SCTTS: Scalable Cost-Time Trade-off Scheduling for Workflow Application in Grids

Vahid Khajehvand[1], Hossein Pedram[2] and Mostafa Zandieh[3]
[1] Department of Computer Engineering and Information Technology
Qazvin Branch, Islamic Azad University, Qazvin, Iran
[e-mail: khajehvand@qiau.ac.ir]
[2] Department of Computer Engineering and Information Technology
Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
[e-mail: pedram@aut.ac.ir]
[3] Department of Industrial Management
Shahid Beheshti University, G.C., Tehran, Iran
[e-mail: m_zandieh@sbu.ac.ir]
*Corresponding author: Vahid Khajehvand

Received July 19, 2013; revised October 20, 2013; accepted November 16, 2013; published December 27, 2013

Abstract

To execute the performance driven Grid applications, an effective and scalable workflow scheduling is seen as an essential. To optimize cost & makespan, in this paper, we propose a Scalable Cost-Time Trade-off (*SCTT*) model for scheduling workflow tasks. We have developed a heuristic algorithm known as Scalable Cost-Time Trade-off Scheduling (*SCTTS*) with a lower runtime complexity based on the proposed *SCTT* model. We have compared the performance of our proposed approach with other heuristic and meta-heuristic based scheduling strategies using simulations. The results show that the proposed approach improves performance and scalability with different workflow sizes, task parallelism and heterogeneous resources. This method, therefore, outperforms other methods.

Keywords: Cost-makespan minimization, cost-time trade-off, workflow scheduling, scalability, utility grids

A preli`minary version of this paper appeared in IEEE/ACM CCGrid 2012, May 13-16, Ottawa, Canada [36]. Then, in [37], the preliminary version is developed to study the impact of the trade-off factor on the degree of the significance of allocation cost to makespan.

http://dx.doi.org/10.3837/tiis.2013.12.008

1. Introduction

To build utility computing systems, they need to consists of a component model, a methodology, a set of tools and common services. A utility computing system is a system that automatically creates and manages multiple utility services on a shared infrastructure. The infrastructure consists of pools of the hardware resources, such as servers, storage and network tools, as well as software resources [1]. A utility Grid gives an economy model in which users pay service providers for their services due to factors such as the Quality of Service (QoS) provided.

To conduct large-scale computations, the grid environment software and hardware resources are supported by the shared distributed infrastructures. To execute applications in sciences such as earthquake [2], astronomy [3], high energy physics [4] and etc., these infrastructures have proven to be highly efficient. Workflows constitute a common model for describing a wide range of applications in distributed environments. The workflow is represented in a "Direct Acyclic Graph" (DAG) with nodes and edges representing the tasks and data dependencies among the tasks, respectively.

Once an application is transformed into the workflow structure, workflow management system will be ready to control and manage the execution of workflow on a distributed infrastructure. A taxonomy of Grid workflow management systems is found in [5]. Workflow scheduling problem is mapping each task on a suitable resource and ordering the tasks on each resource to satisfy performance criterion. Existing Grid scheduling methods try to minimize the execution time (makespan) or the execution cost of the workflows. Moreover, in utility Grids, there is much potential to study the combinations of QoS attributes.

The current methods mostly are not designed for minimizing the cost and time. Considering these QoS attributes, the scheduler faces a time-cost tradeoff in selecting appropriate services, which belongs to the multi-objective optimization problem family. However, none of these papers considered the issue of scalability.

There are three classes of approaches to the problem of multi-objective scheduling [6]. The first class of the approaches extends the definition of optimality to pareto optimality [7-10]. The second one is bi-criteria scheduling approaches, usually limited to optimizing two specific objectives [10-18]. The last one optimizes a linear combination of multiple scheduling objectives with a different weight value assigned to each one of them [9, 19, 20]. This last class assumes that the user is able to specify the requirements in such a model. This method is an easy way to express users' requirements; this method also helps researchers with simplifying the problem at the same time it proposes an effective performance solution. Many researchers use this method to develop algorithms for multi-objective scheduling problem. Due to the complexities of this problem, however most of these methods use time consuming meta-heuristic approaches [8, 9, 21] e.g., Genetic Algorithms.

To address this problem, there are few fast and efficient heuristic methods. The method in [20] is one of them. This method [20] relies on scheduling parallel application, whereas our method is based on scheduling workflow tasks. It is worth noting that in [20] the issue is proposed for future works. Workflows are loosely-coupled parallel applications that consist of a set of computational tasks linked via data dependencies, unlike tightly-coupled applications, such as parallel application, in which tasks communicate directly via the network. Workflow tasks typically communicate using the files. Each task in a workflow produces one or more output files that become input files to other tasks. If tasks are run on different computational nodes, these files will be either stored in a shared file system, or transferred from one node to

the next one by the workflow management system [22, 23]. Therefore, the method presented in [20], is not able to schedule workflow tasks.

Due to the different resource consumers and providers, the scheduling problem becomes highly complicated and NP-complete [24] in such an environment so that each party seeks its own profits. So the resource consumers and providers act independently at the same time pursuing conflicting aims. The resource consumers seek the minimum time and cost for scheduling application, whereas the main focus of resource providers is on the resource utilization gains. Thus, the main users' challenge in this environment will be scheduling an application on the heterogeneous resources. In this case, the users have no explicit control on minimizing both the time and cost. It is also scalable due to an increase in the workflow size, task parallelism and the number of resources.

This study-base is inspired by efforts to combine application management system and resources management system. To execute the workflow tasks, the resource slots required by the workflow tasks, can be virtually obtained by a resource provisioning system. In general, the cost of task execution can be computed according to the task allocation cost and task execution time. Cost and makespan optimization is of great importance due to the resources heterogeneity and scalability relative to an increase in the required number of processors, workflow size and the number of the resources.

This paper introduces a Scalable Cost-Time Trade-off (*SCTT*) model to schedule workflow application tasks for cost-makespan optimization. The model allows users to identify the degree of the significance of each optimization objective using adjusting the trade-off factor. Trade-off involves losing a quality or an aspect of criteria in return for gaining another quality or aspect. It often implies that a decision has to be made with full comprehension of both the upside and downside of a particular choice; the term is also used in an evolutionary context, in which case the selection process acts as the "decision-maker". A decision in which you need to choose between two opposite objectives or cannot be satisfied at the same time.

Scalability is an important designing goal in a grid computing. A system can be scalable with respect to its size, i.e. we can increase the workflow size and resources. To schedule workflow tasks, based on the proposed *SCTT* model, we have developed a heuristic algorithm known as Scalable Cost-Time Trade-off Scheduling (*SCTTS*). To optimize cost and time (makespan) on heterogeneous resources, our proposed heuristic algorithm schedules parallel workflow tasks according to the trade-off factor based on scalability. The main contributions of the paper include:

1) Development of a *SCTT* cost model for scheduling workflow application on heterogeneous resources with the capabilities of cost-makespan optimization.
2) Development of a *SCTTS* heuristic algorithm according to *SCTT*-based model with following characteristics: (a) study the scalability of an increase in workflow size and its impact on the allocation cost, makespan and runtime i.e. performance metrics, for workflow scheduling. (b) study the scalability of an increase in workflow tasks parallelism and its impact on performance metrics. (c) study the scalability of an increase in the number of available resources and its impact on performance metrics.

The rest of this paper is organized as follows: Section 2 discusses related works. In section 3 the details of the proposed model and heuristic algorithm are described. Section 4 deals with a simulation setup. In section 5, the relevant experiments for evaluating the efficiency of the proposed algorithm are described with results analysis. Finally, section 6 ends with a conclusion and future works.

2. Related Work

Workflow scheduling problem has been extensively studied and a number of scheduling algorithms have been proposed. There is a comprehensive introduction on the job scheduling strategies [25, 26]. In [27], the computational models are surveyed for grid scheduling problems and their resolution.

Workflow scheduling algorithms are classified into two main groups: best effort and QoS constraint based scheduling [28]. The first group are further classified into four groups: list scheduling heuristics [5, 12, 29, 30], clustering heuristics [31, 32], task duplication heuristics [31, 32] and guided random search [31-35]. But in QoS-based, there are few works addressing workflow scheduling with QoS. They mainly consider the makespan or execution cost of the workflow as the major QoS attribute. As a result, they are suitable for community Grids, moreover, in utility Grids, there is much potential to study the combinations of QoS attributes. The current methods mostly are not designed with the aim of minimizing the cost and time. Also, the scalability relative to an increase in the workflow size, task parallelism and heterogeneous resources is scarcely considered.

Therefore, there are few fast and efficient heuristic methods addressing this problem. One of them is the method in [20]. This method [20] relies on scheduling parallel applications, whereas our method is centered on scheduling workflow tasks and also includes scalability. Due to the data dependencies among tasks, scheduling workflow tasks becomes more complex than scheduling parallel application.

There are a handful studies conducted on the cost optimization of workflow scheduling close to the current paper's study. In [21] a genetic algorithm is proposed to find an optimized mapping of tasks on resources, minimizing both financial cost and makespan. This approach is developed in [8, 9] presenting cost-based model in which resource providers advertise available resource slots to users. A multi-objective genetic algorithm is presented capable of provisioning a subset of resource slots to minimize application makespan under minimum resource allocation cost. The main difference between these cost minimization algorithms and our proposed algorithm lies in the fact that their works do not pay attention to the heterogeneity of the resources. Thus, their entire resources possess identical CPU rating as well as cost processing whereas, our proposed method pays attention to heterogeneous clusters with different processing cost and CPU rating in real-world Utility Grid environments. Hence, our focus on resource heterogeneity makes the selection of appropriate resource become very complex.

To allocate a task to a resource, in [36], we presented a preliminary version of the proposed algorithm so that it selects a task with a minimum first fit cost-makespan objective function. Again, in [37], this algorithm is expanded to study the impact of the trade-off factor on the degree of the significance of allocation cost to makespan. However, none of these two papers considered the issue of scalability. While, in the proposed approach, a cost based workflow scheduling model and a heuristic scheduling algorithm are added according to the proposed model. To optimize cost to makespan with respect to scalability, the proposed approach schedules parallel workflow tasks even with a lower runtime complexity.

3. Proposed Model and Heuristic Algorithm

In general, users need two QoS: service prices (cost) and execution time (makespan) of their application scheduling on pay-per-use services [7, 38]. Users, normally tend to run their applications in as the lowest makespan and cost as possible. The trade-off factor (α) represents the user's preference between the allocation cost and the makespan. The trade-off factor is a

number between 0 to 1. If "trade-off factor=0", the allocation cost will be insignificant, while the makespan will become significant. So the existing problem becomes time (makespan) optimization problem. If "trade-off factor=0.5", both allocation cost and makespan will be of equal importance, in other words, the main issue will become the cost-time optimization. The scheduler needs to distribute the tasks on the resources to satisfy both criteria. If "trade-off factor=1" the makespan will be insignificant. As a result the problem becomes a cost optimization one, so due to the allocation cost, the scheduling algorithms need to seek economic resources.

In this section, based on a model known as the *SCTT* model, the workflow scheduling problem will be discussed. Moreover, to optimize workflow cost-time, workflow scheduling problem will be solved. According to *SCTT* model, a scalable heuristic algorithm will be developed to solve workflow scheduling problem.

3.1 The Proposed *SCTT* Model

The proposed model consists of a set of heterogeneous users and resources. The users request the execution of an application. *R* is a set of resources in an environment. The resources are considered by a set of time slots where each time slot contains a start time, a finish time and a number of the available processors.

A workflow application is represented in a DAG. A DAG is defined as G = (V,T), where *V* is a set of nodes so that each node represents a task and *T* is a set of links, each link representing the computation precedence order between two tasks. For example, a link $(i, j) \in T$ represents the precedence constraint of the task v_i that needs to be completed before task v_j starts. The data is a *V*V* matrix of communication data, where d_{ij} is the amount of data required to be transmitted from task v_i to task v_j. In a workflow, a task which does not have any parent task is known as an entry task, denoted as v_0 and a task which does not have any child task is known as an exit task, denoted as v_{n+1}. For simplicity, the application is assumed to have a single entry task and a single exit task. If there is more than one entry task or exit task in the workflow, they will be connected to a zero-time pseudo entry/exit task.

If task *t* is executed on one of the available slots of the resource *r*, its Estimated Execution Time (*EET*) will be represented by *EET(i,r)*. The Estimated Allocation Cost (*EAC*) of task *i* on the slots of resource *r* is obtained by the following equation:

$$EAC(i,r) = EET(i,r) \times C_r \times RNP_i \quad \forall i \in T, r \in R, \tag{1}$$

where C_r is an allocation cost of each slot time of the resource *r* for a single processor in time unit. Furthermore RNP_i is the required number of processors of the task *i*.

As the application is defined in DAG form, there needs to be a data dependency between application tasks. If two dependent tasks are to be executed on the slots of the same resource File Transfer Cost (*FTC*) will be negligible. Otherwise its *FTC* needs to be taken into account. Also, assuming that there is a parallel data transfer between different resources, maximum value of the *FTC* needs to be considered as a file transfer cost. So the total FTC(*TFTC*) for allocating task *j* on resource *r* is computed by:

$$TFTC(j,r) = \max_{\forall i \in parentTasks(j)} \{FTC(i,j)\}, \quad j \in T, r \in R, \tag{2}$$

where *parentTasks* is a set of the predecessor tasks of a task. To solve workflow scheduling problem, we find a resource that minimizes *TFTC*. As we see in Eq.(2) if two tasks *i, j* with a data dependency are allocated on the slots of the same resources, its *FTC* will becomes negligible.

To compute Earliest Start Time(*EST*) of each eligible task on each resource slot, the following recursive equation is used:

$$EST(j,r)=\max\{EAT(j,r),\max_{\forall i\in parentTasks(j)}\{EST(i,q)+EET(i,q)+TFTC(j,r)\}\},\ j\in eligibleTasks,\ r,q\in R \quad (3)$$

where Estimated Available Time (*EAT*) is the available slot of resource *r* for executing task *j*. The inner block of Eq.(3) is used for computing Earliest Finish Time(*EFT*) of immediate predecessor tasks of the desired task. If the desired task has more than one immediate predecessor task, the maximum *EFT* of immediate predecessor tasks will be considered as the *EST* of immediate successor task. Eq.(3) computes the *EST* of each task in recursive and bottom up approach so that it starts from computing the *EST* of the input task and finally ends with computing the *EST* of the output task. Therefore, the *EST* of the input task is initialized with simulation current time.

To obtain the *EST* of each task based on Eq. (3), the *EFT* of execution of each task will be obtained:

$$EFT(j,r)=EST(j,r)+EET(j,r),\ j\in eligibleTasks,r\in R. \quad (4)$$

The resource whose slots have the minimum allocation cost and execution finish time of an eligible task, is obtained by Eq. (5). The value of alpha (α) is a number between 0 and 1 which is considered as a constant trade-off factor. This trade-off factor shows the degree of the significance of allocation cost to execution finish time

$$P_j=\arg\min_{\forall r\in R}\{\alpha\times NEAC(j,r)+(1-\alpha)\times NEFT(j,r)\},\ j\in eligibleTasks, \quad (5)$$

where *eligibleTasks* is a set of tasks whose execution of parents' tasks have been completed. *NEAC* and *NEFT* show normal values of *EAC* and *EFT* respectively obtained by the following equation:

$$NEAC(j,r)=\frac{EAC(j,r)-\min_{\forall q\in R}\{EAC(j,q)\}}{\max_{\forall q\in R}\{EAC(j,q)\}-\min_{\forall q\in R}\{EAC(j,q\}},\ j\in eligibleTasks,r\in R \quad (6)$$

$$NEFT(j,r)=\frac{EFT(j,r)-\min_{\forall q\in R}\{EFT(j,q)\}}{\max_{\forall q\in R}\{EFT(j,q)\}-\min_{\forall q\in R}\{EFT(j,q\}},\ j\in eligibleTasks,r\in R \quad (7)$$

As the values range of both target parameters *EAC* and *EFT* differ from one another, values normalization is important. After obtaining the best resource for executing each eligible task, the task that minimizes cost metric of Eq. (8), is the task to be allocated to the best related resource.

$$selectedTask=\arg\min_{\forall j\in eligibleTasks}\{\alpha\times NEAC(j,p_j)+(1-\alpha)\times NEFT(j,p_j)\}. \quad (8)$$

When all of the application tasks are allocated to the resources slots according to the above-mentioned equations, the allocation cost and makespan of application will be computed according to the two following equations:

$$allocationCost=\sum_{\forall j\in T}\sum_{\forall r\in R}EAC(j,r), \quad (9)$$

$$makespan=\sum_{\forall r\in R}EFT(n+1,r)-\sum_{\forall r\in R}EST(0,r). \quad (10)$$

The objective function of the workflow scheduling problem is a multi-objective one that is obtained by minimizing the allocation cost and makespan. The objective function is stated as the following equation:

$$\min\ f(x)=\{\alpha\times allocationCost+(1-\alpha)\times makespan\} \quad (11)$$

To solve the workflow scheduling problem, a heuristic algorithm is developed in the section 3.2, based on the presented model in this section.

3.2 The Proposed *SCTTS* Heuristic Algorithm

To solve workflow scheduling problem, based on proposed *SCTT* model, the *SCTTS* heuristic algorithm is developed. The proposed model schedules application tasks to optimize cost and

time on heterogeneous resources in a scalable manner. Details of the algorithm are described in this section.

As the application is in the form of DAG, therefore, it is necessary to consider task execution precedence during scheduling application tasks. Also, as the resources are selected from a heterogeneous distributed environment, to execute the tasks, there are many options for scheduling. Since in the scheduling process one seeks to optimize two objectives, cost and time, so to show the degree of importance of one objective to another one, a trade-off factor is necessary. The suitable slots are, therefore, selected according to objective function in Eq. (11).

The *SCTTS* pseudo-code is shown in algorithm 1, which is based on the proposed *SCTT* model. At first, the algorithm obtains the list of unscheduled tasks (line 1). Then, the unscheduled tasks are scheduled in the process of executing lines 2 to 19. For scheduling the tasks, a list of all eligible tasks to be executed is obtained (line 3). An eligible task is a task whose execution of all of its parent tasks are completed. The best resource needs to be obtained for executing each eligible task (lines 4 to 13). A list of all tasks which are parent tasks of eligible task *j* is obtained according to line 5. To execute eligible task *j*, slots of the available resource are examined (lines 6 to 11). In the process of line 7, the list of available slots of resource *r* is obtained. File transfer cost for executing task *j* on resource *r* is computed by Eq. (2) (line 8). According to Eqs. (3) and (4), *EST* and *EFT* of task *j* are computed on resource slots *r* respectively (line 9 and 10). Having computed *EST* and *EFT* of task *j* on all resources, we obtain the best resource according to Eq. (5) according to line 12. The algorithm selects the task that minimizes objective function of Eq. (8) as the best task according to Eqs. (5 to 7), (line 14). The best task is assigned to the best resource slots in order to be executed (line 15). Having assigned the best task to the best resource, the available slots characteristics need to be updated (lines 16 and 17). After scheduling the selected task, it needs to be deleted from the unscheduled tasks list (line 18). At the end of the scheduling process of all the tasks, the allocation task and makespan of the application are computed according to Eqs. (9) and (10), (lines 20 and 21).

3.3 Time Complexity

Supposing that *k* is the average number of the time slots available to the resource *j* in time *t* and *n*, *m* are the number of workflow tasks and the number of resources available to the system respectively, the main operations during executing the algorithm *SCTTS* is as follows:

- In the worst case, the appropriate assignment of the time slots to workflow tasks is repeated as many as the number of tasks as in lines 2 to 19, i.e. $O(n)$.
- In each repetition of steps 4 to 13 eligible task are to be searched. As at any level of workflow graph there are on average $\sqrt{n}$ nodes, so the order of its execution will become $O(\sqrt{n})$.
- In each repetition of steps 6 to 11, in the worst case, available time slots of each resource will be obtained. Next, according to Eq.(3) and Eq.(4) *EST* and *EFT* are obtained respectively. At the end of these steps the best resource will be obtain for executing each task whose order of its execution will become $O(mk)$.

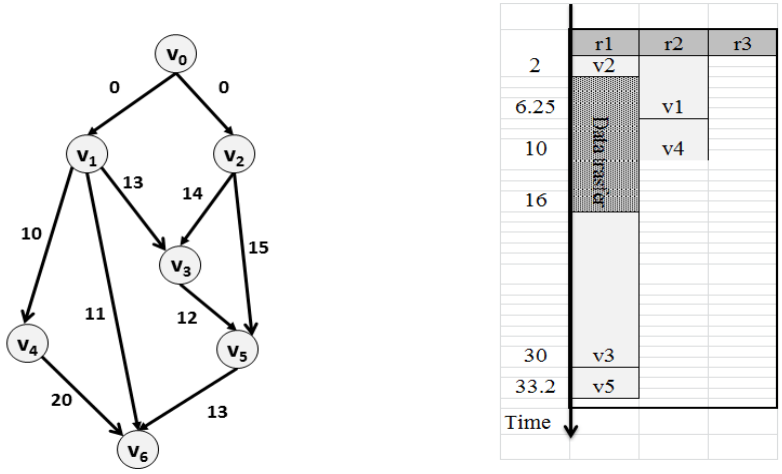
Therefore, the resultant worst case time complexity of the *SCTTS* algorithm will be $O(n\sqrt{n}\,mk)$.

Algorithm 1:	**The pseudo-code for the *SCTTS* heuristic algorithm**
Input:	An application characteristics The reade-off fcator (α) is a number between 0 to 1
Output:	The resource characteristics and the available slots to each resource A workflow application scheduling

```
1   unscheduledTasks = get the list of yet unscheduled tasks
2       While (list of unscheduledTasks is not empty) do
3           eligibleTasks = get the set of unscheduled ready tasks whose parent tasks have been
            scheduled
4           for all j ∈ eligibleTasks do
5               parentTasks = find the set of tasks that are the parents of task j
6               for all r ∈ avail Resources do
7                   get the slots list available to resource r
8                   get the required TFTC for executing task j on the slots of resource r
                    according to Eq. (2)
9                   get the EST of task j on the slots of resource r according to Eq. (3)
10                  get the EFT of task j on the slots of resource r according to Eq. (4)
11              end for
12              get the best resource for executing task j according to Eq. (5)
13          end for
14          get the best eligible task for allocating according to Eq. (8)
15          allocate the best eligible task to the best resource capable of executing it
16          update the slots available to the allocated resource
17          update EAT
18          delete the selected task from the unscheduled tasks list
19      end while
20  get the total allocation cost of application according to Eq. (9)
21  get the makespan of application according to Eq. (10)
```

3.4 The SCTTS Heuristic Example

Table 2 illustrates the *SCTTS* heuristic algorithm with a step-by-step explanation of the mapping of tasks in a sample workflow in **Fig. 1(a)**. The sample workflow has five tasks denoted as v_1, v_2, v_3, v_4 and v_5 with different execution time and data transfer requirements. As a workflow can have sub-workflows with multiple entries and exits, it is necessary to add two pseudo tasks i.e. a top task (v_0) and a bottom task (v_6), with zero execution time as **Fig. 1(a)** shows. While the top task spawns all actual entry tasks, the bottom task joins all actual exit tasks. **Fig. 1(a)** shows the output data of each task. The **Table 1** shows the *EET* and *EAC* of each task. The tasks can be mapped to the slots of three Grid resources r_1, r_2 and r_3 with different processing capability and transfer capacity.



(a) A sample workflow task graph with 7 tasks

(b) Task graph scheduling using *SCTTS* heuristic algorithm

Fig. 1. An example workflow and associated task scheduling using *SCTTS* heurestic algorithm

Table 1. The workflow task execution characteristics

Workflow task	Estimated Execution Time (*EET*) of task on the slots of a resource			Estimated Allocation Cost (*EAC*) of task on the slots of a resource		
	r_1	r_2	r_3	r_1	r_2	r_3
v_0	0	0	0	0	0	0
v_1	10	6.25	5	10	12.5	15
v_2	2	1.25	1	2	2.5	3
v_3	14	8.75	7	14	17.5	21
v_4	6	3.75	3	6	7.5	9
v_5	3.2	2	1.6	3.2	4	4.8
v_6	0	0	0	0	0	0

First, the parents' tasks of each eligible task is obtained as shown in **Table 2.** Using the proposed model, *EAT*, *TFTC*, *EST* and *EFT* of each eligible task is calculated, (Section 3.1). Using the obtained values, we obtain *NEAC* and *NEFT* i.e. Eqs. (6) and (7) which are normal values of *EAC* and *EFT* respectively. Having computed *EST* and *EFT* of task *j* on all resources, we obtain the best resource slot according to Eq. (5) for each step. The algorithm selects the task that minimizes objective function of Eq. (8) as the best task according to Eqs. (5 to 7). The best task is assigned to the best resource slot for being executed. Having assigned the best task to the best resource, the available slots characteristics need to be updated. After scheduling the selected task, it needs to be deleted from the unscheduled tasks list. This process will be repeated for all steps. In the end of the scheduling all the tasks, the allocation task and makespan of the workflow will be computed according to Eqs. (9) and (10). **Table 2** illustrates the *SCTTS* heuristic algorithm with a step-by-step explanation of the mapping of each task on the resource slot.

Table 2. Step-by-step explanation of *SCTTS* heurestic algorithm

Step	Eligible tasks	Parent tasks	*EAT*			*TFTC*			*EST*			*EFT*			*NEAC*			*NEFT*			Selected resource	Selected task
			r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3	r_1	r_2	r_3		
1	v_1	$\{v_0\}$	0	0	0	0	0	0	0	0	0	10	6.3	5	0	0.5	1	1	0.3	0		
	v_2	$\{v_0\}$	0	0	0	0	0	0	0	0	0	2	1.3	1	0	0.5	1	1	0.3	0	r_2	v_1
2	v_2	$\{v_0\}$	0	6.3	0	0	0	0	0	6.3	0	2	7.5	1	0	0.5	1	0.2	1	0		
	v_4	$\{v_1\}$	0	6.3	0	10	0	10	6.3	6.3	6.3	22	10	19	0	0.5	1	1	0	0.8	r_1	v_2
3	v_3	$\{v_1,v_2\}$	2	6.3	0	14	14	14	20	20	20	34	29	27	0	0.7	1	1	0.8	0		
	v_4	$\{v_1\}$	2	6.3	0	10	0	10	16	6.3	16	22	10	19	0	0.5	1	1	0	0.8	r_2	v_4
4	v_3	$\{v_1,v_2\}$	2	10	0	14	14	14	16	24	20	30	33	27	0	0.7	1	0.5	1	0	r_1	v_3
5	v_5	$\{v_2,v_3\}$	30	30	30	0	15	15	30	45	45	33	47	47	0	0.5	1	0	1	1	r_1	v_5

As an illustration, **Fig. 1(b)** presents the scheduling of the workflow tasks obtained by the *SCTTS* heuristic algorithm for the sample workflow of **Fig. 1(a).** The execution makespan and allocation cost are 33.2 and 39.2 respectively.

4. Environment Simulation Setup

In this section, we study simulation environment characteristics. The Grids testbed platform environment which is modeled in this simulation, consists of ten sites of a subset of European Data Grid (EDG) spread across five countries which are interconnected via high-speed network links [20, 39]. The mean bandwidth value of the resources is 10 Gb/s with a mean latency time of 150 s. **Table 3** shows the resources configuration on the grid testbed in order to simulate the distribution system including the cost of using a processor, CPU rating, number of CPUs and site-location of each resource. Each resource configuration is selected so that modeled environment reflects resource heterogeneity [20].

The workload simulated on these sites is based on the workload model generated by Lublin [40]. The main purpose of using this model is to create a more realistic simulation environment where the tasks compete with each other. **Table 4** shows the workload parameters values applied to in the Lublin model.

Table 3. Simulated EDG testbed resources

Site name (Location)	Number of CPUs	Single CPU rating(MIPS)	Processing cost(G$)
RAL(UK)	20	1140	0.0061
Imperial College(UK)	26	1330	0.1799
NorduGrid(Norway)	265	1176	0.0627
NIKHEF(Netherlands)	54	1166	0.0353
Lyon(France)	60	1320	0.1424
Milano(Italy)	135	1000	0.0024
Torina(Italy)	200	1330	1.856
Catania(Italy)	252	1200	0.1267
Padova(Italy)	65	1000	0.0032
Bologna(Italy)	100	1140	0.0069

Table 4. Lublin workload model parameter values

Workload parameter	Description	Value
JobType	The type of job	Batch Jobs
P	Maximum number of CPUs required by a job	1000
uHi	It is the default $\log_2$ of the maximum number of CPUs required by a parallel job	$Log_2(p)$
uMed	It is the default $\log_2$ of the medium size of parallel jobs in the system. Note that uMed should be in [uHi-1.5, uHi-3.5]	uHi-1.5
Other parameters	As presented by Lublin model	-

To conduct experiments, parameterized graph generator is used for creating a synthetic workflow [32]. **Table 5** shows the workflow application characteristics as in [9].

Table 5. Workflow application characteristics

Workflow characteristics	Value
The number of tasks in the workflow application (n)	100
The average runtime of each task	1000 sec
The average number of processors per task	25
The average depth of the workflow graph	$\sqrt{n}$
The average number of tasks per level	$\sqrt{n}$
The mean value of data transfer among the tasks	1000 *Gb*

A benchmark application scheduling algorithm that uses the best-effort QoS for scheduling is simulated and tagged as the *BE*. In the *BE*, the exploited heuristic method selects a resource with the minimum number of tasks in the waiting and running queues [9].

A different benchmark application scheduling algorithm using cost model, is presented by Singh et al. [8, 9]. Their algorithm has provisioned a set of slots to optimize performance under minimum allocation cost in order to execute application on provisioned slots. This cost-modeled algorithm manages a trade-off between makespan and allocation cost based on the trade-off factor. Also scheduling is conducted using multi-objective genetic algorithm [41]. It is tagged as the *MOGA* for brevity [8, 9].

For our experiments, we use the GridSim [42-44] to simulate both benchmark algorithms, testbed platform environment and workload on an Intel Core 2 Duo CPU T9600, 2.80 GHz computer.

We compare our proposed heuristic denoted as *SCTTS* with the *MOGA* and *BE* algorithms by conducting a number of experiments, in which we simulate a variety of real-world scenarios. The three performance metrics used to evaluate the scheduling approaches are the average makespan, allocation cost and runtime. In next section the obtained simulation results will thoroughly be analyzed.

5. Results Analysis

This section involves a comparison and analysis of the application performance results with objectives such as the makespan, allocation-cost and runtime of the proposed *SCTTS* algorithm along with the *MOGA* and the *BE* algorithms [8, 9]. Also, it will show how the proposed heuristic schedules the application on the distributed resources at the same time, optimizing the makespan and allocation cost in minimum runtime. According to the mentioned characteristics in section 4, a synthetic workflow application is produced with "trade-off factor=0.5" [8, 9]. The rest of the simulation parameters is compatible with the setup in section 4. The Y-axis is drawn in logarithmic scale to make the results discernible

.

5.1 The Trade-off Factor Impact

This section examines an experiment conducted for studying the variability impact of the trade-off factor on the makespan and allocation-cost whose results will be analyzed. **Fig. 2** shows the variability impact of the trade-off factor on allocation-cost of three algorithms: the *SCTTS*, the *MOGA* and the *BE* on the heterogeneous resources. Obviously, in all instances except when the trade-off factor is zero, the allocation cost of the *SCTTS* algorithm becomes less than both the *BE* and *MOGA* algorithms. There seems to be an abnormality in "trade-off factor=0" however, it is not true as the workflow scheduler is seeking a resource with the lowest time, whereas the allocation-cost is insignificant. In other words, according to the cost metric of Eq. (8), the allocation-cost will have no effect.

Fig. 3 shows the variability impact of the trade-off factor on the makespan during the workflow scheduling using three algorithms: the *SCTTS*, *MOGA* and *BE* on the heterogeneous resources. In all cases, except when the trade-off factor is equal to 1, the makespan of the *SCTTS* algorithm `is less than both the *MOGA* and *BE* algorithms. The abnormality which develops in "trade-off factor=1" is logical, because the workflow scheduler seeks resource slots with the minimum allocation cost for assigning a single task, whereas the execution time is insignificant, that is, according to the cost metric definition of Eq. (8), the makespan will have no effect. According to the cost metric function, both objectives are to be considered. Observing simultaneously **Fig. 2** and **Fig. 3** indicates the *SCTTS* algorithm can obtain the best solution in all cases. Therefore, the proposed approach can be used for solving the allocation cost optimization, makespan optimization and cost-makespan optimization. We included the confidence intervals in the simulation results in appendix 1 of the manuscript. Moreover, to reinforce the statistical level of the results, paired sample t-test are conducted and reported. It is worth mentioning that these statistical results are only provided for this section to statistically show the superiority of the proposed algorithm.

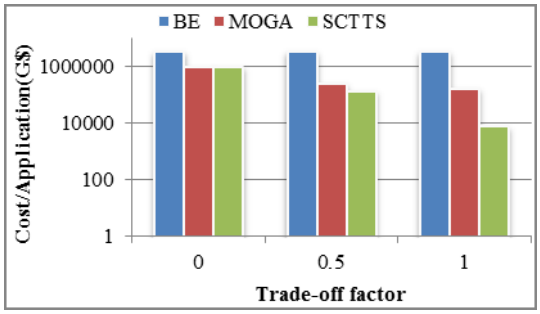


Fig. 2. Variability impact of the trade-off factor on workflow allocation cost

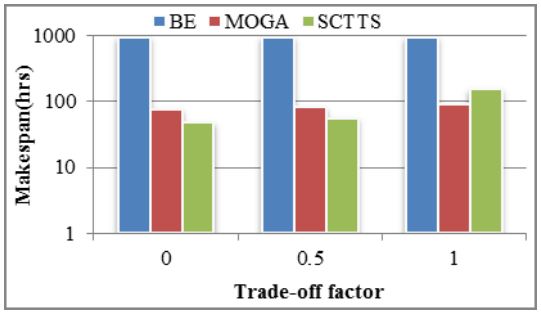


Fig. 3. Variability impact of the trade-off factor on workflow makespan

5.2 Impact of The Workflow Size on The Performance

A few experiments have been conducted to determine the impact of the workflow size on the allocation cost, makespan and runtime in terms of the number of the application tasks. It is followed by an analysis of the comparison between the *SCTTS*, *MOGA* and *BE* algorithms. In these experiments, the application characteristics are similar to those in section 4. The experiments were conducted with the application tasks' sizes of 25, 50, 100, 200, 300 and 500 in order to study its impact on the allocation cost, makespan and runtime in the application scheduling according to the increasing number of the application tasks.

Fig. 4 and **Fig. 5** show the impact of the workflow size on the allocation cost and makespan

in the application scheduling, respectively. As **Fig. 4** and **Fig. 5** indicate that the allocation cost and makespan of the proposed algorithm in all instances are around 82% and 31% less than the *MOGA* algorithm, respectively and also, one order of magnitude less than the *BE* algorithm. The low cost and makespan in the proposed algorithm is explained by the fact that it selects a slot with the minimum start-time for running the eligible task from the whole existing slots according to the cost metric of Eq. (8). However, the *MOGA* algorithm randomly selects a subset of the slots for scheduling the whole tasks. According to the existing data dependency among workflow tasks, if the execution of an eligible task is postponed, it will result in lengthening the makespan. In the *BE* algorithm, as long as the executions of the parent tasks are not completed, child-tasks will not be submitted. As the workflow graph depth is $\sqrt{n}$, the higher the number of the tasks (n) are, the deeper the workflow will be. Eventually, an increase in the workflow graph depth causes an increase in the number of the times a task needs to wait, resulting in an increase in the makespan.

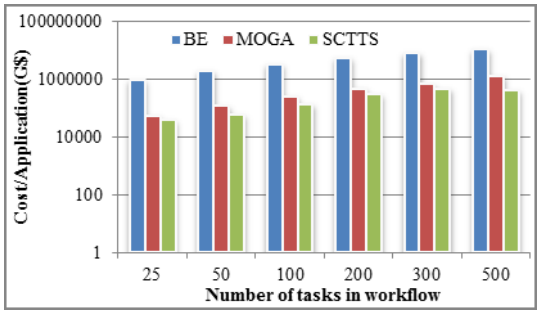


Fig. 4. Workflow size impact on workflow allocation cost

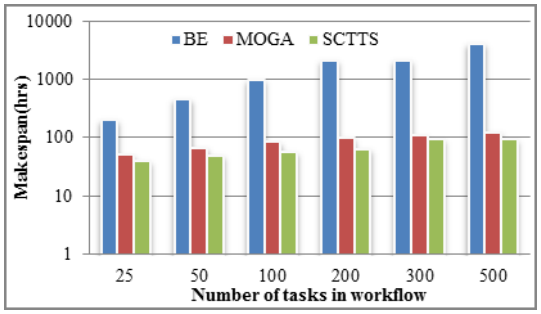


Fig. 5. Workflow size impact on workflow makespan

Fig. 6 shows the *SCTTS*, *MOGA* and *BE* algorithms' runtime relative to an increase in the number of the application tasks. As the figure shows, the proposed algorithm in all instances of an execution is almost three orders of magnitude less than the *MOGA* and the *BE* algorithms. The low time-complexity of the proposed algorithm is explained by the fact that it seeks the best slot for a single task just once, while the *MOGA* algorithm is implemented based on the genetic algorithm repetitively. One of the disadvantages of the genetic algorithms is length of their runtime. Moreover, in order to seek a subset of proper slots, the *MOGA* algorithm needs to repetitively plan the whole chromosomes of each generation of the population so that the best solution of each generation can be selected. The whole process involves a very high time-complexity. Therefore, the higher the number of the application tasks, the longer the runtime of the algorithm is. According to **Fig. 6**, if the number of the workflow tasks increases from 300 tasks to 500 tasks in the *MOGA* algorithm, its runtime will increase around one order of magnitude. The *BE* algorithm employs the best-effort service while disregarding the cost metric optimization. After the execution of all parent tasks of the corresponding task are completed, the execution of the desired task will cause the runtime to lengthen.

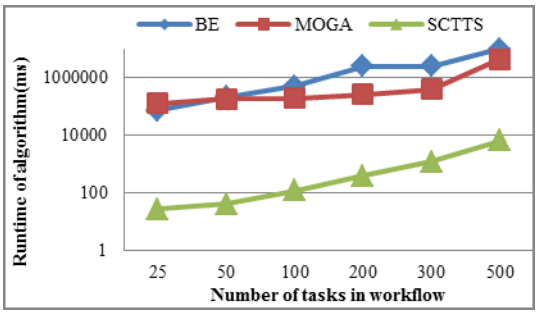


Fig. 6. Workflow size impact on application runtime

According to **Fig. 6**, due to an increase in the application tasks even when it is running 500 tasks, the *SCTTS* algorithm requires much lower runtime. The runtime required by the *SCTTS* algorithm is around 6.7 second for 500 tasks to be executed, whereas in the *MOGA* algorithm, the application runtime lasts almost one hour and twenty minutes. As a result, the *SCTTS* algorithm is scalable caused by an increase in the application tasks as well as capable of scheduling huge applications with the lowest runtime in a heterogeneous environment.

5.3 Impact of The Task Size on Performance

In this section, a few experiments were conducted for studying impact of a task size on the allocation cost, makespan and runtime of the scheduling algorithms. Next, the *SCTTS*, *MOGA* and *BE* algorithms are compared to one another. The task size is the number of the required processors all of which need to be available for execution. In these experiments, the application characteristics are the same as the ones in section 4 taking into account the average number of the required processors of each task: 1, 5, 10, 25, 50, 75 and 100, respectively. The higher the average number of the required processors for application tasks are, the higher the degree of the task parallelism is. These experiments are intended to study the impact of an increase in the application tasks parallelism on the allocation cost, makespan and runtime.

Fig. 7 shows impact of a task size on the allocation cost of the application scheduling. As **Fig. 7** indicates the allocation cost of the *SCTTS* algorithm decreases more than 60% compared to the *MOGA* algorithm due to an increase in the degree of the tasks parallelism in all instances. Also, the allocation cost of the *SCTTS* algorithm decreases in all instances almost one order of magnitude compared to the *BE* algorithm caused by an increase in the degree of the tasks parallelism. **Fig. 8** shows impact of a task size on the makespan in the application scheduling. According to **Fig. 8**, the makespan of the *SCTTS* algorithm is better than the *MOGA* algorithm in terms of an increase in the degree of task parallelism in all instances. However, due to the fact that an increase in the degree of the task parallelism, the makespan of the *SCTTS* and *MOGA* algorithms decreases one order of magnitude in all instances compared to the *BE* algorithm. Thus, a concurrent observation of **Fig. 7** and **Fig. 8** in corresponding cases, demonstrates that the *SCTTS* algorithm has a better performance compared to both *BE* and *MOGA* algorithms.

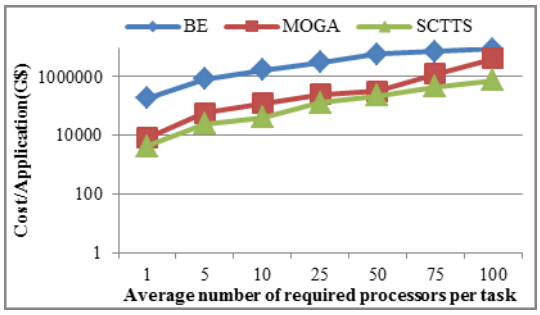


Fig. 7. Task size impact on application allocation cost

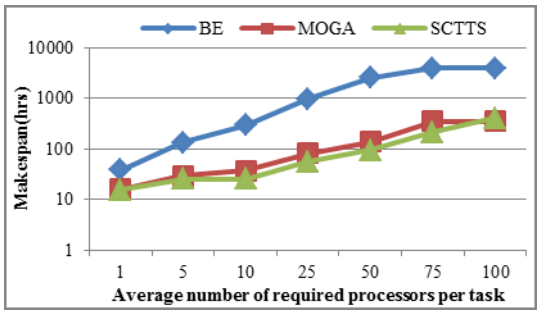


Fig. 8. Task size impact on application makespan

Fig. 9 shows the impact of a task size on the runtime in the application scheduling. As **Fig. 9** shows the *SCTTS* algorithm runtime is at least three orders of magnitude less than the *BE* and *MOGA* algorithms in all instances. It is observed in **Fig. 9**, when the results of the *BE* and *MOGA* algorithm are compared, the abnormalities appear, that is, the runtime of the *BE* algorithm takes less time than the *MOGA* algorithm does as long as the average number of the required task processors remains less than 10. The abnormalities are due to the resource scheduling policy in the *BE* algorithm which is based on the best effort quality of service. Similarly, because of the low degree of the tasks parallelism, there are many ready slots for running each single task. Therefore, in such circumstances, the *BE* algorithm is more suitable than the *MOGA* algorithm in terms of the runtime. However, once the average number of the required tasks processors exceeds 10 processors, the runtime of the *BE* algorithm increases one order of magnitude compared to the *MOGA* algorithm. It is explained by the fact that the *BE* algorithm submits each single task to a resource with the least number of tasks in waiting and running queues. Also, the *BE* algorithm cannot schedule the tasks with a higher degree of the tasks parallelism in an earlier start-time. As a result, waiting time for the fulfillment of the parent task of each single task will become longer which results in a longer runtime.

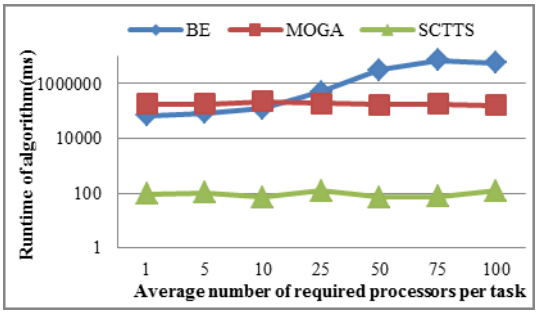


Fig. 9. Task size impact on workflow runtime

The importance of the experiments is shown by the fact that the impact of the results of an increase in the degree of the tasks parallelism becomes clear. As the results of **Fig. 7**, **Fig. 8** and **Fig. 9** show, the *SCTTS* algorithm may deliver a better performance for scheduling the applications with higher degrees of the task parallelism that optimizes the cost-makespan.

5.4 Impact of The Number of Resources

A few experiments are also conducted for studying the impact of the number of the resource providers on the allocation cost, makespan and runtime of the algorithms. The performances of the *SCTTS*, *MOGA* and *BE* algorithms were compared. In these experiments the application

characteristics are as the same as the ones in section 4. The experiments are conducted on a number of different resource providers. According to **Table 3**, firstly, the experiments are conducted on the first three resources, secondly, on the first five resources, next, on the first seven resources after that, on the first nine resources and finally, on the entire ten resources. The impact associated with the different numbers of the resources on the allocation cost, makespan and runtime in the application scheduling are to be considered.

Fig. 10 shows the impact of the different numbers of the resources on the allocation cost. The cost of the proposed algorithm is almost 40% lower than that of the *MOGA* algorithm as well as one order of magnitude of the *BE* algorithm in all cases, except in case, there are 3 and 5 resources. As **Fig. 10** indicates, with an increase in the number of resources, there is an increase in the allocation cost of the *MOGA* algorithm compared to the *SCTTS* algorithm. This increase in the cost is explained by the fact that an increase in the number of resources results in an increase in the number of slots, bearing it in mind that the *MOGA* algorithm is implemented by a genetic algorithm. The number of the slots constitutes the number of the solution chromosomes' genes. As a result, with an increase in the number of the resources, there will be an increase in the number of the genes. Thus the algorithm is trapped in the local optimized solutions.

Fig. 11 shows the impact of the number of the resources on the makespan. The makespan of the proposed algorithm is 14% lower than that of the *MOGA* algorithm as well as one order of magnitude compared to the *BE* algorithm.

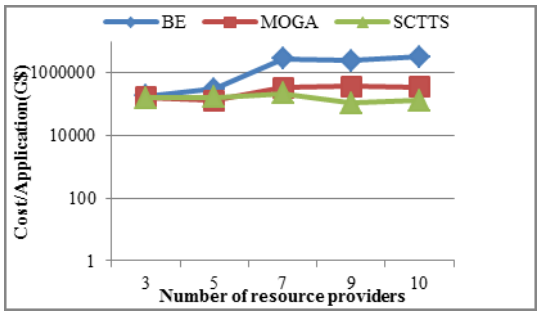


Fig. 10. Impact of the number of resources on workflow allocation cost

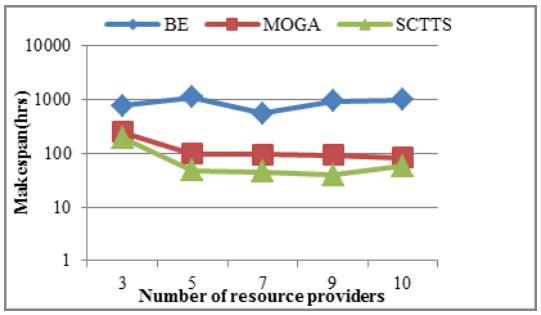


Fig. 11. Impact of the number of resources on workflow makespan

Having concurrently observed **Fig. 10** and **Fig. 11** in corresponding instances, it is understood that in all instances the *SCTTS* algorithm gives a better performance compared to both *MOGA* and *BE* algorithms. Hence, the CTTS algorithm is scalable relative to an increase in the number of the resources. The allocation cost and makespan of the *BE* algorithm rises with a higher rate compared to both *MOGA* and CTTS algorithms according to an increase in the number of the resources. This rise is explained by the fact that the *BE* algorithm selects a resource for running the task with the lowest number of the waiting and running queues. As a result, the higher the number of the resources are, the more the available selections will be. Due to an increase in the number of the resource selections, it is likely that a resource selected for running a task, will be different to the resources which had run the parents' tasks of the task. The explanation is that the *BE* algorithm almost ignores the data transfer minimization. As a result, the algorithm will be exposed to the data transfer cost from the resource which has executed the parent task to the resource which executes the child task. Eventually, there will be an increase in the allocation-cost and makespan.

Fig. 12 shows the impact of the number of the resources on the runtime of the algorithms. In all cases, the runtime of the proposed algorithm is as many as four orders of magnitude

lower than both *MOGA* and *BE* algorithms. The long runtime of the *MOGA* algorithm is explained by the fact that the *MOGA* algorithm is a genetic-based algorithm which generally suffers from the disadvantage of a longer runtime. The longer runtime of the *BE* algorithm explains that each task needs to wait for its parents' tasks to be completed. According to **Fig. 12**, although, the number of the resources increases, the runtime of the *SCTTS* algorithm remains constant. As a result, the *SCTTS* algorithm becomes scalable due to an increase in the heterogeneous resources of the grid environment. Therefore, the *SCTTS* algorithm can schedule the application in a more effective time.

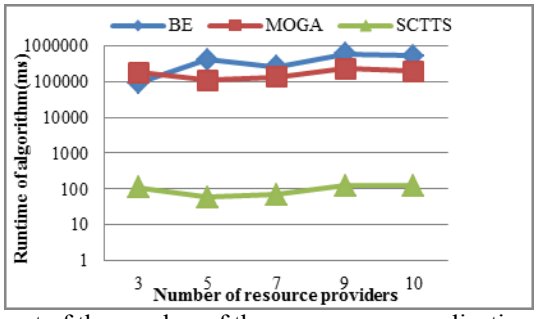


Fig. 12. Impact of the number of the resources on application runtime

6. Conclusion and Future Works

This paper presents both the scalable *SCTT* model and the scalable *SCTTS* heuristic algorithm for scheduling parallel workflow application tasks in Utility Grids while optimizing the allocation cost and makespan. The results of the present study show the *SCTTS* algorithm is scalable caused by an increase in the workflow size, task parallelism and heterogeneous resources.

We also compared the *SCTTS* algorithm with the present best-effort and genetic-based algorithms. We evaluated the sensitivity of the *SCTTS* algorithm by the changes made in the degree of the significance of allocation cost and execution time. Also, the results show the proposed algorithm is scalable caused by an increase in the workflow size, task parallelism and heterogeneous resources with the lowest runtime. We conducted a few experiments on studying the impact of a workflow, task and resource sizes on the cost, makespan and runtime and we found that the proposed algorithm delivers a better performance for scheduling with higher degrees of the task parallelism.

A future extension of the work will consider the effect of uncertainties in the task runtime estimates. In the current model, we have assumed that the workflow tasks runtimes are known accurately. However, the task runtimes might be stochastic. This adds another variable to the problem. Ideally, we intend to provision more resources, since if the slot expires before the task completes, we will lose all the work. As an another extention, we intend to consider makespan, cost reliability, avalability and their combinations as resource utilization.

Appendix 1

To compare our results statistically with other algorithms, we have used paired samples t-test through the SPSS software. Our statistical analysis are shown in **Tables 6, 7** and **8**. In these tables, *** denotes significant difference at 0.1% level, ** denotes significant difference at 5% level and * denotes significant difference at 10% level. The results show that proposed algorithm outperforms other algorithms. It is worth mentioning that these statistical results are

only provided for section 5.1 to statistically show the superiority of the proposed algorithm.

In **Tables 6, 7** and **8** pair 1 is the difference between the cost of *BE* algorithm and the cost of *SCTTS* algorithm, pair 2 is the difference between the cost of *MOGA* algorithm and the cost of *SCTTS* algorithm, pair 3 is the difference between the makespan of *BE* algorithm and the makespan of *SCTTS* algorithm, pair 4 is the difference between the makespan of *MOGA* algorithm and the makespan of *SCTTS* algorithm, pair 5 is the difference between the runtime of BE algorithm and the runtime of *SCTTS* algorithm and pair 6 is the difference between the runtime of *MOGA* algorithm and the runtime of *SCTTS* algorithm.

Table 6. Paired sample t test when trade-off factor is equal with 0.0

Paired Samples Test

		Paired Differences			T	Sig. (2-tailed)
		Mean	95% Confidence Interval of the Difference			
			Lower	Upper		
Pair 1	$Cost_{BE}$ - $Cost_{SCTTS}$	2.16537E6	1.66850E6	2.66225E6	9.858	0.000***
Pair 2	$Cost_{MOGA}$ - $Cost_{SCTTS}$	-23015.29417	-4.20010E5	3.73980E5	-0.131	0.899
Pair 3	$Makespan_{BE}$ - $Makespan_{SCTTS}$	902.31767	836.80795	967.82739	31.158	0.000***
Pair 4	$Makespan_{MOGA}$ - $Makespan_{SCTTS}$	29.09042	8.93158	49.24926	3.264	0.010**
Pair 5	$Runtime_{BE}$ - $Runtime_{SCTTS}$	5.20223E5	4.87467E5	5.52979E5	35.927	0.000***
Pair 6	$Runtime_{MOGA}$ - $Runtime_{SCTTS}$	1.88239E5	1.13513E5	2.62964E5	5.699	0.000***

Table 7. Paired sample t test when trade-off factor is equal with 0.5

Paired Samples Test

		Paired Differences			T	Sig. (2-tailed)
		Mean	95% Confidence Interval of the Difference			
			Lower	Upper		
Pair 1	$Cost_{BE}$ - $Cost_{SCTTS}$	2.97428E6	2.90603E6	3.04253E6	98.582	.000***
Pair 2	$Cost_{MOGA}$ - $Cost_{SCTTS}$	1.13457E5	-8318.57441	2.35232E5	2.108	.064*
Pair 3	$Makespan_{BE}$ - $Makespan_{SCTTS}$	893.91033	835.29830	952.52235	34.501	.000***
Pair 4	$Makespan_{MOGA}$ - $Makespan_{SCTTS}$	26.12825	6.88362	45.37288	3.071	.013**
Pair 5	$Runtime_{BE}$ - $Runtime_{SCTTS}$	5.20195E5	4.87454E5	5.52937E5	35.941	.000***
Pair 6	$Runtime_{MOGA}$ - $Runtime_{SCTTS}$	1.88210E5	1.13489E5	2.62932E5	5.698	.000***

Table 8. Paired sample t test when trade-off factor is equal with 1.0

Paired Samples Test

		Paired Differences			T	Sig. (2-tailed)
		Mean	95% Confidence Interval of the Difference Lower	Upper		
Pair 1	$COST_{BE}$ - $COST_{SCTTS}$	3.09730E6	3.06734E6	3.12727E6	233.830	.000***
Pair 2	$COST_{MOGA}$ - $COST_{SCTTS}$	1.50860E5	63882.40834	2.37838E5	3.924	.003**
Pair 3	$Makespan_{BE}$ - $Makespan_{SCTTS}$	793.92802	686.14602	901.71003	16.663	.000***
Pair 4	$Makespan_{MOGA}$ - $Makespan_{SCTTS}$	-64.22142	-142.39863	13.95580	-1.858	.096*
Pair 5	$Runtime_{BE}$ - $Runtime_{SCTTS}$	5.20251E5	4.87496E5	5.53006E5	35.930	.000***
Pair 6	$Runtime_{MOGA}$ - $Runtime_{SCTTS}$	1.88266E5	1.13528E5	2.63005E5	5.698	.000***

References

[1] T. Eilam, K. Appleby, J. Breh, G. Breiter, H. Daur, S. Fakhouri, et al., "Using a utility computing framework to develop utility systems," *IBM Systems Journal*, vol. 43, pp. 97-120, 2004. Article (CrossRef Link).

[2] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, et al., "Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example," in *Proc. of the Second IEEE international Conference on E-Science and Grid Computing Amsterdam*, Netherlands, 2006. Article (CrossRef Link).

[3] D. S. Katz, J. C. Jacob, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, et al., "A comparison of two methods for building astronomical image mosaics on a grid," in *Proc. of the 34th International Conference on Parallel Processing Workshops (ICPP 2005 Workshops)*, Oslo, Norway, 2005. Article (CrossRef Link).

[4] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, et al., "GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists," in *Proc. of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, UK, 2002. Article (CrossRef Link).

[5] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, pp. 171-200, 2005. Article (CrossRef Link).

[6] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, pp. 237-256, 2009. Article (CrossRef Link).

[7] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on Grids," in *GRID '07 Proc. of the 8th IEEE/ACM International Conference on Grid Austin*, Texas, USA, 2007, pp. 10-17. Article (CrossRef Link).

[8] G. Singh, C. Kesselman, and E. Deelman, "A provisioning model and its comparison with best-effort for performance-cost optimization in grids," in *Proc. of the 16th international symposium on High performance distributed computing*, Monterey, CA, USA, 2007, pp. 117-126. Article (CrossRef Link).

[9] G. Singh, C. Kesselman, and E. Deelman, "An end-to-end framework for provisioning-based resource and application management," *Systems Journal, IEEE*, vol. 3, pp. 25-48, 2009. Article (CrossRef Link).

[10] E. Jeannot, E. Saule, and D. Trystram, "Optimizing performance and reliability on heterogeneous parallel systems: Approximation algorithms and heuristics," *Journal of*

Parallel and Distributed computing, vol. 72, pp. 268-280, 2012. Article (CrossRef Link).

[11] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proc. of the first International Conference on e-Science and Grid Technologies (e-Science 2005)*, Melbourne, Australia, 2005, pp. 140-147. Article (CrossRef Link).

[12] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming,* vol. 14, pp. 217-230, 2006. Article (CrossRef Link).

[13] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos, "Scheduling workflows with budget constraints," *Integrated Research in Grid Computing,* pp. 189-202, 2007. Article (CrossRef Link).

[14] R. Prodan and M. Wieczorek, "Bi-criteria scheduling of scientific grid workflows," *Automation Science and Engineering, IEEE Transactions on,* vol. 7, pp. 364-376, 2010. Article (CrossRef Link).

[15] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer, "Bi-criteria Scheduling of Scientific Workflows for the Grid," in *Proc. of Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, 2008, pp. 9-16. Article (CrossRef Link).

[16] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 23, pp. 1400-1414, 2012. Article (CrossRef Link).

[17] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," in *Proc. of Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 81-88. Article (CrossRef Link).

[18] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, 2007, pp. 280-288. Article (CrossRef Link).

[19] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, "QoS support for time-critical grid workflow applications," in *Proc. of e-Science and Grid Computing, 2005. First International Conference on*, 2005, pp.10 8-115. Article (CrossRef Link).

[20] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade-off management for scheduling parallel applications on Utility Grids," *Future Generation Computer Systems,* vol. 26, pp. 1344-1355, 2010. Article (CrossRef Link).

[21] G. Singh, C. Kesselman, and E. Deelman, "Application-level resource provisioning on the grid," in *Proc. of E-SCIENCE '06 the Second IEEE International Conference on e-Science and Grid Computing* Amsterdam, The Netherlands, 2006, pp. 83-91. Article (CrossRef Link).

[22] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, *et al.*, "Scientific workflow applications on Amazon EC2," 2010, pp. 59-66. Article (CrossRef Link).

[23] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, *et al.*, "Data Sharing Options for Scientific Workflows on Amazon EC2," 2010, pp. 1-9. Article (CrossRef Link).

[24] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences,* vol. 10, pp. 384-393, 1975. Article (CrossRef Link).

[25] D. Feitelson and L. Rudolph, "Parallel job scheduling: Issues and approaches," in *Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing*, Santa Barbara, CA, 1995, pp. 1-18. Article (CrossRef Link).

[26] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Proc. of 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Geneva, Switzerland, 1997, pp. 1-34. Article (CrossRef Link).

[27] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems,* vol. 26, pp. 608-621, 2010. Article (CrossRef Link).

[28] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," *Metaheuristics for Scheduling in Distributed Computing Environments,* vol. 146,

pp. 173-214, 2008. Article (CrossRef Link).
[29] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing,* vol. 61, pp. 810-837, 2001. Article (CrossRef Link).
[30] G. Falzon and M. Li, "Enhancing list scheduling heuristics for dependent job scheduling in grid computing environments," *The Journal of Supercomputing,* vol. 59, pp. 104-130, 2012. Article (CrossRef Link).
[31] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," *Technical Report, Grids-TR-2007-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia,* pp.173-214, 2007. Article (CrossRef Link).
[32] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems,* vol. 13, pp. 260-274, 2002. Article (CrossRef Link).
[33] D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd, "Performance-aware workflow management for Grid computing," *The Computer Journal,* vol. 48, pp. 347-357, 2005. Article (CrossRef Link).
[34] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 5, pp. 113-120, 1994. Article (CrossRef Link).
[35] G. Falzon and M. Li, "Enhancing genetic algorithms for dependent job scheduling in grid computing environments," *The Journal of Supercomputing,* vol. 62, pp. 290-314, 2012. Article (CrossRef Link).
[36] V. Khajevand, H. Pedram, and M. Zandieh, "Provisioning-Based Resource Management for Effective Workflow Scheduling on Utility Grids," in *Proc. of Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, Ottawa, Canada, 2012, pp. 719-720. Article (CrossRef Link).
[37] V. Khajehvand, H. Pedram, and M. Zandieh, "Cost and Makespan Trade-off Management for Scheduling Workflow on Utility Grids," *International Journal of Applied Research on Information Technology and Computing (IJARITAC),* vol. 3, pp. 89-98, 2012. Article (CrossRef Link).
[38] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *Proc. of First International Conference on e-Science and Grid Technologies (e-Science'05)*, Melbourne, Australia, 2005, pp. 140-147. Article (CrossRef Link).
[39] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, "Data management in an international data grid project," in *Proc. of Grid Computing - GRID 2000: First IEEE/ACM International Workshop*, Bangalore, India, 2000, pp. 333-361. Article (CrossRef Link).
[40] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing,* vol. 63, pp. 1105-1122, 2003. Article (CrossRef Link).
[41] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proc. of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, USA, 1993, pp. 416–423. Article (CrossRef Link).
[42] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience,* vol. 14, pp. 1175-1220, 2002. Article (CrossRef Link).
[43] A. Caminero, A. Sulistio, B. Caminero, C. Carrión, and R. Buyya, "Extending GridSim with an architecture for failure detection," 2009, pp. 1-8. Article (CrossRef Link).
[44] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and

simulating data Grids: an extension to GridSim," *Concurrency and Computation: Practice and Experience,* vol. 20, pp. 1591-1609, 2008. Article (CrossRef Link).

Vahid Khajehvand received his B.S. and M.S. degrees in Software Engineering from Qazvin Islamic Azad University (QIAU), in 1999 and 2002, respectively. Currently, he is pursuing his Ph.D. degree in Software Engineering at the same school. His research interests include distributed system, cloud computing, resource management, and workflow scheduling.

Hossein Pedram received his BS degree from Sharif University in 1977 and an MS degree from Ohio State University in 1980, both in Electrical Engineering. He received his PhD degree from Washington State University in 1992 in Computer Engineering. He has served as a faculty member in the Computer Engineering Department of Amirkabir University of Technology since 1992. He teaches courses in Computer architecture and distributed systems. His research interests include innovative methods in computer architecture such as asynchronous circuits, management of computer networks, distributed systems, and robotics.

Mostafa Zandieh accomplished his B.Sc. in Industrial Engineering at Amirkabir University of Technology, Tehran, Iran (1994_1998), and M.Sc. in Industrial Engineering at Sharif University of Technology, Tehran, Iran (1998_2000). He obtained his Ph.D. in IndustrialE ngineering from Amirkabir University of Technology, Tehran, Iran (2000_2006). Currently, he is an Assistant Professor at Industrial Management Department, Shahid Beheshti University, Tehran, Iran. His research interests are Production Planning and Scheduling, Financial Engineering, Quality Engineering, Applied Operations Research, Simulation, and Artificial Intelligence techniques in the areas of manufacturing systems design.