Modeling Workflow of Tasks and Task Interaction Graphs to Schedule on the Cloud

Mahmoud Naghibzadeh
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
Email: naghibzadeh@um.ac.ir

***Abstract*-Many composite computational activities are modeled as directed acyclic graphs called workflows in which each vertex is a task and each directed edge represents both precedence and possible communication from its originating vertex to its ending vertex. When the execution of a task is completed, the communication with its successor(s) starts and anticipated data are transferred. Only after all parents of a task are completed and their results (if any) are received by the task its execution can start. These constraints restrict a more general case in which some tasks could communicate during their executions. In this paper, a task-model composed of both interaction and precedence of tasks is introduced. It is shown that this kind of graph can be transformed into an extended directed acyclic graphs, called hybrid directed acyclic graph, composed of tasks and super-tasks. Super-tasks need not be recognized manually and the proposed method automatically finds them. This can simplify the design of complex workflows. Validity conditions of hybrid directed acyclic graphs are investigated and a verification algorithm is developed. Also, scheduling aspects of hybrid workflows on the cloud is highlighted and some results are reported. This inventive idea can open a whole new area of research and practice in the field of workflow modeling and scheduling.**

Keywords-Task interaction-precedence graph; hybrid DAG; hybrid workflow; Cloud computing.

I. INTRODUCTION

Activities of many composite processes such as likelihood propagation using Bayesian network, family trees (actually graphs) in genealogy, and scientific and industrial workflows, are modeled as *Directed Acyclic Graphs* (DAGs) in which many vertices are connected via directed edges [1]. As the name suggests, an important property of such graphs is that there is no cycle in a DAG. This model is applicable where there is control and data dependencies and a partial precedence relation between tasks and if a task has to transfer data to one (or more) of its successors it can do so at the end of its execution, i.e., in the middle of execution it is not possible for two or more tasks to interact. This resembles an assembly line in which one station has to complete its job and then pass the object to the next station. In any case, a DAG represents a partial ordering of tasks that must be obeyed for their executions. Figure 1 shows a DAG composed of 11 vertices and 15 edges. The number next to a vertex shows its required execution time averaged on all applicable resource types. The real resource type for the execution of each task will be determined during the actual scheduling of the workflow, before the execution of the workflow starts. Consequently, the exact execution time of each task will be computed using the average execution time given on the workflow and the relative processing power of the resource to be used to the average processing power of the resource types. Likewise, the number on an edge shows the average data transfer time from the originating vertex of the edge to the ending vertex of the edge. The exact transfer time will be computed during the scheduling time when the actual resources for source and destination tasks are determined, hence, the communication link and its baud rate is known. The transfer takes place at the end of the execution of the originating task of the edge.

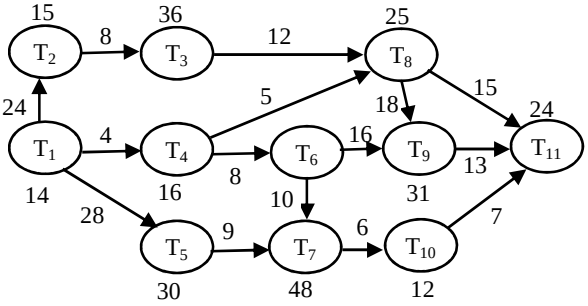

Figure 1. A DAG of precedence-communicating tasks

On the other hand, a Task Interaction Graph (TIG) is used to show which tasks of an application interact during their executions [2]. All tasks of a TIG can start simultaneously and each task can continue its execution so far as it does not need to synchronize or communicate with any other task. Only two directly connected tasks, i.e., neighbors, could interact and it is possible that they may not do so depending on the logic of the corresponding programs and the current values of data objects. Also, even the interaction may be one sided, in some circumstances. The nature of interaction depends on the parallel problem being solved; it may be an actual information transfer or an indirect communication to use a shared data object. For example, suppose a parallel program is designed to multiply a sparse Matrix *M* with *m* rows and *n* columns by vector *V* with *n* rows. Suppose Task *i's* responsibility is pairwise multiplication of elements of Row *i, i=1,2,…m* of Matrix *M* and corresponding elements of Vector *V* and computing their sum, Formula (1),

$$Task_i: \sum_{1 \le j \le n,\ M[i,j] \neq 0} M[i,j] * V[j] \qquad (1)$$

Here, Task *k, k=1,2,..n* and Task *l, l=1,2,…n, l≠k* conflict on elements of Vector *V* for which both *M[k,j]* and *M[l,j]* are nonzero [3]. These two tasks cannot simultaneously access the same memory location. The nature of task interaction in this example is indirect. In the context of scheduling DAGs and TIGs, and now the Task Interaction-Precedence Graph, a task is a piece of work that is completely assigned to one processor to do.

The cloud, provides wide varieties of resources and software in the forms of Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) for public use [4]. Users can lease these services for the needed periods of time and pay as they are used. This is a great opportunity for many users who are short of required resources to run their computational jobs. Scheduling scientific workflows on the cloud is an ongoing research activity with continuous improvements on their quality of services such as make-span, price to be paid, energy efficiency, and fairness. *Fairness* is related to workloads of workflows belonging to one enterprise where it wants to be nondiscriminatory in assigning resources to different workflows [5]. A workflow, if modeled as a DAG, is unable to handle task interactions that can happen any time during their executions. In this paper, a new task modeling approach called *Task Interaction-Precedence Graph* (TIPG) is introduced in which all types of task precedence, dependency, and interaction is possible, subject to passing validity tests. Difficulties involved in the scheduling of applications which can be modeled using TIPG is studied. Some scheduling results are presented which shows the success rate of scheduling is improved. However, the originality of this work is on the introduction of a new task model which allows tasks of a workflow to directly or indirectly interact during their execution.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the problem being studied. Section 4 discusses the validity verification of a given TIPG and its scheduling on the cloud, including the designed algorithms for both verification and scheduling. Section 5 concludes.

II. RELATED WORK

In scheduling workflows on the cloud, we are faced with different quality of services needed by users. Optimal scheduling of a workflow represented by a general DAG is an NP-hard problem [6]. Therefore, different approaches such as integer linear programming [7], genetic algorithms [8], and heuristic algorithms [9], are proposed to produce close to optimal solutions. The objective is often meeting a user-defined deadline, minimizing the computational cost of the workflow, minimizing the make-span, and/or maximizing the success rate of the proposed algorithm. Of course, some algorithms are multi-objective which means that they are designed to optimize more than one objective, such as minimizing timespan and cost at the same time [10]. Here, *make-span* is defined to be the time length from when the workflow is submitted to the cloud to the time when its execution is completed. *Success rate* is the ratio of the number of workflows successfully scheduled to the total number of workflows examined. There are many other aspects to scheduling workflows such as data and computation privacy [11] and simultaneous scheduling of many workflows belonging to one organization [12].

Scheduling TIGs is another field which makes the foundation of this paper's scheduling extended workflows. In a connected TIG, it is not possible to complete one (or many but not all) task at a time but the whole TIG must complete at one time. For example, the whole sparse matrix and vector multiplication is one super-task and computing each element of the resulting vector can be organized as one task. Tasks of such a super-task can simultaneously run on different hardware resources of the cloud. However, it is possible to assign more than one task to one processor to be executed in some specified order. Figure 2 shows a TIG graph for accomplishment of the parallel multiplication of a small-size sparse matrix and a vector. A TIG may represent a complete independent application or it may be part of a larger application. For example, it can be matrix multiplication using matrix-vector product to reduce the number of operations as part of solving a system of linear equations. The latter is widely used in many applications.

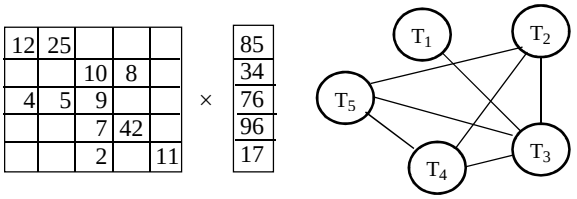

Figure 2. The TIG of matrix-vector multiplication

TIGs are traditionally scheduled on multiprocessors and computer clusters. Nowadays, distributed systems and the clouds provide favorable platforms for solving parallel applications [13]. A TIPG is neither a simple DAG nor a TIG but a new graph model in which both interaction and precedence is allowed. The closest model to this could be a DAG (or workflow) of simple (sequential) tasks and parallel tasks. Nevertheless, there are many differences between the two. (1) Before the production of a DAG of tasks and parallel tasks (which will be called TPDAG for short) parallel tasks of the application being modeled have to be recognized and to be considered as indivisible units, similar to simple tasks within the model. However, a TIPG model starts with (simple) tasks only and there is no composite task in the beginning. Two different types of relations are allowed between each pair of tasks, interaction and communication. Interaction resembles concurrency of tasks and information exchange during execution, while communication resembles precedence and data transfer from one task to the other at the end of execution of the former. The recognition of super-tasks in the TIPG is automatic and without the interference of users. This increases ease of workflow design and at the same time the possibility to recognize parallel tasks in their smallest possible size. (2) Parallel tasks of a TPDAG are co-scheduled to run in parallel, i.e., all subtasks of a parallel task start simultaneously and the whole parallel task

 ISBN: 978-1-61208-460-2

completes at one time. This means, a child of a parallel task cannot start until the whole parallel task is completed and its data (if any) is received [8]. On the other hand, tasks of a super-task within a TIPG can run concurrently, i.e., they can start at different times and complete at different times as long as interactions are possible. Besides, any child of a super-task can start its execution as soon as its parent tasks (not the whole super-task) are completed. This can have a great impact on the time span of the TIPG and as a result on the price to be paid for running the TIPG on the Cloud. (3) Inputs to a parallel task are receive by the parallel task itself and outputs are also sent by the parallel task. For a super-task of a TIPG, any input from external tasks of a super-task is independently received by the receiving task within the super-task and any output from a task within the super-task is sent directly by the sending task within the super-task. As a result, parallel communication from/to tasks of a super-task is the norm. This is another flexibility, which can have a great impact on the quality of scheduling services. Our work could be considered as an extension of workflows of tasks and parallel tasks which have been used by many applications. A well-known such application is Proteomics workflow [14]. Many workflow management software such as Pegasus have the capability to handle parallel tasks.

III. PROBLEM FORMULATION

A directed acyclic graph, when used to model workflows, prohibits tasks to directly or indirectly interact during their executions. We propose an extended model called TIPG which allows interaction, communication, and precedence capabilities, simultaneously.

A TIPG, *G(V, U, D)*, consists of a set of vertices, *V*, a set of undirected edges, *U*, and a set of directed edges, *D*. Each edge, directed or undirected, connects one vertex to another and there could be at the most one edge between any pair of vertices. Note that, the transfer of information between two interacting tasks could occur any time during their executions or even, at the end of the execution of one task it could send information to another task (which is not completed yet) for the last chance. Because we want TIPGs to be extensions of DAGs, considering directed edges only, a TIPG must be acyclic, i.e., it should not be possible to start from some vertex and follow a sequence of directed edges and reach back to the same vertex. For undirected edges only, cycles are not forbidden. For a TIPG graph to be valid it must be acyclic with respect to directed edges and also conflict-free.

Definition 1: A TIPG is *conflict-free* if there is no directed path between any pairs of vertices of any connected component of the TIPG where only undirected edges are considered to obtain these connected components.

Definition 2: In the rest of this paper, wherever we talk about connected components of a TIPG graph *G(V, U, D)* we mean, a sub-graph $G'(V', U') \subseteq G(V, U, D)$, $V' \subseteq V$, *and* $U' \subseteq U$, i.e., U' is composed of only undirected edges of the TIPG.

Definition 2 complies with the classical definition of connected components for undirected graphs when all directed edges of the TIPG are ignored. Recall that, in computer science, for a directed graph, the concept of strongly connected components is used rather than connected components.

If there is any conflict in a TIPG graph it means that there is at least one pair of vertices that one vertex precedes the other and at the same time they can interact during their executions. This is definitely impossible because parallel and precedence constraints may not intermix, hence the design of such a system is incorrect and it should be fixed before going about running it. Therefore, conflict detection, which will be discussed later, could be thought of one type of design verification.

For example, sparse matrix and vector multiplication is seldom a complete application and it can be part of solving a greater problem such as matrix multiplication using matrix-vector product to reduce the number of operations and solving a system of linear equations. Although, in some workflows, cycles composed of two or more tasks are possible, whenever the control flow reaches one of these tasks it should start executing from the beginning. This is not the case for a TIG which is part of a workflow. If an undirected edge connects two vertices v_i and v_j it is represented by either (v_i, v_j) or (v_j, v_i), with no distinction. However, if a directed edge connects two vertices v_k and v_ℓ it is represented by $<v_k, v_\ell>$, where v_k is the starting vertex and v_ℓ is the ending vertex of the edge. A directed edge between a pair of tasks, $<v_k, v_\ell>$, means that the execution of task v_k must precede the execution of task v_ℓ and that v_k can transfer information to v_ℓ once at the end of v_k's execution.

Suppose for the DAG of Figure 1, tasks T_3 and T_4, T_4 and T_5, and T_9 and T_{10} have to interact during their executions. The resulting TIPG is shown in Figure 3. In this figure, all execution and communication times are removed only to increase clarity of the graphical representation of the TIPG, but they are implicitly in place.

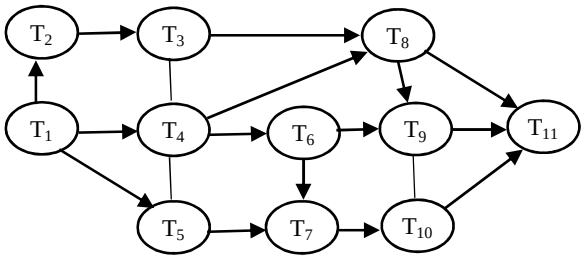

Figure 3. A sample task interaction-precedence graph

Later it will be shown that this TIPG is conflict-free, which means it could be a valid model for tasks of a real application. Such an application cannot be modeled by neither a DAG nor a TIG graph and nor a workflow of tasks and parallel tasks. To represent a TIPG graph an $N \times N$ matrix, *M*, where *N* is the number of vertices, is used. In this matrix, which is usually very sparse, if there is no edge between v_i and v_j, $i, j = 1,2,3,\ldots N$ then $M[i,j] = Null$; if there

is an undirected edge from vertex v_i to vertex v_j then $M[i, j]=0$; otherwise $M[i, j]$ is set to the average data transfer time from v_i to v_j, i.e.,

$$M[i,j] = \begin{cases} null, & if\ no\ edge\ between\ V_i\ and\ V_j \\ 0, & if\ undirected\ edge\ between\ V_i\ and\ V_j \\ data\ transfer\ time\ from\ V_i\ to\ V_j, & otherwise \end{cases}$$

IV. VALIDITY VERIFICATION AND SCHEDULING

In the context of this research a TIPG must pass three test to become a valid *Hybrid Directed Acyclic Graph* (HDAG), i.e., A DAG of tasks and super-tasks. (1) The original TIPG must be acyclic with respect to directed edges, i.e., ignoring all undirected edges. (2) The TIPG must be conflict-free. (3) Considering each connected component as being an indivisible super-task with one entry and one exit, such a graph must be acyclic. Condition 2 is an absolute must satisfy statement, otherwise the model is not valid. Condition 1 is the restriction of our research meaning that, this research aims at eventually scheduling workflows which can be modeled as DAGs and no other models of workflows such as iteration structure. Condition 3 is a complementary to Condition 1.

An algorithm has to be designed to systematically perform these tests. After the first test is successful, it has to find all connected components of the graph considering only undirected edges. A connected component should include at least two vertices. For every pair of vertices of every connected component we have to make sure there is no directed path from one to the other, i.e., considering only directed edges of the TIPG in this stage. This is called conflict-freeness of the TIPG. Using notations of Definition 2, if for any two vertices $(v_i, v_j) \subseteq U'$ there is at least one path, $P \subseteq U'$, from v_i to v_j the TIPG is not conflict-free and hence the model is not valid. Then the whole TIPG is restructured in the form of a directed graph by considering each whole connected component as one indivisible super-task. For this phase, communications to all tasks of a super-task are received by the super-task itself and all communications from the tasks of a super-task are from the super-task itself. The final check is to make sure that the new directed graph of tasks and super-tasks is acyclic. It is worth mentioning that restructuring action is only for the purpose of making sure that there is no cycle in the model. After this checking action is completed, the communications to/from a task within a super-task would directly go to/from the task itself. Algorithm 1 shows the steps to be taken for transformation of a TIPG to a directed graph and verifying its validity as a hybrid DAG. At the start of the algorithm, Matrix *M* is the representation of the TIPG as it is explained earlier. This matrix will be augmented with new rows and columns corresponding to the connected components found. The number of elements in vector *V* is equal to the number of vertices on the original TIPG. At the end, Vector *V* represents which vertex belongs to which component, i.e.,

$$V_i = \begin{cases} 0, if\ vertex\ i\ is\ not\ in\ any\ connected\ component \\ id\ of\ connected\ component, otherwise \end{cases}$$

Since algorithms for finding cycles (if any) in a directed graph (Test 1) is not a new topic, we will assume that the given TIPG graph is already acyclic. Algorithm 1 first produces all connected components and then proceeds with the validity tests. See Figure 4.

```
---------------------------------------------------------------------------
1:Boolean procedure HDAG_Production&Validity(M, V)
2:    while (exists new connected component, C)
3:       Update vector V to represent this component
4:    end while /*from here on components are connected*/
5:    for every (pair of vertices (vi,vj) of every component) do
6:       if (exists a directed path from vi to vj or vj to vi)
7:             return false
8:       end if
9:    end for
10:   for every (component, C) do
11:      add a new row and column to M and
               fill its entries using Rules 2 to 5
12:   end for
13:   for every (vertex v of every component) do
14:      replace all positive values of row v by null
15:      replace all positive values of column v by null
16:   end for
17:   if (cycle (M, V)) return false
18:   else return true
19:   end procedure
---------------------------------------------------------------------------
```

Figure 4. Algorithm 1- producing a potential Hybrid DAG (HDAG) from a TIPG and checking its validity

In Lines 2 to 4 all connected components of the given TIPG are found, one by one. From Line 4 on any reference to component means connected component. Vector *V* is set to represent which tasks of the original TIPG are parts of which component, if any. The conflict-freeness of each connected component is then checked in Lines 5 to 9 and if there is at least one conflict in any connected component there is a design error and the original TIPG must be redesigned. Conflict-freeness of all connected components implies conflict-freeness of the whole TIPG. Each component is called a super-task, in its entirety, is now a new object in the model. To represent connections of each of these new objects with other objects, for every component a new row and a new column is amended to the Matrix *M*, i.e., super-tasks are represented in the same structure where tasks are represented. The values of elements of these rows and columns are filled with respect to communication times between tasks and super-tasks to this super-task (and vice versa) using Rules 2 to 5 that are discussed later. Lines 10 to 12 of the algorithm have this responsibility. There should not be any edge from a task or a super-task outside a given super-task to a task inside this super-task and vice versa. Such edges have to be changed to/from the super-task itself. Lines 13 to 16 are intended to serve this purpose. What remains is that we have to make sure the resulting graph of tasks and super-tasks, where each super-task, as a whole, is considered an indivisible unit of work in this stage, is acyclic. Line 17 will take care of this job and if a violation is diagnosed the graph is not acyclic. Otherwise, the hybrid DAG is valid. In the body of Algorithm 1 the following rules are used.

Rule 1: All vertices and undirected edges of every connected component are separately grouped and encapsulated as a super-task, before going about checking the correctness of the model.

 ISBN: 978-1-61208-460-2

Rule 2: Any data sent from an external task or super-task to a task within a super-task is sent to the super-task itself.
Rule 3: Any data to be sent from a task within a super-task to an external task or super-task is sent by the super-task.
Rule 4: If an external task or super-task wants to send data to more than one task within a super-task the average of the data volumes over all receiving tasks is sent to the super-task. It is assumed that there are independent communication links between each pair of resources. This assumption enables us to consider average of data instead of sum of data. There are two points to be cleared with respect to sending data to more than one task within a super-task: (1) message processing time is negligible compared to message transfer time, hence the sender can handle parallel sends simultaneously, and (2) each receiver task can immediately start its execution after the expected data from its parent(s) is received. These two points justify using the average data volume instead of the sum of data volume on a link which connects an external task or super-task to many tasks within a super-task.
Rule 5: If more than one task within a super-task want to send data to an external task or super-task the average of the data volume is sent by the super-task. The justification for using the average of data volume, instead of the sum of data volume, is similar to the reasoning used for Rule 4.

A. Time complexity of Algorithm 1

The number of operations needed for lines 2 to 4 of Algorithm 1 is proportional to *(N+E), i.e.,* $C_1(N+E)$, where C_1, N, and E, are a constant, the number of vertices, and the number of edges in the original TIPG, respectively. The number of operations for Lines 5 to 9 is proportional to the number of pairs of vertices times *NlogN+E, i.e.,* $C_2N^2(NlogN+E)$. The number of operations for line 17 is $C_3(N+E)$. Therefore, the time complexity of the algorithm is dominated by the time complexity of Lines 5 to 9. Consequently, the time complexity of the algorithm is $O(N^2(NlogN+E))$. For the TIPG of Figure 3, Vector *V* will be filled as in Formula (2).

$$V^{'}=(0,0,0,1,1,1,0,0,0,2,2,0)^{'} \quad (2)$$

There are two connected components, (1) the TIG of tasks *T3*, *T4*, and *T5*, and (2) the TIG of tasks *T9* and *T10*. There are no directed paths from *T3* to *T4*, *T4* to T5, *T3* to *T5*, *T4* to *T3*, *T5* to *T4*, *T5* to *T3*, *T9* to *T10*, or *T10* to *T9*. Therefore, the TIPG of Figure 3 is conflict-free. The algorithm checks and makes sure that there is no cycle in the whole new graph of tasks and super-tasks when each super-task is considered indivisible. The new graph is a valid hybrid workflow. The resulting hybrid DAG is shown in Figure 5.

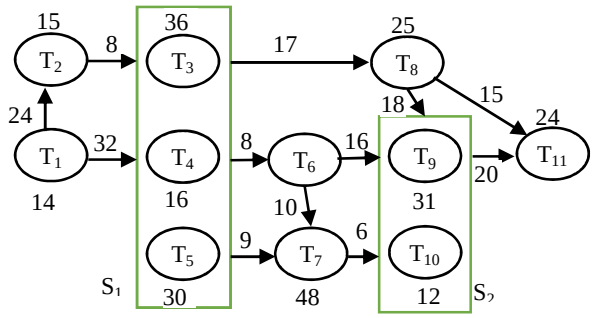

Figure 5. Hybrid DAG obtained from TIPG of Figure 3

An actual implementation of Algorithm 1 could include a preliminary stage of finding the reachability matrix of the graph by considering only directed edges. This can be done using the Floyed-Warshall algorithm [15]. The matrix can be accessed in Line 6 of the algorithm many times. The complexity of the preliminary stage of the algorithm is $O(N^3)$, where N is the number of vertices. This is a lower time complexity than the time complexity of the algorithm which we produced earlier, hence the time complexity of the algorithm is still $O(N^2(NlogN+E))$.

B. Scheduling hybrid workflows

Conventionally, workflow schedulers assume that tasks are non-preemptable hence, each processing resource runs the current task in a single-programming fashion. With the extension of workflows to include super-tasks called TIGs multiprogramming becomes more attractive and beneficial to the cloud users. Tasks of a TIG may have quite different execution times, on the one hand and, they should be co-scheduled to be able to interact during execution. We are faced with three options, (1) allocation of as many processing resources as there are tasks and assigning each task to one resource, (2) assigning all tasks of each TIG to one fast enough processing resource and running then in a multiprogramming fashion, and (3) clustering tasks and allocation of as many processing resources as there are clusters and assigning each cluster to one resource then running each cluster in a multiprogramming fashion.

By selecting Option 1 we are aware that all tasks should start simultaneously in order to be able to interact during execution since otherwise resources' time could be wasted due to tasks wanting to interact with others which are not available. It is not always the case that whenever we want to schedule a TIG in the midst of scheduling a workflow it would be beneficial to lease new resources from the cloud. Furthermore, at the time of assigning a TIG, the finish time of already leased resources may not be the same hence waste of some resource's time is inevitable. Depending on the nature of the application being scheduled, we might want to start a descendent task or super-task when all tasks of a TIG are completed. For this case the scheduler should make use of simultaneous completions of tasks if it is beneficial towards the scheduling objectives. These kind of applications are called parallel TIGs.

Option 2 can be useful in many cases for example, when the objective is to minimize cost of running the workflow on the cloud and the given relative deadline is not very short. A *relative deadline* is a duration of time-interval that is given to a workflow to complete as opposed to an *absolute deadline* which is an exact moment of time before or at which the workflow must be completed. A major benefit of this approach is that the interaction time is negligible because the common main memory could be used for data and results sharing.

Option 3 is applicable when Option 2 is not useable with respect to the scheduling objectives. In any case, we cannot simply say that the time-span of executing a super-task is for example equal to the sum of execution times of its included tasks. It depends on the approach that the TIG

 ISBN: 978-1-61208-460-2

is modeled considering aforementioned models. An estimate of execution time is needed before we can go about designing a hybrid workflow scheduler. Although there are differences between scheduling hybrid workflows and simple workflows, the guidelines are already developed.

To make the paper short, a brief summary of what has been done with respect to scheduling TIPGs and the results obtained up to now follows. The base of our two experiments was the graph of Figure 1. Primarily, Edges $<T_4, T_3>$, $<T_5, T_4>$, and $<T_9, T_{10}>$ were added to the graph. Six hundred TIPGs were randomly generated from the topology of this modified graph. That is, all execution times and communication times were first removed from the graph but vertices and edges were not changed. The execution time of each vertex was then randomly selected from the interval [0, 50] and communication time of each edge was randomly generated from the interval [0, 30]. An integer number between zero to eight (exclusive) was randomly selected and that many vertices of the graph were randomly selected and were changed to undirected edges. If the graph was a valid Hybrid DAG it was selected for scheduling. A list scheduling heuristic with the objective of minimizing timespan [16], while the deadline is respected was developed to schedule the workflow. The deadline parameter was randomly selected within the interval [CPL, 1.4*CPL], where CPL is the Critical Path Length of the workflow. The CPL was calculated in a different way which is usually calculated. All communication times were ignored in the calculation of the critical path because the whole critical path could be assigned to one processor, hence communications are zeroed. This kind of critical path represents the absolute minimum timespan for running the workflow, with regards to the resource type being used. Five types of the cloud processing resources with performances 1.0, 1.2, 1.6, 2.4, and 3.0 with price factors equal to their performances were assumed. The ready task (or super-task) with the highest rank was scheduled next. The two experiments differ in the way a super-task is scheduled. The objective of the experiments was to compare parallel and concurrent execution of super-tasks. See Figure 6 (Algorithm 2.) In this algorithm, *lease_appropriate(1)* will lease one new resource with proper processing power, considering the current state of the scheduler. Similarly, *lease_appropriate(N_i-available)* leases as many as the difference of N_i, i.e., the number of tasks in the i^{th} super-task, and the number of available resources, if N_i is greater than the number of available resources. The procedure *assign_task(task)* assigns the task to the resource which completes it the soonest. On the other hand, *assign_stask(super-task)* assigns all tasks of the super-task to as many processors as needed by the super-task.

Experiment 1 followed Option 1 guideline. For this case, effective execution time of each task is equal to the maximum execution time of the tasks of the super-task. Experiment 2 also followed Option 1 with the possibility of each task of a super-task to complete and sent its data to its children vertices, independently. A simple heuristic (not necessarily optimal) was used to assign tasks within a super-task to resources. If Super-task S_i needed N_i simultaneous resources, N_i resources with earliest availability time were found, first. If there were not enough available resources new resources were leased. Then the task with the longest execution time (with respect to the processor's performance) was assigned to the resource which can complete it the earliest time. This task and the corresponding processor were removed and the process continued until allocation is completed.

```
---------------------------------------------------------------------------
1: Retrieve cloud resources' availability and prices
2: Rank tasks and super-tasks and enqueue (Q)
3: mark the entry task as ready
4: While (exists ready task in Q)
5:     remove (highest ranked ready task T (Q))
5:     case 1: regular task Ti (T, V) //A simple task
6:       begin
7:           lease_appropriate (1)
8:           call assign_task (Ti)
9:           remove father-son links of Ti
10:            if a child becomes orphan mark it ready
11:      end begin
12:    case 2: super-task Si (T, V) // A super-task
13:         begin
14:           if available resources are not enough
15:            lease_appropriate(Ni-available)
16:           end if
17:           call assign_stask(Si)
18:           remove father-son links of Si
19:           if a child becomes orphan mark it ready
20:         end begin
21: end while
22: end algorithm
---------------------------------------------------------------------------
```

Figure 6. Algorithm 2- scheduling hybrid workflows of tasks and TIGs

For these experiments the success rate of the second experiment was 7% higher (with respect to the total number of generated workflows) than that of the first experiment. From the 600 workflows with the assigned deadlines, 276 cases were successfully scheduled in the second experiment whereas 235 were successfully scheduled in the first experiment. The success rate of the second experiment was 0.46 and that of experiment one was approximately 0.39. It is a good indication that TIPGs should have their own schedulers rather than using schedulers of workflows of tasks and parallel tasks. It is worth mentioning that a TIPG can handle both parallel and concurrent super-tasks. Having applied the ideas on small-scale TIPGs it can be extended to large-scale scientific workflows with thousands of tasks and super-tasks. Scheduling, and then running, such workflows requires powerful supercomputers or a Cloud environment composed of thousands of Virtual Machines (VM).

V. SUMMARY AND FUTURE WORK

A new task model called task interaction-precedence graph is presented in this paper. Tasks, in this model, not only can communicate information at the end of their executions but they can also interact during their executions. A validity algorithm is developed to check the correctness of the design of such a model. Furthermore, a procedure for transforming a conflict-free TIPG into a hybrid DAG, i.e., a

 ISBN: 978-1-61208-460-2

DAG composed of simple tasks and super-tasks is established in order to be able to check that the final graph is acyclic. Problems involved in scheduling TIPGs on the cloud are investigated. This new modeling technique allows us to model many more complex applications which were not possible to model using DAGs or TIGs alone. The novelty claim of this paper is in the introduction of a new task model in which precedence, interaction, and communication are all simultaneously possible. Hybrid workflows, Hybrid DAGs, and even *hybrid graphs* become meaningful and very useful. The scheduling algorithm developed in this paper is the first step in this regard and much work has to be done. The experiments of this paper focused on the scheduling aspects of TIPG workflows. It is predictable that starting with a graph in which both directed and undirected edges, i.e., communications and interactions, are allowed and the granularity of all tasks are the same and the recognitions of TIGs are automated, the model designing is simpler and less time consuming. However, the actual performance is left to be evaluated in the future.

REFERENCES

[1] D. C. Kozen, The Design and Analysis of Algorithms. Springer, 1992.

[2] D. L. Long and L. A. Clarke, "Task interaction graphs for concurrency analysis," [1988] Proceedings. Second Work. Softw. Testing, Verif. Anal., 1988.

[3] A. Grama, A. Gupta, G. Karypis, and V. Kumar, "Introduction to Parallel Computing; 2nd Edition," Search, 2003, pp. 856.

[4] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," Futur. Gener. Comput. Syst., vol. 29, no. 1, 2013, pp. 158–169.

[5] L. F. Bittencourt and E. R. M. Madeira, "Towards the Scheduling of Multiple Workflows on Computational Grids," J. Grid Comput., vol. 8, 2010, pp. 419–441.

[6] O. Sinnen, Task Scheduling for Parallel Systems. John Wiley & Sons, 2007.

[7] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Using time discretization to schedule scientific workflows in multiple cloud providers," in IEEE International Conference on Cloud Computing, CLOUD, 2013, pp. 123–130.

[8] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," Sci. Program., vol. 14,, 2006, pp. 217–230.

[9] S. Abrishami and M. Naghibzadeh, "Budget Constrained Scheduling of Grid Workflows Using Partial Critical Paths," in International Conference on Grid Computing and Applications, GCA, 2011, pp. 1o8–114.

[10] E. Juhnke, T. Dörnemann, D. Bock, and B. Freisleben, "Multi-objective scheduling of BPEL workflows in geographically distributed clouds," in Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011, 2011, pp. 412–419.

[11] S. Sharif, J. Taheri, A. Y. Zomaya, and S. Nepal, "MPHC: Preserving Privacy for Workflow Execution in Hybrid Clouds International Conference on Parallel and Distributed Computing," in Applications and Technologies (PDCAT), 2013, pp. 272–280.

[12] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. L. González-García, T. Röblitz, et al., "Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid," Grid Comput., vol. 10, no. 2, 2012, pp. 325–348.

[13] I. D. Falco, U. Scafuri, and E. Tarantino, "Two new fast heuristics for mapping parallel applications on cloud computing," Futur. Gener. Comput. Syst., vol. 37, 2014, pp. 1–13.

[14] G. Mehta, E. Deelman, J. A. Knowles, T. Chen, Y. Wang, J. Vöckler, et al., "Enabling data and compute intensive workflows in bioinformatics," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2012, vol. 7156, LNCS, PART 2, pp. 23–32.

[15] R. W. Floyed, "Algorithm 97: Shortest path," Commun. ACM, vol. 5, no. 6, 1962, pp. 345–345.

[16] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel Distrib. Syst., vol. 13, no. 3,2002, pp. 260–274.

 ISBN: 978-1-61208-460-2