A Novel Job Scheduler for Computational Grid Using Simulated Annealing Heuristic

Hamid Saadi [a,*], AbdolHamid MomenZadeh[b], Ehsan PourAliAkbar[c], Mohsen ShakibaFakhr[d], Mohammad GhanbariAdivi[e]

[a,*] Corresponding Author: Department of Computer Engineering, Masjed-Soleiman Branch, Islamic Azad University, Masjed-Soleiman, Iran.
[b,c] Department of Computer Engineering, Masjed-Soleiman Branch, Islamic Azad University, Masjed-Soleiman, Iran.
[d,e] Sama Technical and Vocational Training College, Islamic Azad University, Shoushtar Branch, Shoushtar, Iran

***Abstract*—Computational Grids enable the coordinated and aggregated use of geographically distributed resources for solving large-scale problems in science, engineering, and commerce. Achieving high performance in a grid system requires effective resource scheduling. The heterogeneous and dynamic nature of the grid, as well as the differing demands of applications running on the grid makes grid scheduling complicated. Also, the execution cost, besides the completion time, has become the great concern to the grid users. Many of grid scheduling systems optimize completion time and execution cost separately. In this paper, a novel scheduling algorithm based on simulated annealing heuristic which considers both the completion time and execution cost is introduced. The proposed model applies a weighted objective function that takes into account both the completion time and execution cost of the tasks. The results obtained from our algorithm have been compared with several algorithms such as random, best of N random and climb search algorithm according to the criteria of completion time and execution cost. We show that the proposed SA scheduler produces a comparatively better result in the case of both time and cost optimization.**

***Keywords*-Computational grid; scheduling; Optimization; simulated annealing.**

I. Introduction

Computational Grids enable the coordinated and aggregated use of geographically distributed resources, often owned by autonomous organizations, for solving large-scale problems in science, engineering, and commerce. Grid resources in different geographic places and by different organizations and are managed under different policies [1]. Today, increasing the efficiency of grid is a problem. To increase the efficiency of Grid, a correct and efficient scheduling is necessary. Unfortunately, the dynamic nature of Grid resources and demands of different users, have increased the complexity of the grid scheduling problem (GSP). The dynamics of resource's efficiency is due to heterogeneous, share and the self-determination of Grid resources. The goal of GSP is optimize the appointment of tasks to resources. Study of grid scheduling is important because the real world systems usually are physical or operational distributed and heterogeneous. Such as air traffic control systems, astronomy, medicine, biology, military and mobile.

In 2001 Buyya, introduced a framework for economic grid [4]. In this method, the grid users pay the owners per resource use. This context motivates the owners to share their resources. Buyya economic model augment the importance of economic costs and benefits for resource owners and users. So those then, most scheduling algorithms, placed cost and economic benefits in the Schedule function to satisfy resource users/owners [9]. But the problem is here that these algorithms apply various objective functions and policies to improve the completion time or the execution cost separately.

For improving grid scheduling, heuristics algorithms inspired by natural rules, have been proposed [2, 5, 9]. These algorithms include genetic algorithms, Simulated Annealing (SA), Tabu Search, game theory and combination methods. In [9] proved that SA solution improved both time and cost better than game theory. The reason is the limitations of non-participation game theory algorithm. In [2] showed that the speed of implementation of SA is more than all the above algorithms. References [2, 5] apply SA algorithm only for improving completion time of jobs. Of course, [9] apply SA algorithm to improve the completion time and execution cost of jobs separately. Neither of [2, 5, 9] did not provide a method when the importance (weight) of the cost and time vary for the user. Another problem is that the above algorithms apply random mapping tasks to resources in their basic solution.

Because cheapness and fastness are two conflicting factors, or in other words usually quick resource is expensive, the scheduler should impose user comments in appointing jobs to resources. For example, if the user is concerned about the execution cost of his job, job's tasks should be appointed on a cheap resource, and if the user is concerned about the completion time of his job, job's tasks should be appointed on a quick resource.

The presented Scheduler gives more freedom to user for applying his concern. Thus the user might 70% worried about execution cost and 30% worried about the completion time of his jobs[10]. We modeled the weight of time and cost using two numbers (wt, wc). These weights are between 0 and 1 and their total is 1. These weights are specified by user and have effect on the objective function of the Scheduler. Due to be NP-complete problem in Grid Scheduling, we apply non-deterministic SA algorithm for optimizing our scheduler. In SA algorithm two factors, time and cost, with their weights are placed in matching[11]. The initial solution in SA proposed, is a nearly optimal solution and is not a random solution. Finally, we compare the proposed algorithm with three algorithms that one of them randomly allocated resources and second use the Best of N method and third use climb search method. The results of this comparison that shows the efficiency of proposed algorithm come in Section V.

The organization of the paper is as follows. In section II the scheduling of grid computing is discussed. In section III simulated annealing method is discussed. In section IV proposed grid scheduling algorithm is introduced. We make some simulation tests and conclude this studying in section V and VI.

II. Grid Scheduling

A grid resource management system controls resource to achieve the goals of a high performance grid system. A grid scheduler is the part of a grid resource management system that uses grid system information (GIS) and job information to produce an assignment of jobs to resources for a given grid job. The assignment decision is known as task allocation or task placement, and the assignment itself is called a schedule. Effective task placement decisions produce schedules that attempt to minimize execution cost and completion time of a job.

The grid scheduling problem is the problem of assigning a set of tasks to a set of machines, or nodes, or resources. We consider a set of k tasks T = {t1, t2, …, tk} and a set of N nodes N={h1, h2, …, hn}. The set T includes the total job to be scheduled, and has been previously partitioned into k tasks by the user, the application writer, or another partitioning tool. The scheduling framework produces some mapping of tasks to nodes ASSIGNMENT = {(t1, hx),…, (tk, hy)}. Each of the k elements in ASSIGNMENT maps each unique task in T to any node in N, so multiple tasks can be assigned to the same node (see Figure 3).

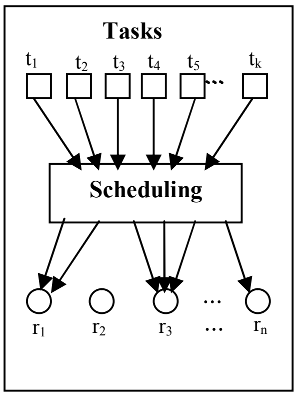


Figure 1. scheduling is Assigning a set of tasks to resources

III. Simulated Annealing

SA is an iterative search method inspired by the annealing of metals. The process can be described as follows. Firstly, a solid is heated from a high temperature and then cooled slowly so that the system at any time is approximately in thermodynamic equilibrium. At equilibrium, there may be many configurations with each one corresponding to a specific energy level. The chance of accepting a change from the current configuration to a new configuration is related to the difference in energy between the two states. Kirkpatrick et al. were the first to introduce SA to optimization problems in 1983 [6]. Since then, SA has been widely used in combinatorial optimization problems and has achieved good results on a variety of problem instances. The key objective of this paper is to find an effective assignment of user's tasks on resources that satisfies his concern about cost and time.

Suppose Fc and Fn are current and new solution's merit. If Fn<Fc then the new solution will replace the current solution, otherwise the new solution will replace the current solution with probability Exp(-(Fn-Fc)/T). In this probability, T is the temperature of algorithm. Therefore possibility acceptance of a worse solution is related to T, Therefore possibility acceptance of a worse solution is above when T is high and less when T is low. The reason of accepting worse solution is escape from local maximum and minimum. Due to the gradual reduction of temperature and acceptance of better solutions in lower temperatures, the algorithm will converge toward the best solution [6].

In any temperature, the best configuration so far seen and its merit are reserved. If Fn<Fb then the new solution is accepted as the best solution. With gradual decrease of temperature, new solutions may be achieved until further improvement is not possible. Finally, the best solution is the SA algorithm output.

IV. Proposed Model

In this section a new Scheduler for grid tasks with optimizing time and cost based on weights specified by the user, has been proposed. We apply SA algorithm for the optimization.

A. *Objective function for new scheduler*

As we say above some of the users concerned and worried about the cost and some other worried about the time. Therefore, in the proposed method, the user can determine weights for the time and cost (Wt, Wc) when he/she submitting a job. These two weights are in the range of [0,1] and their total is 1. For example, if for one job Wc=0.7 means that the user is 70% worried about the financial cost of implementation and 30% worried about the completion time of his job. So the scheduler must find resources for implementing his/her tasks that he/she has 70% improvement in cost and 30% improvement in time. Usually more users concerned about the cost of executing their jobs.

Schedulers almost use Performance Fitness Function (PFF). Scheduler by using PFF can determine the merit of different resources for execution a certain task. We use a weighted and two-variable function for this purpose. Our proposed function consists of two parameters; time and cost. In this function the weight of completion time and the weight of execution cost (Wt, Wc) are take into account. We note that the time and cost are not of a same gender and range. To solve this problem we take them in range [0,1], in other words we normalize time and cost. The proposed objective function is:

$$PFF\,(t_i, r_j) = w_{ti} \times \frac{t_{i,j} - t_{\min}}{t_{\max} - t_{\min}} + w_{ci} \times \frac{c_{i,j} - c_{\min}}{c_{\max} - c_{\min}} \tag{1}$$

This function determines the merit of resource j for executing task i. As we see in the above function, normalized time and normalized cost are multiplied in their weights. In this function tmax and tmin are the maximum and minimum completion time of the task on appropriate resources. These two values are estimated values that produced from GIS and task information. Appropriate resources for a task are described in Section IV-C. Cmax and Cmin are the maximum and minimum execution cost of the task on appropriate resources. These two values are estimated values that produced from GIS and task information. ti,j and Ci,j are completion time and execution cost of task i on resource j.

Should be noted that the weight coefficients, determined by the user during job specification. These weights are causing the users more freedom to introduce their jobs to the grid. The goal of the scheduler is to find a resource y for each task x so that PFF(tx,ry) be Minimum. The method of finding the resource is discussed in following section.

B. *Prove efficiency of PFF*

Let's presume for task T_k resource R_b is the best resource In this case,

$$\begin{aligned}
&\forall_i\,(((c_b \le c_i) \wedge (t_b \le t_i)) \vee \\
&((c_b \le c_i) \wedge (t_b \ge t_i) \wedge (w_c \ge w_t)) \vee \\
&((c_b \ge c_i) \wedge (t_b \le t_i) \wedge (w_c \le w_t)))
\end{aligned} \tag{2}$$

Related to the above w_c and w_t are the weights of cost and time for task T_k, c_b and t_b are execution cost and completion time for task T_k on the best resource i.e. R_b. c_i and t_i are execution cost and completion time for task T_k on other resources. Now let's presume,

$$\exists_j\,(PFF(R_j, T_K) < PFF(R_b, T_k)) \tag{3}$$

in this case

$$w_c \frac{c_j - c_{\min}}{c_{\max} - c_{\min}} + w_t \frac{t_j - t_{\min}}{t_{\max} - t_{\min}} < w_c \frac{c_b - c_{\min}}{c_{\max} - c_{\min}} + w_t \frac{t_b - t_{\min}}{t_{\max} - t_{\min}}$$

$$\Rightarrow w_c \frac{c_j - c_{\min} - c_b + c_{\min}}{c_{\max} - c_{\min}} < w_t \frac{t_b - t_{\min} - t_j + t_{\min}}{t_{\max} - t_{\min}}$$

$$\Rightarrow w_c \frac{c_j - c_b}{c_{\max} - c_{\min}} < w_t \frac{t_b - t_j}{t_{\max} - t_{\min}}$$

$$\Rightarrow w_c \frac{c_j - c_b}{c_{\max} - c_{\min}} + w_t \frac{t_j - t_b}{t_{\max} - t_{\min}} < 0 \tag{4}$$

since $\frac{w_c}{c_{\max} - c_{\min}} > 0$ and $\frac{w_t}{t_{\max} - t_{\min}} > 0$, one of the following cases must be established to justify the above equation.

- First case:

$$(c_b > c_j) \wedge (t_b > t_j) \tag{5}$$

but this relationship is in contradiction with the presumption of the problem, so this case is not acceptable.

- Second case:

$$(c_b < c_j) \wedge (t_b > t_j) \wedge$$

$$\left| w_t \frac{t_j - t_b}{t_{\max} - t_{\min}} \right| > \left| w_c \frac{c_j - c_b}{c_{\max} - c_{\min}} \right|$$

$$\frac{w_t}{t_{\max} - t_{\min}} \left| t_j - t_b \right| > \frac{w_c}{c_{\max} - c_{\min}} \left| c_j - c_b \right| \tag{6}$$

this relationship must be true for different values of c_b, c_j, t_b and t_j even in the worst case:

$$c_j = c_{\max}$$

$$c_b = c_{\min}$$

$$\left| t_j - t_b \right| = \varepsilon \tag{7}$$

but in this case

$$w_t \frac{\varepsilon}{t_{\max} - t_{\min}} > w_c \tag{8}$$

As we know $\frac{\varepsilon}{t_{\max} - t_{\min}} < 1$ so $w_t > w_c$. This means that in second case the conclusion is:

$$(c_b < c_j) \wedge (t_b > t_j) \wedge (w_c < w_t) \tag{9}$$

but this relationship is in contradiction with the presumption of the problem, so this case is not acceptable too.

- Third case:

$$(c_b > c_j) \wedge (t_b < t_j) \wedge$$

$$\left| w_c \frac{c_j - c_b}{c_{\max} - c_{\min}} \right| > \left| w_t \frac{t_j - t_b}{t_{\max} - t_{\min}} \right|$$

$$\frac{w_c}{c_{\max} - c_{\min}} \left| c_j - c_b \right| > \frac{w_t}{t_{\max} - t_{\min}} \left| t_j - t_b \right| \tag{10}$$

this relationship must be true for different values of c_b, c_j, t_b and t_j even in the worst case:

$$t_j = t_{\max}$$

$$t_b = t_{\min}$$

$$\left| c_j - c_b \right| = \varepsilon \tag{11}$$

but in this case

$$w_c \frac{\varepsilon}{c_{\max} - c_{\min}} > w_t \tag{12}$$

As we know $\frac{\varepsilon}{c_{\max} - c_{\min}} < 1$ so $w_c > w_t$. This means that in second case the conclusion is:

$$(c_b > c_j) \wedge (t_b < t_j) \wedge (w_c > w_t) \tag{13}$$

since this relationship is in contradiction with the presumption of the problem, this case is not acceptable too.

So, all cases of the rule accuracy are contradicted with the presumption of the problem, so the rule is wrong. This means that there is no resource that its PFF is lower than the best resource i.e. R_b. In other words, the best resource has the minimum PFF.

C. *Proposrd Scheduler*

The first action performed by scheduler for scheduling tasks in grid is determining the appropriate collection of resources for implementing of each task. Resources those are suitable for performing task should have these conditions:

- They must have suitable Operating System and Architectural Requirements for implementing the task.
- Estimated cost of implementing the task must be lower than the cost that the user specified for performing the task.
- The user has license to use the resource.

After determine the appropriate collection of resources for implementing of each task, the scheduler must select the best resource of this collection. The best resource is the resource that it's PFF, equation (1), is Minimum. Appropriate resources in the collection may be one of the following two modes, each of which cases shall apply the different scheduling policies. These policies are described below.

A: All processors of suitable resources for performing the task are busy. In this case the scheduler put the task in the end of the waiting queue of the best suitable resource. Then SA algorithm will be called to exchange tasks in the waiting queue of different resources. In this manner the best scheduling is yield. Proposed SA is described in section IV-D.

B: some of the processors of the appropriate resources are idle. In this case, at first, the scheduler will assign the tasks that are in the waiting queue of the resources of idle processors (if any). This operation continues until waiting queue of a resource is not empty and idle processor is available. If processors are idle on some appropriate resources (empty waiting queue of resources), the scheduler will assign the task on the best idle resource. Otherwise, the scheduler will put the task at the end of waiting queue of suitable resource. Then SA algorithm will be called to exchange tasks in the waiting queue of different resources. In this manner the best scheduling is yield.

If one of the resources completing the execution of a task, the processor was appointed to the task is released. The scheduler assigns the released processor to the first task of the waiting queue of the resource. In this case after completion of each task, a new task, if any, will be assigned, and due to the fact that the proposed SA algorithm balance the waiting queue of the resources, we don't concern about an empty waiting queue in one resource and a busy waiting queue in the other. In fact, using SA, we reduce completion time and execution cost of tasks and increase resources utilization simultaneously.

D. *Proposed SA*

For improving completion time and execution cost of tasks in grid, novel SA is presented in this section. This algorithm shown in figures (3) and (4) and also the process of execution of this algorithm discussed in section IV-D-1.

In this algorithm the time and cost have different weights. Each configuration in the SA equivalent resources mapping to tasks, that indeed the solution of the scheduling problem. In each of the three solutions current, new and best, two merits one for time and one for cost will be considered. SA algorithm goal is to reduce the total cost merit and time merit of scheduling. Of course, these two merits multiplied with the average of time and cost weights of tasks.

As shown in Figure (2), time merit in each solution is the maximum active time of resources in the solution. Active time of each resource is the total of completion time of available tasks in the waiting queue of that resource. Completion time of a task is the total run time and the time consumed for scheduling tasks on the resource.

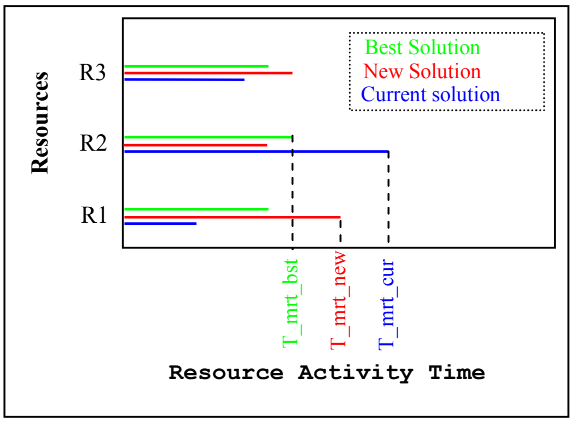


Figure 2. Time merit of different solutions on SA

Cost merit in each solution is the total cost of executing tasks on the waiting queue of the resource. General merit in each solution is a combination of cost merit and time merit of that solution. But it can't be easily determined, because normalization of time and cost is complex process.

For solve time and cost normalization problem in the SA, we directly calculate ΔMerit_cur and ΔMerit_bst. ΔMerit_cur is the difference between new and current solution merit. ΔMerit_bst is the difference between new and best solution merit. As shown in Figure (4) in the calculation of these two values, the mean weight of completion time, avg(wt), and the mean weight of execution cost, avg(wc), multiplied with the normalized difference merit of time and cost. We use these two values, ΔMerit_cur and ΔMerit_bst, to accept or reject the new solution as the current solution or the best solution.

```
AlgorithmSimulated_Annealing($S_0$, $T_{\max}$, $T_{\min}$, $k$);
Begin
  $T = T_{\max}$;   // initial temperature
  $CurS = S_0$;   // initial solution
  $BstS = CurS$;  //BstS is the best solution seen so far
  $T\_Mrt\_cur = TimeMerit(CurS)$; //time merit of current solution
  $C\_Mrt\_cur = CostMerit(CurS)$; //cost merit of current solution
  $T\_Mrt\_bst = TimeMerit(BstS)$; //time merit of best solution
  $C\_Mrt\_bst = CostMerit(BstS)$; //cost merit of best solution
  Repeat
        Call Metropolis($CurS, T\_Mrt\_cur, C\_Mrt\_cur,$
                        $BstS, T\_mrt\_bst, C\_mrt\_bst, T, k$);
         $T = \alpha \times T$  // $\alpha$ is cool rate
  Until ($T > T_{\min}$);  // $T_{\min}$ is final temperature
  Return ($BestS$);
End.
```

Figure 3. Simulated annealing algorithm

```
Algorithm Metropolis ($CurS, T\_Mrt\_cur, C\_Mrt\_cur,$
                      $BstS, T\_Mrt\_bst, C\_Mrt\_bst, T, k$);
Begin
  Repeat
     $NewS = Neighbor(CurS)$;  // new solution
     $T\_Mrt\_new = TimeMerit(NewS)$;  //time merit of new solution
     $C\_Mrt\_new = CostMerit(NewS)$;  //cost merit of new solution
     $\Delta Merit\_cur = avg(w_t) \times \frac{T\_Mrt\_new - T\_Mrt\_cur}{Max(T\_Mrt\_new, T\_Mrt\_cur)} + avg(w_c) \times \frac{C\_Mrt\_new - C\_Mrt\_cur}{Max(C\_Mrt\_new, C\_Mrt\_cur)}$
     $\Delta Merit\_bst = avg(w_t) \times \frac{T\_Mrt\_new - T\_Mrt\_bst}{Max(T\_Mrt\_new, T\_Mrt\_bst)} + avg(w_c) \times \frac{C\_Mrt\_new - C\_Mrt\_bst}{Max(C\_Mrt\_new, C\_Mrt\_bst)}$
     If ($\Delta Merit\_cur > 0$) then
        $CurS = NewS$;
        $T\_Mrt\_cur = T\_Mrt\_new$;
        $C\_Mrt\_cur = C\_Mrt\_new$;
        If ($\Delta Merit\_bst > 0$) then
           $BstS = NewS$;
           $T\_Mrt\_bst = T\_Mrt\_new$;
           $C\_Mrt\_bst = C\_Mrt\_new$;
        End if;
     Else if ($Random < e^{-\frac{\Delta Merit\_cur}{T}}$) then
        $CurS = NewS$;
        $T\_Mrt\_cur = T\_Mrt\_new$;
        $C\_Mrt\_cur = C\_Mrt\_new$;
     End if;
     $k = k - 1$;    // $k$ is the amount time for which annealing …
  Until ($k = 0$);   // must be applied at temperature T.
End.  //of Metropolis
```

Figure 4. Metropolis algorithm

E. *The process of SA algorithm*

Should be noted that the proposed SA algorithm, exchange the tasks in waiting queues of different resources to achieve an optimal Schedule. This algorithm works as the following:

Step 1: At first, we have a nearly optimal solution. In this solution each task is within the waiting queue of the best appropriate resource. Best resource is the resource which has minimum PFF (equation (1)).

Step 2: The algorithm begins from Tmax temperature.

Step 3: The current solution is considered as the best solution.

Step 4: Time and cost merit for the current solution and best solution is calculated.

Step 5: The new solution is created as follow. Method: Neighbor(CurS)

Step 5-1: two resources are selected in the current solution arbitrary.

Step 5-2: if waiting queue of first resource is not empty, one task from the waiting queue of the first resource is deleted randomly. This task may be in any place in the queue.

Step 5-3: deleted task is added to the end of waiting queue of second resource.

Step 6: Time and cost merit of new solution is calculated.

Step 7: ΔMerit_cur is calculated. This value is the difference between the new solution and current solution merit.

Step 8: If ΔMerit_cur is positive, means the new solution is better than the current solution, so the new solution replaces the current solution.

Step 9: ΔMerit_bst is calculated. This value is the difference between the new solution and best solution merit.

Step 10: If ΔMerit_bst is positive, means the new solution is better than the best solution, so the new solution replaces the best solution.

Step 11: To escape the relative optimal solutions, if ΔMerit_cur is not positive, we replace the new solution with the current solution with chance Exp(-(ΔMerit_cur/T)). This chance has direct relationship with current temperature. This means in lower temperature, new solutions that are not optimal are accepted by lower chance. In SA the temperature gradually decreases.

Step 12: Steps 5 to 11, are repeat K times.

Step 13: Temperature multiplied with iciness rate and new temperature is produced.

Step 14: Steps 5 to 13 are repeated until Tmin temperature.

Step 15: The best solution is the algorithm output.

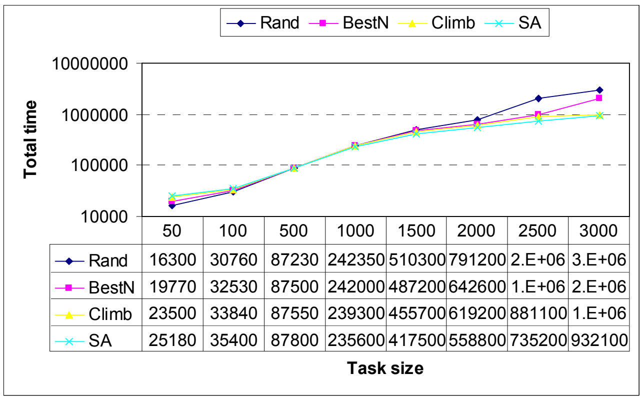
Step 16: End of algorithm.

V. Evaluation Results

In this section the results of applying the proposed algorithm for scheduling independent tasks on a computational grid network is presented. Since the objective function of a proposed scheduler is a two-variable weighted function, the results of the proposed scheduler compared with the results of three schedulers in terms of both time and cost. First Scheduler randomly assigned resources (Rand). Second scheduler is The Best of n Random scheduler randomly generates n schedules and then returns the one with the highest benefit value. Third scheduler use climb search algorithm for optimization. Figures (5) to (8) show that proposed SA scheduler is better than other three schedulers in optimization time and cost. Since our method is non-deterministic, results of each run can be somewhat different with the previous run, to ensure the results all schedulers are run at least 30 times, we consider the average performance results of these runs. Standard Deviation of different runs is 7.4%.

In time optimization the aim is to minimize the time between the submission of the application and its return. Each task should be scheduled first and then be executed. The scheduling algorithms try to minimize the duration of the execution time but at the cost of increasing the duration of the scheduling time. In order to evaluate the effectiveness of this tradeoff we consider the completion time which is the sum of the scheduling time and execution time.

Figure (5) and (6) show the difference between schedulers in terms of total time. Figure (5) shows the effect of task size on the total time of scheduling. As in Figure (5), when the size of tasks is low scheduler (1) works out better.

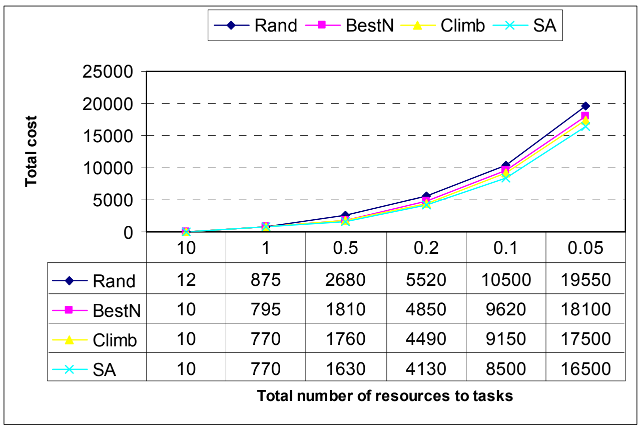


	50	100	500	1000	1500	2000	2500	3000
Rand	16300	30760	87230	242350	510300	791200	2.E+06	3.E+06
BestN	19770	32530	87500	242000	487200	642600	1.E+06	2.E+06
Climb	23500	33840	87550	239300	455700	619200	881100	1.E+06
SA	25180	35400	87800	235600	417500	558800	735200	932100

Figure 5. The effect of task's size on total time of scheduling

Because time consumed for scheduling the low-size task in schedulers (2), (3) and (4) is more than theirs execution time, and since the time consumed for Scheduler (1) is very small, this scheduler works better for low-size tasks. But with increasing the task size, schedulers (2), (3) and (4) will have better performance. The above results due to, the scheduling time of scheduler (4) is more than the scheduler (3) and scheduler (3) more than schedule (2) and scheduler (2) more than schedule (1). Accordingly, when the task size is large, the proposed scheduler (scheduler (4)) produces the shortest total time. Usually grid task sizes are large.

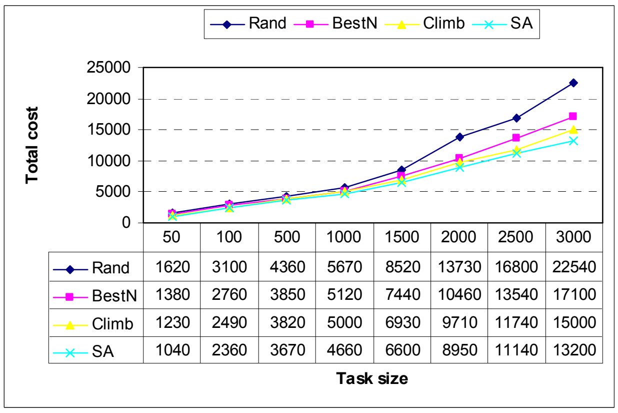
Figure (6) shows how the time of the result schedule varies with the number of processors available in the Grid. It shows that when the number of processors is less than the number of tasks, scheduler (4) outperforming the other schedulers in terms of time. If the number of processors is greater than the number of tasks in grid, the performance of schedulers (2), (3) and (4) is almost same in terms of time. The reason is that by increasing the number of processors available in grid, the waiting queue of these processors is almost empty. If waiting queue is empty, SA has no improvement in the total time of the schedule. Therefore the performance of schedulers (2), (3) and (4) is similar. In grid the number of processors is limited and their number is less than the number of tasks.



	10	1	0.5	0.2	0.1	0.05
Rand	12	875	2680	5520	10500	19550
BestN	10	795	1810	4850	9620	18100
Climb	10	770	1760	4490	9150	17500
SA	10	770	1630	4130	8500	16500

Figure 6. The effect of number of processors on total time of scheduling

Cost optimization aims to minimize the expected cost of the application. The expected cost is defined as the sum of the price per unit time multiplied by the expected execution time. The total cost of the scheduling is the sum of expected cost of the applications. Therefore the goal is minimizing the total time.



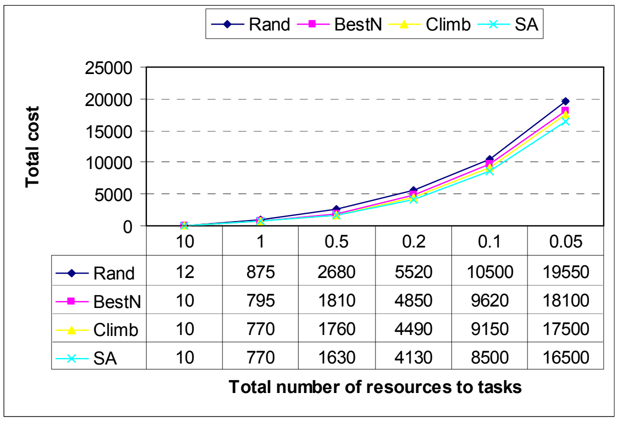
	50	100	500	1000	1500	2000	2500	3000
Rand	1620	3100	4360	5670	8520	13730	16800	22540
BestN	1380	2760	3850	5120	7440	10460	13540	17100
Climb	1230	2490	3820	5000	6930	9710	11740	15000
SA	1040	2360	3670	4660	6600	8950	11140	13200

Figure 7. The effect of task's size on total cost of scheduling

Figure (7) and (8) show the difference between schedulers in terms of total cost. Figure (7) shows the effect of task size on the total time of scheduling.

As shown in Figure (7) schedule (4) outperforming the other schedulers in any size of tasks. This is because in calculating the cost of tasks, only the execution time is important and time consuming for Schedule is not considered. As scheduler (4) reduces the execution time of tasks, it also reduces the cost of tasks.

Figure (8) shows how the cost of the result schedule varies with the number of processors available in the Grid. It shows that when the number of processors is less than the number of tasks, scheduler (4) outperforming the other schedulers in terms of cost. But if the number of processors is greater than the number of tasks in grid, the performance of schedulers (2), (3) and (4) is almost same in terms of cost. The reason is that by increasing the number of processors available in grid, the waiting queue of these processors is almost empty. If waiting queue is empty, SA has no improvement in the total time of the schedule. Therefore the performance of schedulers (2), (3) and (4) is similar. In grid the number of processors is limited and their number is less than the number of tasks.



	10	1	0.5	0.2	0.1	0.05
Rand	12	875	2680	5520	10500	19550
BestN	10	795	1810	4850	9620	18100
Climb	10	770	1760	4490	9150	17500
SA	10	770	1630	4130	8500	16500

Figure 8. The effect of number of processors on total cost of scheduling

VI. Conclusion

In this paper, we offer an algorithm for scheduling independent tasks in the computational grid. The proposed algorithm gives more freedom to user for applying his concerns about the importance of time and cost. We proposed a weighted objective function for the scheduler to consider the importance of time and cost. Both completion time and execution cost of tasks are considered simultaneously. The goal of this scheduler is minimizing the output of objective function for different tasks of users. For improving the tasks that are currently in waiting queue of different resources, we apply simulated annealing algorithm. As the objective function, in using this algorithm time and cost weights are considered.

To prove the proposed algorithm we compare it with the results of three schedulers (Random, Best of N random and Climb search) in terms of both time and cost. We show that if the number of resources is limited and the size of tasks is high, the reality is such, the proposed scheduler outperforming the other compared schedulers in reducing total cost and time. This scheduler can be extended for scheduling dependent tasks in the future. Also, in calculating the cost, network costs can be brought to account.

References

[1] Foster, Ian, Carl Kesselman, and Steven Tuecke. "The anatomy of the grid: Enabling scalable virtual organizations." International journal of high performance computing applications 15.3 (2001): 200-222.

[2] Braun, Martin, et al. "Is the distribution grid ready to accept large□scale photovoltaic deployment? State of the art, progress, and future prospects." Progress in photovoltaics: Research and applications 20.6 (2012): 681-697.

[3] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", J. Chem. Phys., vol. 21, no. 6, pp. 1087-1092, 1953.

[4] Yu, Jia, and Rajkumar Buyya. "A taxonomy of workflow management systems for grid computing." Journal of Grid Computing 3.3-4 (2005): 171-200.

[5] Carretero, Javier, Fatos Xhafa, and Ajith Abraham. "Genetic algorithm based schedulers for grid computing systems." International Journal of Innovative Computing, Information and Control 3.6 (2007): 1-19.

[6] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, “Optimization by Simulated Annealing”, Science, 220, 4598, pp. 671-680, 1983.

[7] M, Arora, S.K. Das, R. Biswas, A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments, in Proc. of International Conference on Parallel Processing Workshops (ICPPW'02), pp.:499 – 505, Vancouver, British Columbia Canada, August 2002.

[8] D. P. Silva, W. Cirne and F. V. Brasileiro, Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids, in Proc of Euro-Par 2003, pp.169-180, Klagenfurt, Austria, August 2003.

[9] L. Young, S. McGough, S. Newhouse, and J. Darlington, Scheduling Architecture and Algorithms within the ICENI Grid Middleware, in Proc. of UK e-Science All Hands Meeting, pp. 5-12, Nottingham, UK, September 2003.

[10] Schopf, Jennifer M. "Ten actions when grid scheduling." In Grid resource management, pp. 15-23. Springer US, 2004.

[11] Fujimoto, Noriyuki, and Kenichi Hagihara. "A comparison among grid scheduling algorithms for independent coarse-grained tasks." In *Applications and the Internet Workshops, 2004. SAINT 2004 Workshops. 2004 International Symposium on*, pp. 674-680. IEEE, 2004.