 38–51.

[27] G. Singh, C. Kesselman and E. Deelman, A provisioning model and its comparison with best-effort for performance-cost optimization in grids, in: *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing(HPDC07)*, Monterey, CA, USA, June 2007, pp. 117–126.

[28] J.D. Sonnek, A. Chandra and J.B. Weissman, Adaptive reputation-based scheduling on unreliable distributed infrastructures, *IEEE Transactions on Parallel and Distributed Systems* **18**(11) (2007), 1551–1564.

[29] P. Trunfio, D. Talia, P. Fragopoulou, C. Papadakis, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov and S. Haridi, Peer-to-peer models for resource discovery on grids, Technical Report TR-0028, Institute on System Architecture, CoreGRID, March 2006.

[30] C. Wang, A. Garcia and H.W. Shen, Interactive level-of-detail selection using image-based quality metric for large volume visualization, *IEEE Transactions on Visualization and Computer Graphics* **13** (2006), 122–134.

[31] C. Wang and H.W. Shen, A framework for rendering large time-varying data using wavelet-based time–space partitioning (wtsp) tree, Technical Report OSUCISRC-1/04-TR05, Department of Computer and Information Science, The Ohio State University, January 2004.

[32] J. Yu and R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, *Scientific Programming Journal* **14**(3) (2006), 217–230.

[33] J. Yu and R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: *The Workshop on Workflows in Support of Large-Scale Science (WORKS06)*, Paris, France, June 2006, pp. 1–10.

[34] X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo and R. Schlichting, Fault-tolerant grid services using primary-backup: Feasibility and performance, in: *Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster04)*, San Diego, CA, USA, September 2004, pp. 105–114.

[35] Q. Zhu and G. Agrawal, An adaptive middleware for supporting time-critical event response, in: *Proceedings of the 5th International Conference on Autonomic Computing (ICAC08)*, Chicago, IL, USA, June 2008, pp. 99–108.

[36] Q. Zhu and G. Agrawal, A resource allocation approach for supporting time-critical event handling in grid environments, in: *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS09)*, Rome, Italy, May 2009, pp. 1–12.