

A Provisioning Model and its Comparison with Best Effort for Performance-Cost Optimization in Grids

Gurmeet Singh, Carl Kesselman, Ewa Deelman
Information Sciences Institute, Marina Del Rey, CA 90292
{gurmeet, carl, ewa}@isi.edu

Abstract

The resource availability in Grids is generally unpredictable due to the autonomous and shared nature of the Grid resources and stochastic nature of the workload resulting in a best effort quality of service. The resource providers optimize for throughput and utilization whereas the users optimize for application performance. We present a cost-based model where the providers advertise resource availability to the user community. We also present a multi-objective genetic algorithm formulation for selecting the set of resources to be provisioned that optimizes the application performance while minimizing the resource costs. We use trace-based simulations to compare the application performance and cost using the provisioned and the best effort approach with a number of artificially generated workflow-structured applications and a seismic hazard application from the earthquake science community. The provisioned approach shows promising results when the resources are under high utilization and/or the applications have significant resource requirements.

1. Introduction

Shared distributed infrastructures such as the Teragrid [1], the Open Science Grid [2], etc provide the software and hardware resources for cross-organizational collaborative work in different scientific fields such as astronomy [3], high energy physics [4], earthquake science [5], etc. Such collaborations, also called Virtual Organizations (VO) [6] require resources from multiple providers for executing large-scale scientific applications. Furthermore, emerging classes of deadline driven scientific applications such as severe weather modeling [7] require simultaneous access to multiple resources and predictable completion times. Most of the Grid resources are managed using queuing-based local resource management systems such as PBS [8], LSF [9], Condor [10], etc. Due to the shared nature of these resources and the dynamic nature of the workload, the completion time of the application tasks on these resources is not predictable. Moreover, the applications require co-allocation of multiple resources such as storage and network etc for storing the input data and the generated results, communicating data between executing tasks etc. In the absence of a more explicit

control, resources are offered to an application on a best effort basis with little or no influence as to exactly when the resource will be delivered to the application. This makes it difficult to predetermine, much less optimize the resulting application performance and meet deadline constraints and performance goals.

Agreement-based resource management [11] is an alternative to the best effort execution for overcoming these deficiencies. The agreement-based management model allows the VO-level resource brokers to provision resources ahead of the execution of the applications by entering into agreements with the resource providers about the guaranteed availability of desired resources for a mutually agreed upon timeframe and cost. While there can be various approaches for creating agreements, in this paper, we use a model where the providers create resource offers or slots and advertise them to the general community periodically or on demand. Each slot represents the availability of a certain resource capability such as the number of processors, for a certain timeframe (start time and duration), for a certain cost and can be provisioned independently of any other offered slot i.e. we don't assume overbooking of resources. Once a slot has been provisioned, it can be used to execute any workload e.g. set of tasks subject to the capacity constraints of the slot without any further interaction with the provider. While slot-based management significantly reduces the unpredictability in application performance over best effort execution of applications, it introduces a new issue: how to identify and select a set of slots to provision for an application that would allow the execution of the application while optimizing the application performance and minimizing the cost of the provisioned resources from the perspective of the user. While there can be many performance objectives such as the execution time, reliability etc, in this paper, we use the execution time of the application, e.g. the makespan of a workflow, as the performance metric that has to be minimized.

The slot selection problem is non-trivial because of the possibility of a large number of resource slots with different capabilities on offer from various providers at the time when an application is submitted and the fact that the resource providers are free to ask any price for an offered

slot. Indeed, we can imagine a model similar to yield management [12] where similar resource capability is available immediately at a higher price while being available at some future time at a lower price, similar to advance purchase discounts for airline ticketing [13]. Thus the objectives of minimizing the application makespan and minimizing the total cost of the provisioned resources may be inconsistent or in conflict with each other. In a previous work [14], we initially explored the problem with two algorithms for minimizing the weighted sum of the two objectives. However, since the objectives are generally non-commensurate, in this paper, we create a set of Pareto-optimal solutions using a multi-objective genetic algorithm (MOGA) formulation [15] of the problem and then select one solution from this set using a specified trade-off factor and a normalized objective function.

Resource provisioning for applications is an endeavor worth undertaking if only for the fact that it allows for the predictable and deterministic execution of applications and performance optimization without worrying about external factors that might affect the execution of the application. However, in this paper, we also look at how the application performance with resource provisioning might compare with the application performance using the best effort quality of service using trace-based simulations and workload logs from two supercomputer centers. In addition to the application performance, we also compare the cost of the two approaches.

In order to compare the provisioned execution of the application with its best effort counterpart, we extended a Grid resource scheduler to support both qualities of service concurrently. We used a range of artificially generated applications and a seismic hazard analysis application from the earthquake engineering community [5] and compared their performance and cost with the best effort and the provisioned approach using this extended scheduler. Two clear trends were observed from the results. First is that the provisioned approach becomes increasingly attractive as the tasks in the workflow require more compute resources. Second, the provisioned approach fares better than the best effort execution of an application as the resources become loaded. Both of these characteristics are visible at the current operational Grid sites such as TeraGrid. These sites target large-scale applications with highly parallel tasks which are a natural match for these resources. Moreover due to the shared nature of these resources, the large number of users, and their production nature, these sites generally show high to very high utilization levels (60-80+%) [16].

The rest of the paper is structured as follows. Section 2 presents the provisioning model. Section 3 presents the principle of Pareto optimality and describes a heuristic for the provisioning problem and some initial results. Section 4 presents the design of the extended Grid scheduler. Sections 5 and 6 compare the performance of the best

effort and provisioned approach using generated and real applications. Section 7 presents the discussion followed by related work in Section 8 and conclusions in Section 9.

2. Provisioning Model

We assume the system to be composed of $r = 1 \dots R$ Grid sites. The latency and bandwidth between these sites is assumed to be known. Each site advertises its resource availability in the form of a set of resources slots or offers. While the model can be extended to other types of resources, in this paper, we focus our attention on compute resources only. Each slot represents the availability of certain number of processors for a certain duration starting at a certain time at a certain cost. The set of resource slots available from resource r are denoted by E_r .

$$E_r = \{ \langle s_1, n_1, d_1, c_1, f_1, b_1, e_1 \rangle, \dots, \langle s_i, n_i, d_i, c_i, f_i, b_i, e_i \rangle, \dots \} \quad (\text{Eq 1})$$

where $\langle s_i, n_i, d_i, c_i, f_i, b_i, e_i \rangle$ represents the availability of n_i processors for duration d_i starting at time s_i from resource r . b_i is a boolean indicating whether the slot has to be provisioned in its entirety or can be partially provisioned ($b_i = \text{true}$:divisible, $b_i = \text{false}$:non-divisible). The term c_i is called the multiplicative cost and represents the per-unit charge for the resource and the term f_i is called the additive cost and represents the overhead cost of provisioning the resource. e_i indicates whether the duration of the slot can be extended past the specified endtime ($s_i + d_i$), i.e. in case we want to provision resources for tasks with longer runtimes than the specified slot duration.

Suppose, the broker wants to provision n_k processors for duration d_k starting at time s_k when the offered slot is $\langle s_i, n_i, d_i, c_i, f_i, b_i, e_i \rangle$. Then the following possibilities with associated costs exist based on the values of b_i and e_i .

Table 1. Possible subslot allocations.

b_i	e_i	restrictions	cost
false	false	$n_k = n_i, s_k = s_i, d_k = d_i$	$n_i * d_i * c_i + f_i$
false	true	not allowed	
true	false	$n_k \leq n_i, s_i \leq s_k, s_k + d_k \leq s_i + d_i$	$n_k * d_k * c_i + f_i$
true	true	$n_k \leq n_i, s_i \leq s_k$	$n_k * d_k * c_i + f_i$

The set of resource slots available from resource r between times t_0 and t_n are denoted by E_{r,t_0,t_n}

$$E_{r,t_0,t_n} = \{ \langle \dots \rangle \in E_r \mid t_0 \leq s_i \leq t_n \} \quad (\text{Eq 2})$$

The global resource availability, denoted by $\hat{A}_{0,t}$ is the combination of the aggregated resource availability of each resource from the current time to some time in future, t .

$$\hat{A}_{0,t} = \bigcup_{r=1..R} E_{r,0,t} \quad (\text{Eq 3})$$

Application Model

For the purposes of evaluation we assume that an application can be represented as a set of tasks with

dependencies, also known as a workflow or a DAG (directed acyclic graph). We note that our techniques can be applied to other application models as well, however within this paper, we use the term application and workflow interchangeably. The task runtimes on each of the computational resources is assumed to be known. The runtimes could be estimated analytically using component performance modeling [17] or empirically using historical information [18]. The amount of data transferred between each parent and child task in the application is assumed to be known. If the parent and child tasks are executed on the same site, the data transfer time is assumed zero. Otherwise, the data transfer time can be computed using bandwidth and latency information.

Resource Provisioning Problem

The goal of resource provisioning is to identify a subset \hat{a} of $\hat{A}_{0,t}$ such that the application makespan and the provisioning costs are minimized. The subset \hat{a} is also called the resource plan. The provisioning cost of the resource plan \hat{a} , is termed as the allocation cost and is denoted by $AC(\hat{a})$. The allocation cost is defined as

$$AC(\hat{a}) = \sum_{\langle i \rangle \in \hat{a}} (n_i \cdot d_i \cdot c_i + f_i) \quad (\text{Eq 4})$$

The allocation cost represents the total payment for allocating the slots in the plan from the providers. Minimizing the allocation cost encourages the efficient utilization of the provisioned resources since any unused capacity represents an unnecessary addition to the allocation cost.

The application makespan on \hat{a} is termed as the scheduling cost of the plan, $SC(\hat{a})$.

$$SC(\hat{a}) = \text{makespan of the application over } \hat{a}. \quad (\text{Eq 5})$$

In order to determine the makespan of the application over the resource plan, \hat{a} , a scheduling algorithm is required. Since the resource availability and the application task runtimes are known deterministically, the schedule of the application tasks over \hat{a} can be computed using any DAG scheduling heuristic. We use the Heterogeneous Earliest Finish Time (HEFT) [19] that is described in section 3 and is regarded as a good scheduling heuristic. A resource plan is called infeasible if the application cannot be completely scheduled onto it and its scheduling cost is considered infinite.

The resource provisioning problem faced by the resource broker is the following multi-objective optimization problem

$$\min_{\hat{a} \subseteq \hat{A}} \begin{bmatrix} AC(\hat{a}) \\ SC(\hat{a}) \end{bmatrix} \quad (\text{Eq 6})$$

where both the allocation cost, $AC(\hat{a})$ and the scheduling cost, $SC(\hat{a})$ are sought to be minimized. In the

next section, we describe the heuristic for obtaining a solution using the concept of Pareto-optimality.

3. Concept of Pareto Optimality and MOGA Evaluation

While two different solutions can be directly compared to each other based on the value of a single objective, it is not possible to do so in the case of multiple objectives. For example, a particular resource plan, \hat{a}_1 may have a lower allocation cost and a higher scheduling cost than another resource plan, \hat{a}_2 . Thus \hat{a}_1 and \hat{a}_2 cannot be directly compared with each other. However, if \hat{a}_1 or \hat{a}_2 have a lower allocation cost and a lower scheduling cost than another resource plan \hat{a}_3 , then both \hat{a}_1 and \hat{a}_2 are superior to \hat{a}_3 . We use the concept of domination from [20] in order to compare two resource plans in the context of our optimization problem.

A resource plan \hat{a}_i is said to dominate another resource plan \hat{a}_j , if both condition 1 and 2 are true:

1. The allocation cost and scheduling cost of \hat{a}_i is no worse than that of \hat{a}_j i.e. $AC(\hat{a}_i) \leq AC(\hat{a}_j)$ and $SC(\hat{a}_i) \leq SC(\hat{a}_j)$
2. The solution \hat{a}_i is strictly better than \hat{a}_j in at least one objective i.e. either $AC(\hat{a}_i) < AC(\hat{a}_j)$ or $SC(\hat{a}_i) < SC(\hat{a}_j)$ or both.

If any of the above conditions is violated, \hat{a}_i does not dominate \hat{a}_j . Figure 1 shows the allocation cost and scheduling cost of five hypothetical solutions, each representing a different resource plan. As can be seen, none of the solutions is optimal with respect to both objectives. Solution \hat{a}_4 is dominated by \hat{a}_2 and solution \hat{a}_5 is dominated by both \hat{a}_2 and \hat{a}_3 . Solutions \hat{a}_1 , \hat{a}_2 , \hat{a}_3 do not dominate each other and are not dominated by any other solution. Given, the entire set of solutions to a multi-objective problem, the set of solutions that are not dominated by any other solution in the entire set is known as *Pareto-optimal* set. For example, in Figure 1, if the entire solution set was composed of the given five solutions, then \hat{a}_1 , \hat{a}_2 , and \hat{a}_3 would form the Pareto-optimal set.

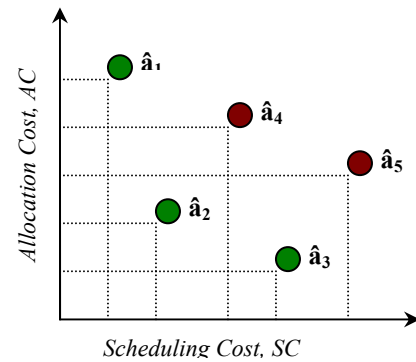


Figure 1. The solutions \hat{a}_1 , \hat{a}_2 , and \hat{a}_3 dominate the solutions \hat{a}_4 and \hat{a}_5 .

Thus a multi-objective problem can have multiple solutions represented by the Pareto-optimal set instead of having a single solution as would be the case in a single objective optimization problem. However, since a single solution is usually required for implementation instead of multiple ones, the most common approach in multi-objective optimization problems is to create a weighted sum of the objectives as the single objective. For example, we could have created the following single objective formulation (Equation 7) of the resource provisioning problem instead of using Equation 6 and found a solution using any search based heuristic such as taboo search, simulated annealing etc.

$$\min_{\hat{a} \in \hat{A}} \alpha.AC(\hat{a}) + (1 - \alpha).SC(\hat{a}) \quad (\text{Eq 7})$$

However, the drawback of this approach is that the value of the allocation and the scheduling costs are not normalized in this equation. The range of the possible values of allocation cost and the scheduling cost for a given $\hat{A}_{0,t}$ and an application is generally non-commensurate and not known in advance. Without normalizing these costs, any weight factor would not have the desired effect. The Pareto-optimal set allows the user to normalize the objective function values by defining the following normalization functions.

$$AC_{nr}(\hat{a}) = \frac{AC(\hat{a}) - AC_{\min}}{AC_{\max} - AC_{\min}}, \quad SC_{nr}(\hat{a}) = \frac{SC(\hat{a}) - SC_{\min}}{SC_{\max} - SC_{\min}} \quad (\text{Eq 8})$$

where, $AC_{\min(\max)}$ = minimum(maximum) allocation cost and $SC_{\min(\max)}$ = minimum(maximum) scheduling cost of any solution in the Pareto-optimal set

The approach that we have taken in this paper is to create the Pareto-optimal set of solutions (PO) and then select a solution from the set using the following objective function (Equation 9) and a trade-off factor, α [0,1]

$$\min_{\hat{a} \in PO} \alpha.AC_{nr}(\hat{a}) + (1 - \alpha).SC_{nr}(\hat{a}) \quad (\text{Eq 9})$$

The trade-off factor represents the user's preference for the allocation and the scheduling cost. Using this approach multiple solutions representing different tradeoffs can be selected from the Pareto-optimal set without any extra effort. In the single objective formulation, each desired tradeoff would require the search to be repeated again with new weights. Moreover, any solution to the single objective optimization problem (Equation 7) must be a Pareto-optimal solution.

Most search procedures such as taboo search, simulated annealing, etc work with a single solution at a time and are designed to find a single optimal solution. Genetic algorithms (GA) on the other hand work with a population of solutions and hence are well-suited to find the Pareto-optimal set. There has been a tremendous amount of work in the last decade on using GA for multi-objective optimization where the goal is to find the Pareto-

optimal set [20, 21]. In this paper, we use a multi-objective GA (called MOGA) version first introduced by Fonseca and Fleming [15] in 1993 for creating the Pareto-optimal set of solutions. While there are other well known formulations of multi-objective GA such as Non-Dominated Sorting GA and niched Pareto GA [20], we use MOGA for its simplicity and the fact that it operates in the objective variables space rather than the decision variable space (explained below).

Multi-objective Genetic Algorithm (MOGA) formulation

All multi-objective Genetic algorithms are modifications of their single objective versions in special ways and do not differ significantly from their single objective counterparts. A resource plan in MOGA is encoded as a n bit binary number where $n = |\hat{A}_{0,t}|$. Each bit in the number represents a slot in $\hat{A}_{0,t}$. If the bit is 1, the slot is included in the plan otherwise it is not. The fact that MOGA operates in the objective variable space implies that it uses allocation cost and scheduling cost values of resource plans for determining their fitness rather than the absolute value of the binary number representing the plan which doesn't represent any meaningful value. The scheduling cost of a resource plan is determined by the makespan of the application as determined by applying a specified scheduling algorithm, such as HEFT (described below) to the workflow and slots. The allocation cost is computed from the size, duration, multiplicative and additive costs of each slot in the plan (Equation 4).

The detailed working of MOGA is described in [20] and a high-level description follows. MOGA, like any other GA operates using a population of certain number of solutions. Each solution is a resource plan. The initial population is created by randomly picking numbers between 0 and $(2^n - 1)$ and creating resource plans based on the binary representation of the numbers. The allocation and scheduling cost of each solution in the population is evaluated and the non-dominated members of this initial population are added to a Pareto-optimal set. MOGA goes through a specified number of iterations. At each iteration,

- a new population is created from the old one using the genetic operators of selection, cross-over and mutation.
- A combined set is created by adding the new population to the members of the Pareto-optimal set.
- The non-dominated members of this combined set becomes the new Pareto-optimal set.
- The new population replaces the old population.

At the end of the specified number of iterations, the current Pareto-optimal set is returned as the solution. Note that the Pareto-optimal solutions created by MOGA are only an approximation to the real set of Pareto-optimal solution since MOGA like any other GA is a stochastic algorithm and cannot guarantee optimality. However, for

the purposes of this paper, we will refer to the set of solutions returned by MOGA as the Pareto-optimal set.

Heterogeneous Earliest Finish Time (HEFT) scheduling heuristic

HEFT[19] has been found to perform better than other scheduling heuristics in some studies [22] and also in our experiments (later this section). HEFT was originally developed for scheduling task graphs on heterogeneous dedicated multiprocessing systems. It works by assigning a rank to each task in the application and then scheduling tasks based on the rank. While in the original HEFT[19], a task can be scheduled on any processor, in our case, it may not be possible to schedule a task on a slot if there isn't sufficient overlap between the timeframe when the slot is active and the possible execution time of the task. Figure 2 lists the HEFT algorithm along with the *slotSchedule* routine that checks for the feasibility of running a task on a slot. If HEFT returns failure to schedule a task on any slot in step 6 (Figure 2), then the resource plan is considered infeasible and its scheduling cost is considered infinite.

```

Compute ranks of all application tasks (bottom-up)
Rank( $n_i$ ) =  $\underline{w}_i + \max_{n_j \in \text{Csucc}(n_i)} (\underline{c}_{i,j} + \text{rank}(n_j))$ 
 $\underline{w}_i$  = average runtime of task  $i$ 
 $\underline{c}_{i,j}$  = average data transfer time between task  $i$  and  $j$ 
1. Sort the tasks in a list by non-increasing rank values
2. While there are unscheduled tasks in the list do
3.   Select the first unscheduled task,  $n_i$  from the list
4.   For each slot  $s$  in the resource plan
5.     Compute the finish time of  $n_i$  on  $s$  (slotSchedule( $s, n_i$ ))
6.   If no slot can schedule  $n_i$ , return failure, else schedule  $n_i$  on the
     slot that leads to the earliest finish time, update the slot schedule
7. Endwhile
8. Return the application task schedule on the slots in the plan.

numproc( $n_i$ ) = number of processors required by task  $n_i$ 
runtime( $n_i, s$ ) = runtime of task  $n_i$  on slot  $s$ 
numproc( $s$ ) = number of processors in slot  $s$ 
runtime( $s$ ) = duration of slot  $s$ 
starttime( $s$ ) = starttime of slot  $s$ 
est( $n_i, s$ ) = earliest possible start time of  $n_i$  on the slot  $s$  based on finish
time of its parents and the data transfer times if necessary

slotSchedule(slot  $s$ , task  $n_i$ )
if (numproc( $n_i$ ) > numproc( $s$ )), return failure
if (runtime( $s$ ) < runtime( $n_i, s$ )), return failure
if (starttime( $s$ ) + runtime( $s$ ) < (est( $n_i, s$ ) + runtime( $n_i, s$ ))), return
failure
sort the earliest available time of processors in the slot in ascending
order in a list
term( $p$ ) =  $p^{\text{th}}$  term in the list
ast( $n_i, s$ ) = larger of est( $n_i, s$ ) and term(numproc( $n_i$ ))
if (ast( $n_i, s$ ) + runtime( $n_i, s$ ))  $\leq$  (starttime( $s$ ) + runtime( $s$ ))
return (ast( $n_i, s$ ) + runtime( $n_i, s$ )) as the earliest time when  $n_i$  can
finish on the slot
else
return failure
if actual scheduling is required, update the availability time of the
first numproc( $n_i$ ) processors in the list as (ast( $n_i, s$ ) + runtime( $n_i, s$ ))
EndslotFinishTime

```

Figure 2. Slot based Heterogeneous Earliest Finish Time (HEFT).

MOGA experiments

In order to evaluate MOGA, we created a set of slots representing 4 Grid sites where each is a cluster of 10, 20, 30, and 40 processors respectively. Slots for each site are generated using a Poisson process with a mean slot runtime of 1000 seconds and a mean inter-arrival time of 1000 seconds. For each site, the start time of the first slot is zero. The start times of the other slots equal to the start time of the previous slot plus the inter-arrival time. The number of processors in each slot is randomly generated from 1 to the maximum number of processors on the resource. The set $\hat{A}_{0,t}$ is generated by using $t = 30000$ and $|\hat{A}_{0,t}| = 124$. All the slots have $c_i = 1$, $f_i = 0$, $b_i = \text{false}$, $e_i = \text{false}$.

The application is generated using a parameterized task graph generator. The application consists of 100 tasks with an average runtime of 100 seconds ($< 1000s$ which is the average slot runtime) which ensures that the tasks can be easily scheduled on the slots. The application structure is shown in Figure 3. The tasks with no input dependencies are considered at level 0 and the level of any other task is the maximum level of any of its parents plus one. The application is balanced with 10 levels and 10 tasks at each level on the average. The amount of data transferred between the parent and child tasks is such that the average data transfer time is of the same magnitude as the average runtime of tasks.



Figure 3. Artificially generated workflow with 100 tasks.

We instrumented MOGA to record all feasible solutions encountered. While there were 1742 feasible solutions for this particular problem, only 40 of them were Pareto-optimal (Figure 4). Figure 5 shows the makespan and the allocation cost of the solution selected from the Pareto-optimal set using different trade-off factors (α). The value of the trade-off factor is shown in X-axis while the makespan and allocation cost are plotted against the Y1(left) and Y2(right) axis respectively. Figure 5 shows the behavior of the costs and the combined metric of Equation 9. There is clearly a trade-off between the allocation and scheduling costs. We can see that using the combined metric, we can achieve considerable reduction in the allocation cost for a little increase in the scheduling cost.

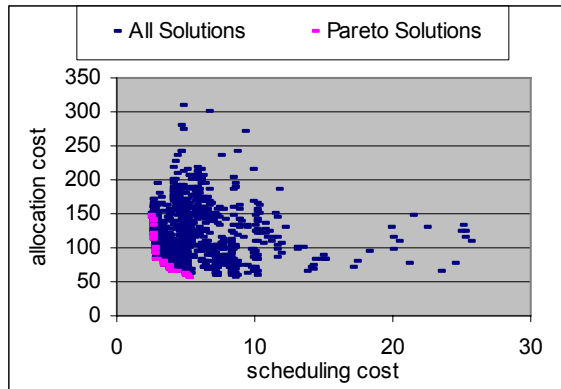


Figure 4. The set of all feasible and Pareto solutions.

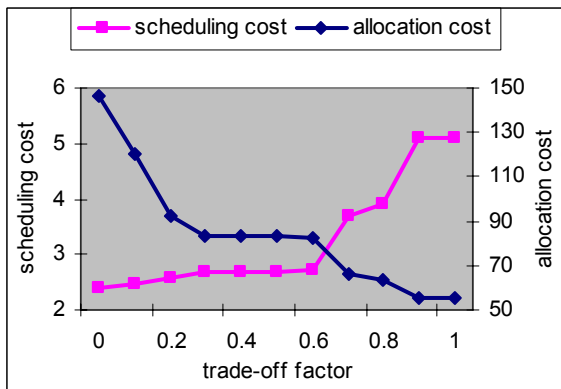


Figure 5. The scheduling and allocation cost of solutions selected from the Pareto set using different α .

We also evaluated three other scheduling algorithms in addition to HEFT. The first one was a low complexity greedy scheduling algorithm that creates a topologically sorted list of tasks in the application. Then it goes through each task in the list, scheduling it on a slot that will minimize the current makespan. The other algorithms are the Min-Min and the Max-Min algorithms as described in [17]. These algorithms partition the application into levels where each level is a set of independent tasks. At each level the Min-Min and the Max-Min algorithms are used to schedule the tasks onto slots. We created the set of Pareto-optimal solutions using each of the scheduling algorithms and then combined them. In the combined set, the dominated solutions were purged, leaving behind 39 non-dominated solutions. Out of these 39 non-dominated solutions, Figure 6 shows the number of solutions found by each of the four scheduling algorithms. Clearly, most of the non-dominated solutions were found using the HEFT algorithm. While all the scheduling algorithms found similar number of solutions, the solutions found by HEFT dominated the solutions found using other scheduling algorithms showing its superiority. HEFT has also been shown to perform better than other myopic or evolutionary based workflow scheduling strategies in Grid environments

[22]. Hence, we use HEFT as the scheduling policy for all the experiments described in this paper.

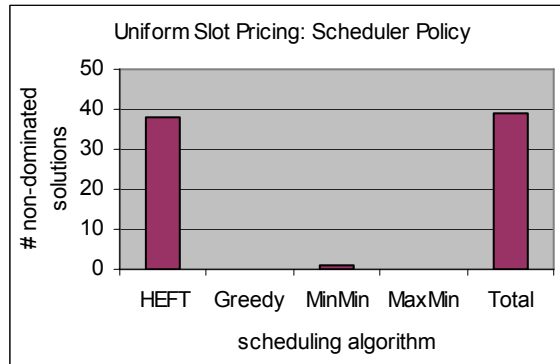


Figure 6. Effect of scheduling policy.

We also performed experiments to determine the effect of different MOGA parameters on the solution quality. The experiments compared the solutions using the domination principle as described previously in the scheduler policy experiment. The population size was found to have a greater effect than the number of iterations. This can be understood by the fact that in the initial population, each member represents a unique solution. All the later solutions are generated from this initial population using different genetic operators and so their solution quality is related somehow to the solution quality of the initial population. Hence, in situations where the runtime complexity of MOGA becomes important, we prefer to have a bigger population size and a smaller number of iterations. Also, we found the 2 point crossover operator to be more effective than a 1 point or a 3 point crossover operator. Similarly, mutation with a mutation probability of $1/n$ was found helpful in finding a diverse set of Pareto-optimal solutions.

4. Design of a Slot Generating Local Scheduler

In this section, we describe extensions that allow a conservative backfilling based scheduler to advertise available resources in the form of free slots. Conservative backfilling is an optimization to the first come first serve scheduling policy that allows a task to be started earlier if it would not delay the start time of any task that has come before this task [23]. When a task is en-queued, the scheduler allocates resources for it i.e. creates a reservation at the earliest possible time. In order for the non-delaying guarantee to work, we do not move the reservations forward in time if a task completes earlier than the specified runtime and tasks are not allowed to execute past their specified runtime either. Thus for all practical purposes, the expiration time of the reservation for the task is considered as the task completion time.

The set of reservations for the currently running and queued tasks represents the site schedule. The scheduler can then determine the free slots in its schedule by creating windows for each processor that shows when the processor is free from the current time to some time in future called the slot horizon. The slot horizon is strictly larger than the end time of any current reservation thereby guaranteeing that each processor has at least one free window. For this paper, we use a slot horizon that in addition, falls on a 24 hour boundary from the current time in order to model availability in multiples of 24 hr periods. Each window is a tuple containing $\langle \text{processor id, start time, end time} \rangle$. Windows across processors having the same start and end time are consolidated into free slots and advertised periodically or on demand. The set of free slots is affected whenever a new task is queued and resources are allocated to the task from those currently available.

In order to illustrate the slot formation process, Figure 7 shows a simple site schedule with three reservations (Resv A, Resv B, Resv C) for submitted tasks A, B, and C respectively. The current time is represented by the leftmost end with the future moving to the right. The slot horizon is at end of time 6 while the maximum scheduled end time of any reservation is time 4 (for task C). Then for each processor, we create windows of free time from the current time to the slot horizon based on the current schedule. The scheduler then consolidates windows with the same start and end time into slots resulting in the six free slots shown in the figure.

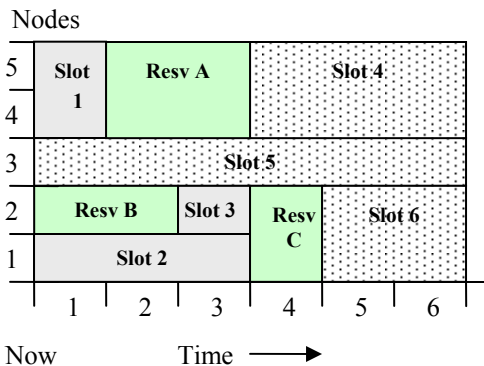


Figure 7. Free slots in the resource schedule.

In order to reduce the time required to create the set of free slots, the scheduler always maintains this set and it is kept sorted by the start time of the slots in an ascending order. This set is updated every time a task is submitted. Indeed, even the conservative backfilling is implemented by scheduling a submitted task to the first slot or set of slots that can execute the task. When the end time of a slot is followed by the start time of a reservation on the processors constituting the slot, the slot is termed indivisible and non-extensible ($b_i = \text{false}$, $e_i = \text{false}$, Section 2). Slot 1, 2, and 3 are indivisible and non-extensibility in Figure 7. When the end time of a slot

coincides with the slot horizon, then the slot is termed divisible and extensible ($b_i = \text{true}$, $e_i = \text{true}$). Such slots are 4, 5, and 6 (Figure 7). By making these slots extensible, the scheduler is able to advertise future resource availability beyond the slot horizon as well.

While extensibility constraints can be easily understood, the divisibility constraints are meant to reduce fragmentation of the resources. The best effort jobs get a better treatment by the scheduler since a best effort job can get backfilled onto an indivisible slot paying only for the resource requested while if the same job were to be run by provisioning the slot, payment for the whole slot would have to be made regardless of usage if the slot were indivisible. However, in return, the best effort jobs are charged for their requested runtime even if they complete earlier. Divisible and extensible slots are more attractive for the provisioned approach but tend to lie farther in the future in this design. All the slots are advertised with a multiplicative cost of 1 and an additive cost of 0.

5. Comparison of Best effort and Provisioned approach using trace simulation

Previously it has been asserted that resource provisioning might or might not improve the application performance while always making it more predictable in a simplistic single machine environment [24]. In this section, we do a more thorough comparison of the application performance (makespan) when the application is executed using the best effort and the provisioned approach. In addition, we also compare the resource costs of the two approaches from the perspective of the user.

We simulated two Grid sites representing a 430 node and a 128 node cluster. The workload simulated on these clusters was based on logs from the 512 node IBM SP2 Cornell Theory Center (CTC) (430 nodes operational) and the 128 node IBM SP2 at the San Diego Supercomputer Center (SDSC) obtained from the Parallel Workloads Archive [25]. The simulator was a modified version of GridSim simulator [26] with extensions for parallel job scheduling and conservative backfilling. Most of the current Grid sites use some sort of backfill-based scheduling policy [27]. The scheduler described in the previous section was used as the local scheduler for these resources. The performance of conservative backfilling has been shown to have similar benefits as the more widely used aggressive backfilling [23] with the added advantage that the job start time can be predicted in advance. Moreover, we use the actual runtime of the tasks as reported in the logs as their requested runtime with the result that the end time of the jobs is also known in advance. The processors on both sites were considered homogeneous for simulation purposes.

Executing an application using the best effort scheduling on the sites was done by simulating a just in time scheduling policy. Each task in the application was submitted to the resource queue when it became ready for execution i.e. all its parent tasks finished execution. The time difference between the time when the first task in the application was submitted to the resource queue and the time when the last task finished execution is taken as the best effort makespan of the application. Each application was executed on a single site only and the experiments were repeated for both sites.

The allocation cost of best effort execution is the sum of the runtime of the task multiplied by the number of processors required by the task over all the tasks in the application.

$$AC_{best-effort} = \sum_V v_{it} \cdot v_{ip} \quad (\text{Eq 10})$$

V = set of all tasks in the application (v_1, \dots, v_n)

v_{it} = runtime of a task v_i

v_{ip} = number of processors required by task v_i

This allocation cost reflects the fact that in best effort execution, as per the current practice, one only pays for the resources used by the task during its execution. Thus the allocation cost for an application is constant for best effort execution and is the lower bound on the allocation cost of the provisioned approach.

For the experiments, we use the workflow shown in Figure 3. The average runtime of the tasks is 1000 seconds. Since each application is executed on a single site, the data transfer time between the tasks is zero. For the SDSC cluster, the average number of processors per task in the application is 6 while for the CTC cluster, the average number of processors per task is 22, making the average width of the application, half of the maximum number of processors on the resource.

We further modified the HEFT algorithm to do cross slot scheduling. This implies that if a slot does not have enough processors to schedule a task, then HEFT would consider combinations of slots that overlap in time to schedule the task. Furthermore, it uses the extensibility attributes of slots to extend them for scheduling tasks with longer runtimes than the specified slot duration. After the schedule is completed, for divisible slots, we allocate only those parts of the slots where tasks are scheduled leading to lower allocation costs.

The client queries the set of free slots (using $t = \infty$) from the site being simulated when the application is submitted (usually a week or more into the simulation time) and generates a resource plan using MOGA and a given trade-off factor α . We compute and record the makespan and the allocation cost of this resource plan. The application is then executed using best effort. The allocation cost of the best effort approach is always

constant for an application as mentioned earlier. This process is repeated 50 times by submitting the application at different times during the simulation run and the average is taken. All the experiments were performed for both sites.

Figure 9 shows the makespan and the allocation cost of the best effort and the provisioned approach on the SDSC and CTC cluster. The results of the provisioned approach are shown for different values of the trade-off factor α . For the SDSC cluster, the makespan of the provisioned approach is considerably less than that of the best effort approach for lower values of the trade-off factor α at the cost of an increased allocation cost. For the CTC cluster, however, the provisioned makespan is lower than the best effort makespan for all values of α at a cost of little or no increase in the allocation cost. This can be explained by the lower utilization of the CTC cluster (64%) as compared to the SDSC cluster (74%) which provides more free slots to schedule the application on. While the best effort allocation cost remains a lower bound on the provisioned makespan, MOGA is able to approach that lower bound using higher values of α . The lower makespan in general of the provisioned approach is due to the fact that each best effort task experiences some queue wait time at the sites based on the current workload and the child tasks are not released for submission until the parent tasks have finished execution. With the provisioned approach, resources are provisioned for the entire application ahead of execution leading to lower makespans in general.

Figure 9 also shows the standard deviation of the makespan and the allocation cost. The standard deviation of the makespan using the best effort is generally higher than that of the provisioned approach. This proves that the provisioned approach provides better insulation for the application against dynamic changes in the resource workload. As the value of α increases, the standard deviation of the provisioned makespan increases and the allocation cost decreases due to the increased emphasis on reducing the allocation cost. The allocation cost of the best effort is a constant and hence its deviation is zero.

We also performed experiments to determine the effect of the task size on the makespan and the allocation cost. The task size is varied by changing the average number of processors per task in the workflow while the workflow structure remains the same. Figure 10 shows the effect of the task size on the makespan and the allocation cost of the best effort and the provisioned approach on the SDSC and the CTC clusters. The performance of the provisioned approach is shown using tradeoff factors of zero and one that represents the lower and upper bounds for the makespan and the allocation cost of the provisioned approach. The figure shows the provisioned makespan and allocation cost in percentage terms relative to the best effort, which is considered 100 percent. For both the clusters, as the task size increases, the application performance in the provisioned approach relative to the

best effort improves. As the task size increase, there is no significant effect on the allocation cost since most of the tasks are scheduled on the divisible slots or their combinations due to the large task size. With larger task sizes, the chances of the best effort scheduler being able to backfill them earlier in the schedule decreases and their queue wait time increases. This shows that the provisioned approach is well suited for workflows with highly parallel tasks.

The production Grid sites such as Teragrid [1] etc often experience high to very high utilization levels [16]. We performed experiments to ascertain the effect of the resource utilization on the performance of the provisioned approach. These experiments were done only for the SDSC cluster though we expect the results to be similar for the CTC cluster. For increasing the load, we made two copies of the workload trace using a method adopted in [28]. The first trace started on midnight 1st May 1998 and the second one on midnight 29th May 98 (exactly four weeks later). Just simulating the first trace gave a resource utilization of 74%. For increasing the load, both the traces were fed simultaneously to the simulator. While all tasks in the first trace were submitted to the resource as per their recorded submission time, the tasks in the second trace was submitted to the resource only with a certain probability. Increasing this probability allowed us to increase the load while not changing the task characteristics in the workload.

Figure 11 shows the allocation cost and the makespan of the provisioned approach as a percentage of the best effort, which is plotted as 100 percent. In this case, a single tradeoff factor of 0.5 is used for generating the provisioned results. The results are shown for different task sizes and different resource utilization levels. Except for the case where the application is composed of uniprocessor tasks, the provisioned makespan becomes smaller as compared to the best effort as the resource utilization increases. The reason is that due to the increased load, the queue wait times of the best effort tasks increase leading to an increased makespan. The allocation cost of the provisioned approach is not significantly affected. This shows that the provisioned approach is more attractive when the resources are highly loaded and the applications have significant resource requirements.

6. Comparison of the best effort and provisioned performance of a seismic hazard analysis application

In the previous sections, we compared the performance of the provisioned and the best effort approaches using artificially-generated applications. In this section, we use a seismic hazard analysis application taken from the earthquake engineering community, called CyberShake [29]. The workflow has 5 levels with 8039 tasks and the structure of the workflow along with the

module names is shown in Figure 8. Table 2 **Error! Reference source not found.** shows the average runtime and number of processors required for each module. Levels four and five in the workflow contain 4017 tasks each with the same module (synthSGT and peakValCal respectively) operating on different datasets. Since the parallel tasks at level 2 and 3 in the workflow (shown in gray) would not be able to execute on the SDSC cluster, we simulated the execution of this workflow on the CTC cluster only. The runtimes of the tasks in the workflow were acquired from the provenance records of the previous runs of the workflow on the HPCC (High Performance Computing & Communications) cluster at the University of Southern California (USC).

Table 2. Module details of the CyberShake workflow.

Module name	# of tasks	Avg runtime (seconds)	# of processors
fd_grid_xyz	1	1	1
preSGT	1	300	1
fd_grid_cvm	1	2100	288
pmvl_chk1	1	86400	288
pmvl_chk2	1	86400	288
synthSGT	4017	519	1
peakValCal	4017	1	1

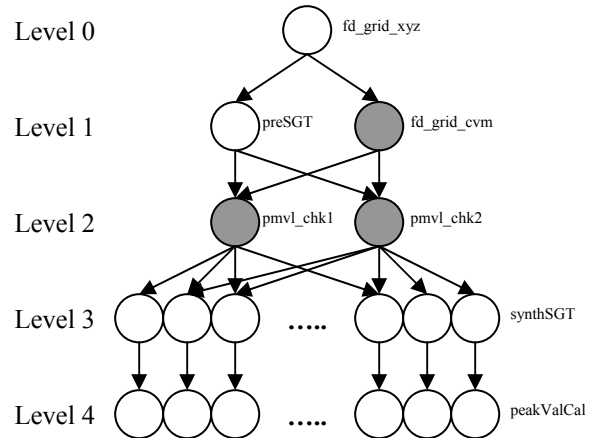


Figure 8. The seismic hazard analysis workflow.

We simulated different background loads on the CTC cluster by making two copies of the workload and probabilistically adding portions of the second copy to the simulated workload as described in the previous section. As a result, we were able to vary the resource utilization of the CTC cluster from 63% to 94%. For each utilization level, we simulated the workflow execution using both best effort and the provisioned approach at different points during the simulation run and averaged the makespan and allocation cost metrics. We used a trade-off factor of 0.5 for creating the resource plan.

Figure 12 show the makespan and the allocation cost of the workflow execution using the best effort and the provisioned approach. At 63% utilization, the best effort makespan is 476811 seconds (5.5 days) which is similar in

magnitude to the observed makespan of the workflow on the HPCC cluster. The provisioned makespan at this utilization level is 23% lower than the best effort value. The best effort makespan sharply increases as the resource utilization exceeds 80%. The growth in the provisioned makespan is less pronounced. At 94% utilization level, the provisioned makespan is 56% lower than the best effort makespan. **Figure 12** also shows the standard deviation of the makespan and allocation cost. The standard deviation of the best effort makespan is more than the provisioned makespan and increases with the utilization of the resource. The standard deviation of the provisioned allocation cost is very small at the highest utilization level and negligible at lower levels.

The allocation cost of the best effort and provisioned approach does not differ significantly as the resource utilization level is changed. The reason is that with the provisioned approach, due to the large size of the parallel tasks, these and the later parts of the workflow had to be scheduled on the divisible/extensible slots at the end of the resource schedule which are very cost effective. In practice, special reservations had to be set up for these tasks on the HPCC cluster since the best effort queues were not configured for such large tasks.

7. Discussion

The makespan using best effort was generally less than the makespan of the provisioned approach in our evaluation due to the fact that the Grid scheduler was not capable of handling dependencies and hence the child tasks has to be submitted only after the parent tasks had finished execution. Some resource management systems such as PBS, Maui, Condor, LSF etc do allow the user to specify dependencies between submitted tasks. Depending on the system used, these tasks might or might not gain priority in queue while waiting for their dependencies to be satisfied. For example, PBS when used standalone allows such tasks to gain priority but when used with the Maui scheduler (which is the most common case), it is generally configured otherwise. The dependency manager in Condor (DAGMan) also waits for the completion of parent tasks before submitting child tasks to resource queue. Any production system must disallow such tasks with unsatisfied dependencies from gaining priority while waiting in order to prevent system abuse. Abuse can occur when a single user submits a large scale workflow in its entirety thus dominating the resource allocation for the timeframe of the execution of the workflow, preventing other users from gaining their fair share of the resource. Moreover, since dependencies are only recognized between the tasks submitted to the same resource queue, this mechanism has limited use for workflows executed using multiple resources.

One major concern with the use of Genetic Algorithms is its run time complexity. In our case, the complexity derives from the fact that in order to compute the fitness of an individual (resource plan), a schedule of the application has to be created. This fitness computation has to be done for each individual in the population at each iteration. Thus the total complexity is roughly equal to the scheduling complexity times the population size times the number of iterations. While the scheduling complexity is fixed, the runtime complexity of the GA can be controlled by reducing the population size and the number of iterations. For example in the CyberShake workflow with 8039 tasks (Section 6) in the 63% utilization case, the average time taken by MOGA to compute the pareto set was 5.6 minutes with a population size of 10 and 10 iterations on a 2 GHz Pentium 4 machine. Moreover, this runtime cost is negligible as compared to the reduction in makespan due to provisioning which was 30.6 hours in this case.

In Section 4, we have assumed that the tasks report their actual runtimes so that the end times of these tasks are accurately known to the scheduler. However, in practice users generally quote task run times that may be significantly more than the actual run times. In a provisioning based environment, these loose runtime estimates can adversely impact the performance perceived by the provisioned users since the free slot calculation is based on these task runtime estimates. For example, increasing the runtimes of the tasks in the workload by 10% and 20%, the provisioned makespan increased by 3% and 7.7% respectively in the case of CyberShake workflow with 63% resource utilization (it was still less than best effort makespan which was 30% more than the provisioned makespan). Thus the resource providers might decide to charge best effort users based on their runtime estimates or charge penalties for wrong estimates which would encourage users to provide tight runtime estimates.

The multi-objective GA formulation used in this paper can be used to consider other performance metrics in addition to or instead of the application makespan. However, the size of the Pareto-optimal set generally increases with the number of objectives. In addition, the GA formulation becomes computationally expensive as the number of tasks in the application increase. This is due to the fact we that we have to create a complete schedule of the application on each individual in the population to gauge its fitness (allocation and scheduling cost). For applications with large number of tasks, we might have to consider additional measures such as partitioning the application and overlapping the planning with the execution of the application.

The increased allocation cost of the provisioned approach as compared to best effort is a fallout of the restriction that the non-divisible slots have to be provisioned in its entirety irrespective of how it is going to

be ultimately used. Thus the client has to pay for the whole slot even if only a part of it is going to be used. By requiring that slots be provisioned as a whole, the resource provider increases its revenue by passing on this fragmentation cost to the user as the allocation cost. However, this over-provisioning of resources by the broker can become useful when the application tasks exceed their expected resource requirements and need more resources to execute or multiple applications have to be executed concurrently.

The advertisement of a slot is advisory and does not guarantee that the slot will actually be available should an application decide to allocate it. Actual allocation of the slot will be subject to policy enforcement by the resource provider at the point the allocation request is made. However, the model adopted in this paper leads to fewer interactions between the resource brokers and providers than other negotiation based approaches that provision resources for each application task separately [30-32]. Considering the large-scale nature of Grid applications, the multitude of users and resource providers, it is essential to make the provisioning process fast and efficient. Moreover, mechanisms such as the SNAP based three phase commit protocol [33] may be used to converge to a resource plan quickly using retries when contention between users or policy issues make some desirable slots available to the user.

In this paper, we have restricted our attention to provisioning of compute resources. However, this framework can be extended to consider other types of resources also. In the described experiments, provisioning of network resources was not considered since each application was executed on a single cluster. Provisioning of network resources would become important when co-allocating resources from multiple providers. Design of a generic slot based resource manager that can be used to manage network resources is discussed in [34]. The idea is to manage allocations using a reservation table similar to the site scheduler in Section 4 while doing the actual reservation using the RSVP [35] protocol. Hard guarantees can be obtained if the resource manager does admission control as well.

8. Related Work

There is a large body of research on application scheduling on dedicated systems [19, 36]. The resource provisioning is implicit in that the entire resource is considered provisioned. In Grid computing, due to the non deterministic nature of the resource availability, prediction services such as the Network Weather Service [37], queue wait time estimators [38, 39], etc are used to make scheduling decisions for the applications [40-42]. However, the resulting application performance is highly dependent on the quality of the predictions. Moreover, frequent

adaptation is required for countering the dynamic nature of resource availability.

Advance reservations have been widely proposed for provisioning resources for performance predictability, meeting resource requirements and providing guaranteed quality of service to applications [24, 30, 31, 43, 44]. A theoretical proof that reservations improve the performance predictability of applications in a simplistic scenario is presented in [24]. A resource model similar to ours is adopted in [43] where a client can probe the possible start times of a task on a resource. However, the reservation is for a single job instead of an application and a single resource slot has to be reserved. This has been extended in [44] to support co-reservations. However, the reservations are made based on an abstract resource description from the user. In our case, we create the set of reservations based on the resource availability in the Grid and a given application. Additionally it creates all combinations of the resource slots to find a feasible candidate. This might not be a feasible strategy if there are large numbers of available resource slots.

In [31], authors create a reservation plan for a workflow by making reservations for each task individually. This may not be a feasible approach for large scale workflows containing thousands of fine-grained tasks. The focus in [31] is on increasing reliability of execution of the workflow and contention for resources is not considered. A similar strategy is adopted in [30, 45] where reservation or allocation for each task or activity in the application is done separately by negotiating with the resource providers. The cost of allocation is not considered in [30, 31]. In [45], authors employ a cost aware resource model similar to ours. However, the goal here is to concurrently maximize the resource and application utility using a centralized resource allocator. In our case, we use a distributed approach where the resource providers and users maximize their own utility.

Genetic algorithms have been widely used for application scheduling on heterogeneous distributed systems [46-49]. However, in almost all the cases, the GA is used for scheduling or mapping the application whereas in our case, the GA is used for resource allocation and the scheduling is done using HEFT [19]. Moreover, even in cases, where multiple objectives are to be optimized, a weighted combination of the various objectives is used as a single objective to guide the search [47]. We do not use any weights or preferences during the GA search and the preference is specified only when the entire Pareto set has been created.

There has been a recent focus on the framework [11, 50] and protocols for agreement based resource management [33, 51, 52]. These provide the underlying plumbing required to instantiate a set of resource agreements for an application and complement the work

presented in this paper. However, there has been little work on the reasoning process for selecting the set of agreements to be created for workflow structured applications.

9. Conclusion

In this paper, we have presented a multi-objective GA formulation for provisioning resources for an application using a slot-based resource model for optimizing the application performance and minimizing the provisioning cost. We have extended a conservative backfilling-based Grid scheduler to support resource provisioning in addition to providing best effort quality of service. Using a trace-based simulation and an artificial and a real application, we have compared the application performance using the best effort and provisioned approach. We evaluated the sensitivity of the provisioned approach to changes in the user preference (trade-off factor), application task size and resource utilization. The results show that the provisioned approach outperforms the best effort execution when the size of the tasks in the application increase and the background load on the resources increase.

In future, we plan to experiment with an advance discount pricing strategy and investigate the effect on the application and system performance. Furthermore, we plan to include the provisioning of storage and network resources in our provisioning model.

10. Acknowledgements

The authors and research described here are supported in part by the National Science Foundation under NSF Cooperative Agreement NSF CCR-0331645, NSF NGS-0305390.”””

11. References

1. Catlett, C. *The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility*. in *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*. 2002.
2. *The Open Science Grid Consortium*, <http://www.opensciencegrid.org>.
3. Katz, D.S., et al. *A Comparison of Two Methods for Building Astronomical Image Mosaics on a Grid*. in *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*. 2005.
4. Deelman, E., et al. *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists*. in *11th Intl Symposium on High Performance Distributed Computing*. 2002.
5. Maechling, P., et al., *Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment* SIGMOD Rec. , 2005 **34** (3): p. 24-30
6. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
7. Droegeemeier, K.K., et al. *Linked Environments for Atmospheric Discovery (LEAD): A CyberInfrastructure for Mesoscale Meteorology Research and Education*. in *20th Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*. 2004. Seattle, WA.
8. Henderson, R.L., *Job Scheduling Under the Portable Batch System* in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* 1995 Springer-Verlag. p. 279-294
9. Zhou, S., et al., *Utopia: a load sharing facility for large, heterogeneous distributed computer systems* Softw. Pract. Exper. , 1993 **23** (12): p. 1305-1336
10. Litzkow, M.J., M. Livny, and M.W. Mutka. *Condor-a hunter of idle workstations*. in *Distributed Computing Systems, 1988., 8th International Conference on*. 1988.
11. Czajkowski, K., I. Foster, and C. Kesselman, *Agreement-based resource management*. Proceedings of the IEEE, 2005. **93**(3): p. 631-643.
12. Netessine, S. and R. Shumsky, *Introduction to the Theory and Practice of Yield Management*. INFORMS Transactions on Education, 2002. **3**(1): p. 34-44.
13. Gale, I.L. and T.J. Holmes, *Advance-Purchase Discounts and Monopoly Allocation of Capacity*. American Economic Review, 1993. **83**(1): p. 135-146.
14. Singh, G., C. Kesselman, and E. Deelman. *Application-level resource provisioning on the Grid*. in *2nd IEEE conference on e-Science and Grid computing*. 2006. Amsterdam.
15. Fonseca, C.M. and P.J. Fleming. *Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization*. in *Proceedings of the Fifth International Conference on Genetic Algorithms*. 1993.
16. Iosup, A., et al. *How are Real Grids Used? The Analysis of Four Grid Traces and its Implications*. in *7th IEEE/ACM International Conference on Grid Computing*. 2006. Barcelona, Spain.
17. Mandal, A., et al. *Scheduling Strategies for Mapping Application Workflows onto the Grid*. in *The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*. 2005.
18. Smith, W., I.T. Foster, and V.E. Taylor, *Predicting Application Run Times Using Historical Information* in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* 1998 Springer-Verlag. p. 122-142
19. Topcuoglu, H., S. Hariri, and M.-y. Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing* IEEE Trans. Parallel Distrib. Syst. , 2002 **13** (3): p. 260-274
20. Deb, K., *Multi-Objective Optimization using Evolutionary Algorithms*. 2001: John Wiley & Sons.
21. Coello, C.A.C., *A Comprehensive Survey of Evolutionary-Based MultiObjective Optimization Techniques*. Knowledge and Information Systems, 1999. **1**(3): p. 269-308.
22. Wiecezorek, M., R. Prodan, and T. Fahringer, *Scheduling of scientific workflows in the ASKALON grid environment* SIGMOD Rec. , 2005 **34** (3): p. 56-62

23. Feitelson, D.G. and A.M. Weil. *Utilization and predictability in scheduling the IBM SP2 with backfilling*. in *Parallel Processing Symposium, 1998. 1998 IPPS/SPDP. Proceedings of the First Merged International...and Symposium on Parallel and Distributed Processing 1998*. 1998.
24. McGough, A.S., et al., *Making the Grid Predictable through Reservations and Performance Modelling*. The Computer Journal, 2005. **48**(3).
25. Feitelson, D.G., *Logs of real parallel workloads from production systems*, in URL: <http://www.cs.huji.ac.il/labs/parallel/workload>.
26. Buyya, R. 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, 2002. **14**(13-15): p. 1175-1220.
27. Feitelson, D.G., L. Rudolph, and U. Schwiegelshohn, *Parallel Job Scheduling --- {A} Status Report*, in *Job Scheduling Strategies for Parallel Processing*, D.G.F.a.L.R.a.U. Schwiegelshohn, Editor. 2004, Springer Verlag. p. 1--16.
28. Sabin, G., V. Sahasrabudhe, and P. Sadayappan. *Assessment and enhancement of meta-schedulers for multi-site job sharing*. in *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*. 2005.
29. Deelman, E., et al. *Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example*. in *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on*. 2006.
30. Wiczczyk, M., et al. *Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid*. in *Second IEEE International Conference on e-Science and Grid Computing*. 2006. Amsterdam.
31. Zhao, H. and R. Sakellariou. *Advance Reservation Policies for Workflows*. in *12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2006. Saint-Malo, France.
32. Li, J. and R. Yahyapour. *Negotiation Model Supporting Co-Allocation for Grid Scheduling*. in *7th IEEE/ACM International Conference on Grid Computing*. 2006. Barcelona, Spain.
33. Haji, M.H., et al., *A SNAP-Based Community Resource Broker Using a Three-Phase Commit Protocol: A Performance Study*. The Computer Journal, 2005. **48**(3): p. 333-346.
34. Foster, I., et al. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. in *Proc. International Workshop on Quality of Service*. 1999.
35. Zhang, L., et al., *RSVP: a new resource ReSerVation Protocol*. Network, IEEE, 1993. **7**(5): p. 8-18.
36. Kwok, Y.-K. and I. Ahmad, *Static scheduling algorithms for allocating directed task graphs to multiprocessors* ACM Comput. Surv. , 1999 **31** (4): p. 406-471
37. Wolski, R., N. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems, 1999. **15**(5-6): p. 757-768.
38. Brevik, J., D. Nurmi, and R. Wolski. *Predicting bounds on queuing delay for batch-scheduled parallel machines*. in *ACM Symposium on Principles and Practice of Parallel Programming*. 2006. New York.
39. Downey, A.B. *Predicting queue times on space-sharing parallel computers*. in *Proceedings of the 11th International Parallel Processing Symposium*. 1997.
40. Nurmi, D., et al. *Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction*. in *SuperComputing Conference*. 2006. Tampa, Florida.
41. Berman, F., et al., *Adaptive computing on the Grid using AppLeS*. Parallel and Distributed Systems, IEEE Transactions on, 2003. **14**(4): p. 369-382.
42. Casanova, H., et al. *Heuristics for scheduling parameter sweep applications in grid environments*. in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*. 2000.
43. Roblitz, T., F. Schintke, and J. Wendler. *Elastic Grid Reservations with User-Defined Optimization Policies*. in *Proceedings of the Workshop on Adaptive Grid Middleware*. 2004.
44. Roblitz, T. and A. Reinefeld. *Co-reservation with the concept of virtual resources*. in *IEEE International Symposium on Cluster Computing and the Grid*. 2005.
45. Siddiqui, M., A. Villazon, and T. Fahringer. *Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS*. in *SuperComputing Conference*. 2006. Tampa, Florida.
46. Daoud, M.I. and N. Kharmia. *An Efficient Genetic Algorithm for Task Scheduling in Heterogeneous Distributed Computing Systems*. in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. 2006.
47. Dogan, A. and F. Ozguner, *Biobjective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems*. The Computer Journal, 2005. **48**(3): p. 300-314.
48. Yu, J. and R. Buyya, *Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms*. Scientific Programming Journal, to appear.
49. Prodan, R. and T. Fahringer. *Dynamic scheduling of scientific workflow applications on the grid: a case study*. in *Proceedings of the ACM symposium on Applied Computing*. 2005. Santa Fe, New Mexico.
50. *The Grid Resource Allocation and Agreement Protocol Working Group*, in <https://forge.gridforum.org/projects/graap-wg>.
51. Andreozzi, S., et al. *Agreement-Based Workload and Resource Management*. in *First International Conference on e-Science and Grid Computing*. 2005.
52. Kuo, D. and M. Mckeown. *Advance Reservation and Co-Allocation Protocol for Grid Computing*. in *First International Conference on e-Science and Grid Computing*. 2005.

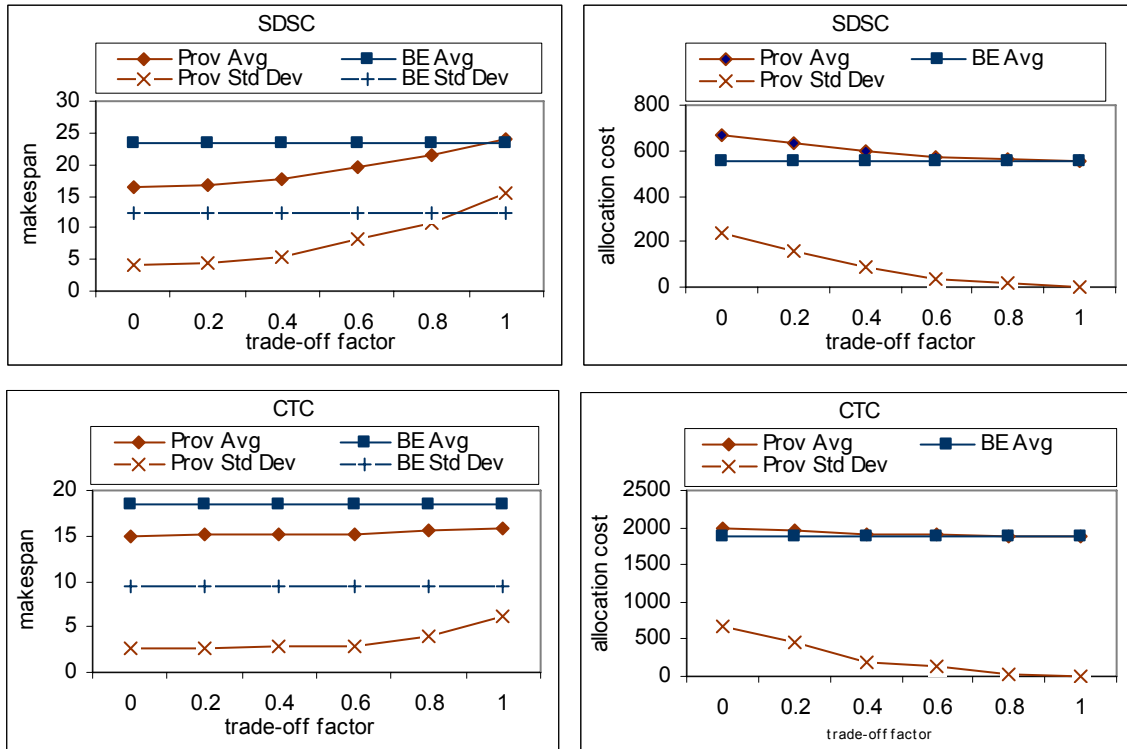


Figure 9. Allocation cost and makespan using best effort and the provisioned approach for different trade-off factors.

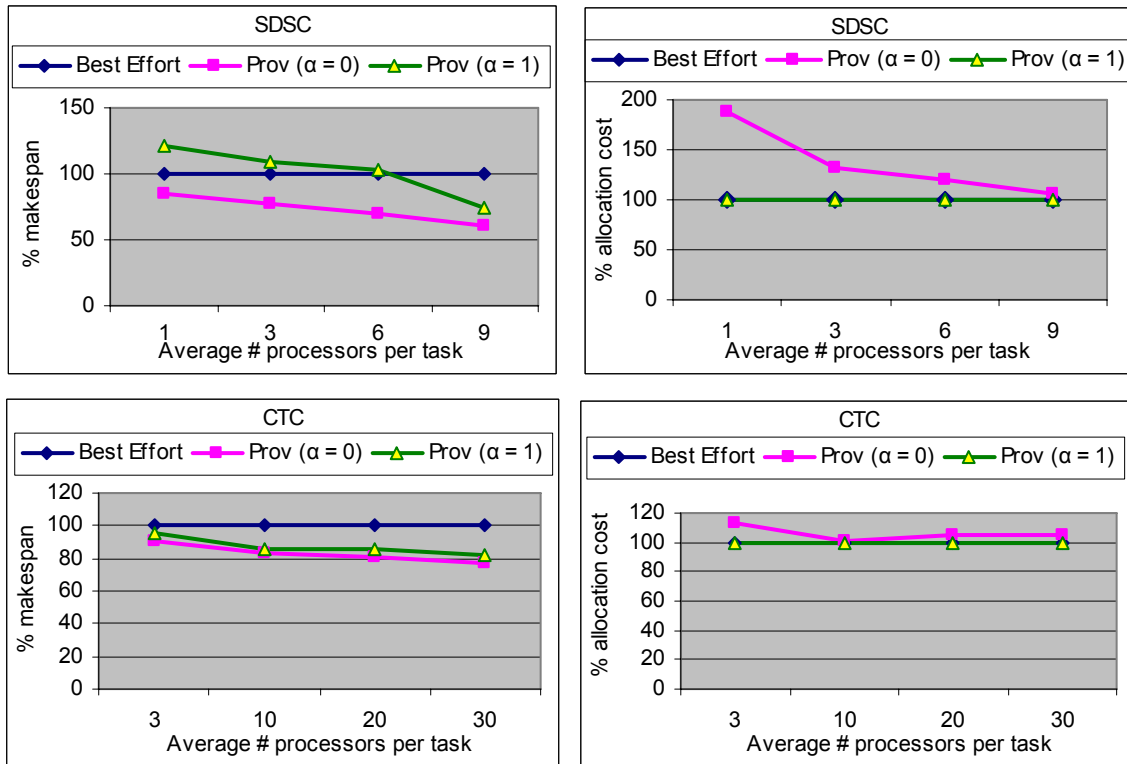


Figure 10. Allocation cost and makespan using the best effort and the provisioned approach for varying task sizes.

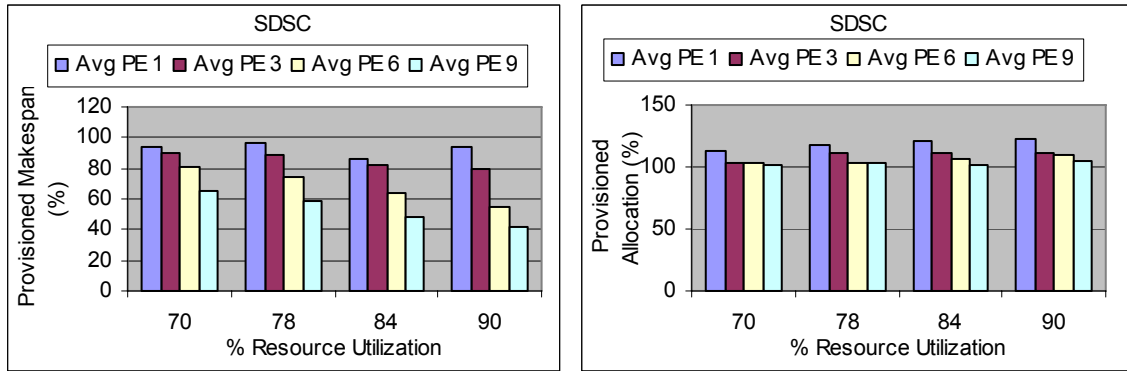


Figure 11. Allocation cost and makespan (as % of best effort values) of the provisioned approach with increasing resource utilization.

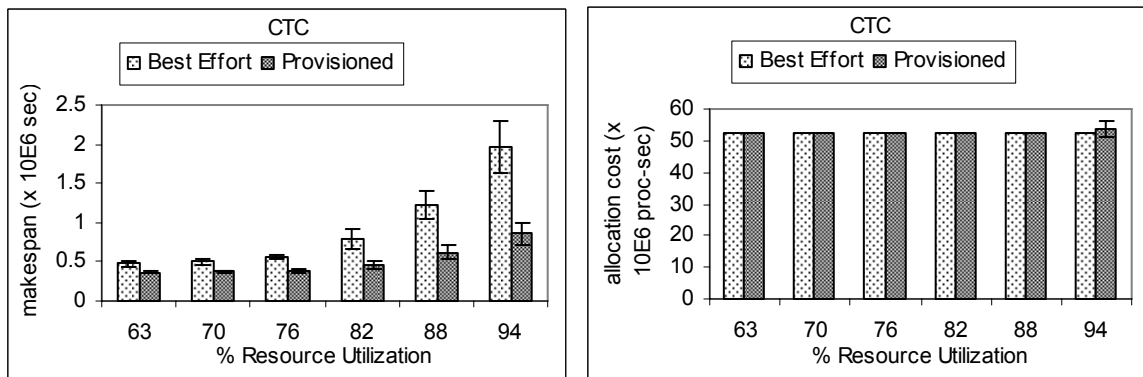


Figure 12. Allocation cost and makespan of the seismic hazard analysis workflow on the CTC cluster with the best effort and the provisioned strategy.