Distributed Workflow Scheduling under Throughput and Budget Constraints in Grid Environments

Fei Cao, Michelle M. Zhu, and Dabin Ding

Department of Computer Science
Southern Illinois University Carbondale
Carbondale IL 62901, USA

Abstract. Grids enable sharing, selection and aggregation of geographically distributed resources among various organizations. They are emerging as promising computing paradigms for resource and compute-intensive scientific workflow applications modeled as Directed Acyclic Graph (DAG) with intricate inter-task dependencies. With the growing popularity of real-time applications, streaming workflows continuously produce large quantity of experimental or simulation datasets, which need to be processed in a timely manner subject to certain performance and resource constraints. However, the heterogeneity and dynamics of Grid resources complicate the scheduling of streaming applications. In addition, the commercialization of Grids as a future trend is calling for policies to take resource cost into account while striving to satisfy the users' Quality of Service (QoS) requirements. In this paper, streaming workflow applications are modeled as DAGs. We formulate scheduling problems with two different objectives in mind, namely either maximize the throughput under a budget/cost constraint or minimize the execution cost under a minimum throughput constraint. Two different algorithms named as Budget constrained RATE (B-RATE) and Budget constrained SWAP (B-SWAP) are developed and evaluated under the first objective; Another two algorithms named as Throughput constrained RATE (TP-RATE) and Throughput constrained SWAP (TP-SWAP) are evaluated under the second objective. Experimental results based on GridSim showed that our algorithms either achieved much lower cost with similar throughput, or higher throughput with similar cost compared with other comparable existing algorithms.

Keywords: streaming workflow; task scheduling; Grid computing; throughput and budget

1 Introduction

Grid computing has emerged as a promising solution for large-scale resource and compute-intensive applications. A wide range of scientific applications can be represented as complex workflows comprised of many computing tasks with inter-task dependencies. Many of the scientific jobs can be modeled as Directed Acyclic Graphs (DAGs) where each vertex represents a computing task and each directed edge represents the execution dependency between adjacent tasks. Scheduling tasks onto heterogeneous and dynamically changing Grid resources needs to respect the precedence constraints and optimize certain criteria based on various user and system situations. We consider the Grid environment as an overlay network consisting of a number of heterogeneous computer nodes interconnected by network links. The network can be modeled as a directed weighted graph which can be complete or not due to network types.

A number of Grid workflow management systems such as Condor DAGMan [1], Pegasus [2], and GridFlow [3], etc. have been developed. These systems provide middleware tools to control the mapping and execution of workflow modules by strategically considering the availability and capacities of the underling Grid resources. However, managing Grid environment to run various jobs is a complex task which requires scheduling policies to reach certain tradeoff due to different requirements from the perspectives of users and various Grid providers usually from different organizations. Existing Grid resource management systems are mainly driven by system-centric policies which aim to optimize a system-wide standard of performance, whereas future Grid environments need to guarantee certain level of Quality of Service (QoS) requirements as well as meet user-centric economic concerns [4]. A number of Grid systems such as Globus [5] have considered some of these multi-objective issues by using resource trading and QoS-based scheduling [4].

In recent years, execution costs on the Grid are being considered by more and more scientists due to the fact that different resources belonging to different organizations may have different allocation/pricing policies on resource usage [4]. The user computing cycle quote/allocation policy can be translated into certain pricing scheme which will be utilized by the scheduler to balance the workload. Such pricing mechanism is widely used by Cloud computing and could be converted to virtual dollars and utilized in the future Grid environment. Therefore, users with budget or quota constraint may not always desire the highest possible QoS such as throughput, i.e., the data production rate at the last task, for a smooth flow in streaming applications with multiple instances of input datasets [6]. Typical examples of these applications include video-based real-time monitoring systems that perform feature extraction and detection, facial reconstruction, pattern recognition, and data mining, etc.

In order to build some theoretical foundations for the future generation of paid Grid, we focus on developing workflow scheduling algorithms considering both budget and throughput constraints. In particular, we consider two different objectives of user requirements. One is to maximize throughput under a budget constraint while another one is to minimize the execution cost under the minimum throughput constraint.

In our approach, we strategically select an appropriate set of heterogeneous Grid resources in an arbitrarily connected network and map each computing task from the workflow to the most appropriate Grid nodes for certain performance criteria. If multiple tasks are mapped onto the same node (i.e., node reuse), the node's computing resource is shared in a fair manner by concurrent tasks executing on that node. Similarly, the bandwidth of a network link is also shared by concurrent data transfers. For budget constrained objective, the B-RATE algorithm adopts a layer-based mapping scheme and assigns partial cost constraint for each layer, then chooses the maximum partial throughput from the first layer to current mapping layer; The B-SWAP algorithm starts with a schedule that is optimized for throughput, and keeps swapping tasks between nodes by choosing those tasks whose cost savings result in the smallest loss in throughput under the budget constraint. To our best knowledge, there is currently no algorithm to maximize the throughput for streaming applications under budget constraint or minimize execution cost under throughput constraint in the Grid. The superiority of these two algorithms are demonstrated in comparison with some representative workflow scheduling algorithms to maximize the throughput including Streamline [7] and LDP [6] in a set of different scales of simulation cases. For throughput constrained objective, the TP-RATE algorithm also follows layer-based scheduling scheme and chooses the minimum partial cost whose partial throughput is larger or equal to the throughput constraint for each layer; the TP-SWAP algorithm starts with the cheapest schedule of tasks onto resources, and keeps swapping tasks that can lead to higher throughput with low cost increase.

This paper is organized as follows: Section 2 gives an overview of related works. Section 3 conducts analytical models and formulates the scheduling problem. In Section 4, the algorithms are described in details. Section 5 presents the performance evaluations. Conclusion can be found in Section 6.

2 Related Works

The optimization problem of scheduling DAG-structured tasks with complex execution dependencies has been studied for years and is known to be NP-complete [7]. Over the years, workflow scheduling problems in heterogeneous environments have attracted many research efforts, among which a significant amount of efforts have been devoted to workflow scheduling in Grid environments under different scheduling and resource constraints. For example, a number of DAG-structured Grid workflow management systems such as Condor DAGMan [1, 8, 9], Globus [5, 10], Pegasus [2, 11] and GridFlow [3] provide tools and infrastructure to control the execution of various workflow applications on the Grid. Condor is a specialized workload management system for compute-intensive jobs [8] and it can be used to serve Grid environment such as Globus Grid [10]. Directed Acyclic Graph Manager (DAGMan) [9] is a meta-scheduler for Condor jobs and manages dependencies between jobs at a higher level than the Condor Scheduler. Pegasus Workflow Management System [11] bridges the scientific domain with the execution environment (e.g., Clusters, Grids, Clouds, etc.) by automatically scheduling and monitoring high-level workflow onto underlying distributed resources. GridFlow includes a user portal and services of both global Grid workflow management and local Grid sub-workflow scheduling. Simulation, execution and monitoring functionalities are provided at the global Grid level, which work on the top of an existing agent-based Grid resource management system [3].

Meanwhile, many performance-driven workflow scheduling algorithms aim to achieve optimal execution performances [12] including the minimum overall execution time, the maximum reliability and throughput for streaming applications. Streamline [7], a workflow scheduler for streaming data, takes dynamic nature of the Grid into account and takes application requirements, constraints, and resource availability into consideration for scheduling decisions. To achieve the reduced overall execution time, Heterogeneous Earliest Finish Time (HEFT) heuristic algorithm [13] was proposed to initially order all the tasks of a workflow in descending order of their upward rank values calculated as the sum of the execution time and the communication time of the tasks. This algorithm is a commonly cited list-scheduling heuristic [14]. In [15], Dongarra et al. discussed the fundamental properties of a good bi-objective scheduling algorithm, and proposed an approximation algorithm namely RHEFT, which is extended from HEFT algorithm [13] by considering the reliability and allows the user to subjectively choose a trade-off between high reliability and low overall execution time. Recursive Critical Path (RCP) algorithm utilizes the dynamic programming strategy and iteratively find critical path to minimize the overall execution time [16]. This algorithm is used as the mapping scheme for the Scientific Workflow Automation and Management Platform (SWAMP) [17] which is a Condor/DAGMan-based workflow system that enables scientists to conveniently assemble, execute, monitor, control, and steer computing workflows in distributed environments via a unified web-based user interface. Experiments show that RCP provides a better mapping performance than the mapping scheme currently employed by the Condor Scheduler [17, 18]. In [19], a new non-critical task mapping approach using A* and Beam Search (BS) algorithms to improve the RCP algorithm was proposed. For high workflow throughput, Gu et al. designed a greedy layer-oriented heuristic workflow mapping scheme (LDP) to identify and minimize the global bottleneck [6]. In [20], the same authors extended the LDP algorithm by taking reliability into account, and further developed a decentralized mapping procedure.

However, the commercialization of Grids requires market-driven strategies while considering users' QoS constraints like deadline and computation cost (budget). Such a guarantee of service is hard to provide in a Grid environment due to its shared, heterogeneous and distributed resources owned by different organizations with their own policies and pricing mechanisms [4]. Many algorithms for deadline and budget constrained scheduling have been proposed [21–27]. In [25], Sakellariou et al. proposed two different approaches, namely the LOSS approach and the GAIN approach to find the schedule for a given DAG-structured workflow and a given set of resources without exceeding the budget and is still optimized for overall execution time. The LOSS approach starts with a schedule that is optimized for overall execution time by using HEFT [14] or

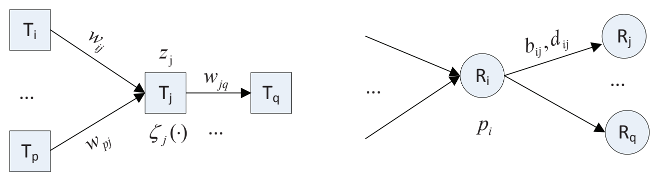


Fig. 1. Workflow model (left), Grid network model (right).

HBMCT [28] and keeps re-mapping as long as the budget is not exceeded. The GAIN approach starts with the cheapest schedule and conducts re-mapping to minimize the overall execution time as long as the budget is still available. In [27], Yu et al. proposed a cost-based workflow scheduling algorithm that minimizes the execution cost for time-critical workflow applications by partitioning workflow tasks and generating schedules based on optimal task partition. It also allows the scheduler to re-compute some partial workflows during execution when their initial schedules are violated. A deadline assignment strategy was developed to distribute the overall deadline over each task partition. Abrishami et al. proposed a QoS-based workflow scheduling algorithm [23] based on the partial critical paths which first tries to map the overall critical path of the workflow such that it completes before the deadline and execution cost can be minimized, then it finds the partial critical path for each mapped task on the critical path and executes the same procedure recursively.

Our work differs from the above mentioned works in several aspects: (i) we consider both throughput and budget requirements; (ii) we consider incomplete Grid environment due to network connectivity and facility accessibility; (iii) we consider resource sharing among multiple concurrent computing tasks on computing nodes or concurrent data transfers over network links.

3 Problem Overview

3.1 Analytical Models

The left side of Fig. 1 shows a workflow of a distributed computing application constructed as directed acyclic graph (DAG) $G_T = (V_T, E_T)$ with $|V_T| = m$. Vertices are used to represent the set of computing tasks $V_T = \{T_1, T_2, ...T_m\}$: T_1 is the starting task and T_m denotes the ending task. The weight w_{ij} on edge e_{ij} represents the size of data transferred from task T_i to task T_j. The dependency between a pair of tasks is shown as a directed edge. Task T_j receives a data input w_{ij} from each of its preceding tasks T_i and performs a predefined computing routine whose complexity is modeled as a function $\zeta_j(\cdot)$ of the total aggregated input data size z_j. However, in real scenario, the complexity of a task is an abstract quantity which not only depends on the computational complexity of its own function but also on the implementation details realized in its algorithm. Upon completion of execution of task T_j, data output w_{jk} will be sent to each

Table 1. Parameter of workflow and Grid network model

Parameters	Definitions
$G_T = (V_T, E_T)$	the computation workflow
m	number of tasks in the workflow
T_i	the i-th computing task
e_{ij}	dependency edge from task T_i to T_j
w_{ij}	data size transferred over dependency edge e_{ij}
z_i	aggregated input data size of task T_i
$\zeta_i(\cdot)$	computational complexity of task T_i
$G_R = (V_R, E_R)$	the Grid network environment
n	number of computing nodes in the Grid environment
R_j	the j-th node
p_j	computing power of node R_j
l_{ij}	network link between nodes R_i and R_j
b_{ij}	bandwidth of link $l_{i,j}$
d_{ij}	the minimum link delay of link $l_{i,j}$
ξ_j	unit executing price of node j (G$/sec)
λ_{ij}	unit executing price of network link l_{ij} (G$/sec)

of its succeeding tasks T_k. A task cannot start its execution until all input data required by this task arrive. To generalize our model, if an application task has multiple starting or ending tasks, a virtual starting or ending task of complexity zero can be created and connected to all starting or ending tasks without any data transfer along the edges.

The right side of Fig. 1 shows a heterogeneous Grid network environment and is represented as an arbitrary weighted network graph $G_R = (V_R, E_R)$ with $|V_R| = n$, consisting of a set of computing nodes $V_R = \{R_1, R_2, ...R_n\}$. Depending on the network infrastructure, the topology of a computer network may be complete or not due to network connectivity and facility accessibility. Resource R_j is featured by its computing power p_j. The network link l_{ij} between resources R_i and R_j is featured by bandwidth b_{ij}, and the minimum link delay d_{ij}. Both nodes and links are considered as Grid resources. The parameters of a workflow are given in Tab. 1.

Inspired by previous work [20], executing a workflow will require the following time and cost:

(1) Execution time of task T_i on node $R_{i'}$

$$t_{exec}(T_i, R_{i'}) = \sum \frac{\alpha(t) \cdot \delta_i(t)}{p_{i'}} \qquad (1)$$

where $\alpha(t)$ denotes the number of concurrent tasks executing on node $R_{i'}$ during Δt, $\delta_i(t) = \frac{p_{i'}}{\alpha(t)} \Delta t$ is the amount of partial task execution completed during time interval $[t, t+\Delta t]$ when $\alpha(t)$ remains unchanged, and $\zeta_i(z_i) = \sum \delta_i(t)$ is the total computational requirement of task T_i.

(2) Data transfer time of dependency edge e_{jk} over network link $l_{j'k'}$

$$t_{tran}(e_{jk}, l_{j'k'}) = \sum \frac{\beta(t)\cdot\delta_{jk}(t)}{b_{j'k'}} + d_{j'k'} \tag{2}$$

where $\beta(t)$ denotes the number of concurrent data transfer over link $l_{j'k'}$ during Δt, $\delta_{jk}(t) = \frac{b_{j'k'}}{\beta(t)}\Delta t$ is the amount of partial data transfer execution completed during time interval $[t, t+\Delta t]$ when $\beta(t)$ remains unchanged, and $w_{jk} = \sum \delta_{jk}(t)$ is the total data transfer size of dependency edge e_{jk}.

(3) Bottleneck time

$$BT = \max_{\substack{T_i \in V_T, e_{jk} \in E_T \\ R_{i'} \in V_R, l_{j'k'} \in E_R}} \begin{pmatrix} t_{exec}(T_i, R'_i), \\ t_{tran}(e_{jk}, l_{j'k'}) \end{pmatrix} \tag{3}$$

(4) Throughput

Throughput is the inverse of the global bottleneck of a mapped workflow in streaming applications where multiple instances of input datasets are continuously generated and fed into the workflow.

$$TP = \frac{1}{BT} \tag{4}$$

(5) Cost of executing task T_i on node R_j

$$C_j(T_i) = \xi_j \times t_{exec}(T_i, R_j) \tag{5}$$

(6) Cost of transfer data of dependency edge e_{jk} over network link $l_{j'k'}$

$$C_{j'k'}(e_{jk}) = \lambda_{jk} \times t_{tran}(e_{jk}, l_{j'k'}) \tag{6}$$

(7) Total execution cost (i.e. user charge) of scheduling a workflow

$$Cost = \sum_{i=1}^{m} C_j(T_i) + \sum_{\forall e_{jk} \in E_T} C_{j'k'}(e_{jk}) \tag{7}$$

3.2 Problem Formulation

The scheduling problem is defined as follows:

Definition 1. *Grid users can submit DAG-structured workflow applications modeled as $G_T = (V_T, E_T)$ that process streaming datasets with both budget and throughput requirements. The budget constrained user aims to maximize the application throughput within their specific budgets:*

$$\max_{all\ possible\ schedules} (TP),\ such\ that\ Cost \leq Budget \tag{8}$$

The throughput constrained user aims to minimize the execution cost while the minimum throughput is guaranteed:

$$\min_{all\ possible\ schedules} (Cost),\ such\ that\ TP \geq TPConst \tag{9}$$

where TP is the throughput, $Cost$ is the user charge, $Budget$ is the budget constraint, and $TPConst$ is the minimum throughput constraint.

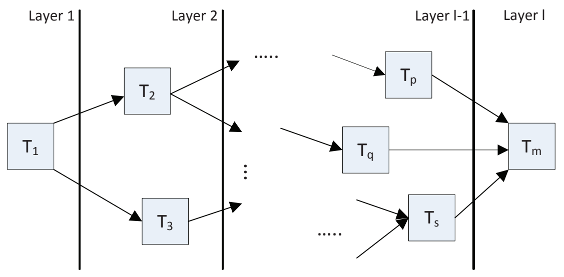


Fig. 2. Layer based sorting of the DAG-structured workflow.

4 Algorithm Design

The following notations are introduced to facilitate the description of our algorithms:

- $pre(T_i)$: the set of preceding tasks of task T_i;
- $V_{one-schedule}(pre(T_i))$: the set of nodes for possible mapping of those tasks in $pre(T_i)$;
- $suc(R_j)$: the set of succeeding nodes of node R_j;
- $\underset{\forall R \in V_{one-schedule}(pre(T_i))}{\cap}(suc(R))$: an intersection operation that finds the set of common succeeding nodes for $V_{one-schedule}(pre(T_i))$;
- $\underset{\forall R \in V_{one-schedule}(pre(T_i))}{\cap}(suc(R)) \cup R$, as the candidate mapping node set for task T_i, denoted as $V_{candidate}(T_i)$;
- $V_{Loss-candidate}(T_i)$: $V_{pre(T_i)} \cap V_{suc(T_i)}$, an intersection operation that casts on the set of nodes that task T_i's predecessor tasks are mapped onto, and the set of nodes that task T_i's successor tasks are mapped onto;
- $V_{Gain-candidate}(T_i)$: $V_{pre(T_i)} \cap V_{suc(T_i)}$, an intersection operation that casts on the set of nodes that task T_i's predecessor tasks are mapped onto, and the set of nodes that task T_i's successor tasks are mapped onto;

4.1 Budget Constrained Approaches

We develop two algorithms, namely B-RATE, B-SWAP for budget constrained users. The purpose of these two algorithms is to find the affordable resources to map workflow tasks in order to achieve the maximum throughput under certain budget constraint.

The *B*-RATE algorithm The B-RATE algorithm in Alg. 1 first separates DAG-structured workflow tasks into ordered layers based on task dependency and node connectivity in the Grid environment as shown in Fig. 2. For each layer k ($k \in [1, MaxLayer]$), we calculate a cost constraint $CostConst_k$ using Eq. 10 where CR is the total computing requirement (i.e., number of instructions) for the entire workflow, and CR_k denotes the partial computing requirement for tasks in layer k:

$$CostConst_k = \frac{CR_k}{CR} * Budget \tag{10}$$

Algorithm 1 B-RATE(G_t,G_n,$Budget$)

Input: Task graph G_t, Grid Resource graph G_n, $Budget$
Output: A workflow schedule that maximizes the throughput under budget constraint.

```
1: for all $T_i \in$ task graph do
2:     Apply layer-based sorting;
3:     Calculate computing requirement for each task;
4: end for
5: Calculate total computing requirement $CR$ for the entire workflow;
6: $MaxLayer$ = the number of total layers in $G_t$;
7: for $k$ = layer 1 to $MaxLayer$ do
8:     Calculate computing requirement $CR_k$ for current layer;
9:     Calculate cost constraint $CostConst_k$ for current layer;
10:    for all task $T_i \in$ current layer do
11:        Find $pre(T_i)$ and $V_{one-schedule}(pre(T_i))$;
12:        Find $V_{candidate}(T_i)$;
13:    end for
14:    Find all possible mapping combinations of $V_{candidate}(T_i)$ for all tasks $T_i$ in current layer;
15:    for all possible mapping combinations do
16:        Calculate $curCost$ for current layer;
17:        if $curCost \leq CostConst_k$ then
18:            Calculate $partialTP$;
19:        else
20:            Continue;
21:        end if
22:    end for
23:    Select the schedule(s) with the maximum $partialTP$, if there're several schedules with the same $partialTP$, choose the one with the minimum $curCost$;
24: end for
25: Calculate total $Cost$;
26: return $TP$, $Cost$;
```

In lines 10-14, for each task T_i in the current layer, we find its preceding tasks $pre(T_i)$ and possible set of their mapping nodes $V_{one-schedule}(pre(T_i))$, then determine the candidate node set $V_{candidate}(T_i)$ for mapping. In lines 15-22, we consider all possible combinations of $V_{candidate}(T_i)$ for all tasks T_i in current layer and calculate their costs. For those possible mapping combinations whose costs are within the cost constraint of current layer, the partial throughput $partialTP$ from the first layer to the current layer is calculated. In line 23, the schedule with the maximum *partialTP* is selected. There might exist several possible schedules with the same throughput, we simply choose the one with the minimum *curCost*. Lines 8-23 are repeated until tasks from the last level are mapped, then we get the throughput and total execution cost. The complexity of this algorithm is $O(mn)$.

Algorithm 2 B-SWAP(G_t,G_n,$Budget$)

Input: Task graph G_t, Grid Resource graph G_n, $Budget$
Output: A workflow schedule that maximizes the throughput under budget constraint.

```
1: for all $T_i \in$ task graph do
2:    Apply layer-based sorting;
3:    Calculate computing requirement for each task;
4: end for
5: $MaxLayer =$ the number of total layers in $G_t$;
6: for $k =$ layer 1 to $MaxLayer$ do
7:    for all task $T_i \in$ current layer do
8:       Find $pre(T_i)$ and $V_{one-schedule}(pre(T_i))$;
9:       Find $V_{candidate}(T_i)$;
10:   end for
11:   Find all possible mapping combinations of $V_{candidate}(T_i)$ for all tasks $T_i$ in current layer;
12:   for all possible mapping combinations do
13:      Calculate $partialTP$;
14:      Select the schedule(s) with the maximum $partialTP$, if there're several schedules with the same $partialTP$, choose the one with the minimum $partialCost$;
15:   end for
16: end for
17: Calculate total $Cost$;
18: while $Cost_{new} > Budget$ && $Cost_{cur} > Cost_{new}$ do
19:    for all $T_i \in$ task graph do
20:       $GenerateLossCandidateSetForEachTask()$;
21:    end for
22:    for all $R_j \in V_{Loss-candidate}(T_i)$ do
23:       Calculate $LossWeight(j)$;
24:    end for
25:    Select the task with the minimum $LossWeight$ to re-map;
26: end while
27: Calculate total $Cost$;
28: return $TP$, $Cost$;
```

The B-SWAP algorithm The B-SWAP algorithm in Alg. 2 starts with identifying an initial schedule (in lines 1-16) which produces the maximum throughput of the entire workflow regardless of the budget (e.g., by using LDP [6]). In lines 18-26, if the available budget is larger or equal to the cost required for this schedule, this schedule can be used right away. However, if the budget is less than the cost of this schedule, swapping operations are invoked. The objective of this algorithm is to re-map those tasks to achieve the minimum loss in throughput for the largest cost savings. Each iteration ends with a reduced total cost with similar throughput. To determine the swapping strategy, $LossWeight$ for task T_i as the iteration loss between the current and new possible mapping schemes onto its candidate nodes in $V_{Loss-candidate}(T_i)$ are computed in Eq. 11:

$$LossWeight(j) = \frac{TP_{Cur} - TP_{New}}{Cost_{Cur} - Cost_{New}} \tag{11}$$

where TP_{Cur} and $Cost_{Cur}$ are the throughput and cost of current schedule, respectively; TP_{New} and $Cost_{New}$ are the throughput and cost of T_i re-mapped onto node R_j which is a candidate node from $V_{Loss-candidate}(T_i)$, respectively. If $Cost_{New}$ is larger than $Cost_{Cur}$, we ignore this candidate node. The algorithm keeps re-mapping by considering the smallest values of $LossWeight$. Our selection criteria of having large cost saving and small throughput loss will result in small value of $LossWeight$. The complexity of this algorithm is $O(mns)$, where s is the number of swaps.

4.2 Throughput Constrained Approaches

We develop two algorithms, namely TP-RATE, TP-SWAP for throughput constrained users. The purpose of this set of algorithms is to satisfy the minimum throughput constraint by finding the best resources that minimizes the execution cost.

The TP-RATE algorithm The TP-RATE algorithm (provided in Alg. 3) applies layer-based sorting to the DAG-structured workflow and then schedule computing tasks to network nodes layer-by-layer. In line 9-17, for each layer, we consider all possible combinations of $V_{candidate}(T_i)$ for all tasks T_i in current layer, calculate their $partialTP$, and calculate $partialCost$ if their $partialTP$ is larger or equal to the throughput constraint $TPConst$. In line 18, the schedule with the minimum $partialCost$ is selected. If there are several possible schedules with the same $partialCost$, we simply choose the one with the minimum $partialTP$. Line 5-19 is repeated until the last task is reached, then we compute the total Cost. The complexity of this algorithm is $O(mn)$.

The TP-SWAP algorithm The TP-SWAP algorithm (provided in Alg. 4) first schedules all the tasks to the cheapest node, there might be several nodes with the same unit cost, then choose the one with the maximum computing power. If the throughput is bigger or equal to required throughput constraint, then this schedule can be used straightaway. In other cases that the throughput is smaller than the constraint, swap is invoked. The objective of this algorithm is to achieve the maximum gain in throughput for the least increase in cost via module remapping. It means that for each re-map, the new schedule's throughput is close

Algorithm 3 TP-RATE(G_t,G_n,$TPConst$)

```
1: for all $T_i \in$ task graph do
2:     Apply lay-based sorting;
3:     $MaxLayer$ = the number of total layers in $G_t$;
4:     for $k$ = layer 1 to $MaxLayer$ do
5:         for all task $T_i \in$ current layer do
6:             Find $pre(T_i)$ and $V_{one-schedule}(pre(T_i))$;
7:             Find $V_{candidate}(T_i)$;
8:         end for
9:         Find all possible schedule combinations of $V_{candidate}(T_i)$ for all tasks $T_i$ in
           current layer;
10:        for all possible schedule combinations do
11:            Calculate $partialTP$;
12:            if $partialTP \geq TPConst$ then
13:                Calculate $partialCost$;
14:            else
15:                Continue;
16:            end if
17:        end for
18:        Select the schedule(s) with the minimum $partialCost$, if there're several sched-
           ules with the same $partialCost$, choose the one with the maximum $partialTP$;
19:    end for
20: end for
21: Calculate total $Cost$;
22: return $TP$, $Cost$;
```

Algorithm 4 TP-SWAP(G_t,G_n,$TPConst$)

```
1: for all $T_i \in$ task graph do
2:     Schedule $T_i$ to the cheapest node, if several nodes have the same unit cost, choose
       the one with the maximum computing power;
3: end for
4: Calculate $TP$, $Cost$;
5: while $curTP < TPConst$ do
6:     for all $T_i \in$ task graph do
7:         $GenerateGainCandidateSetForEachTask()$;
8:         for all $R_j \in V_{Gain-candidate}(T_i)$ do
9:             Calculate $GainWeight$;
10:        end for
11:        Select the task with the maximum $GainWeight$ to re-assign;
12:    end for
13: end while
14: Calculate total $Cost$;
15: return $TP$, $Cost$;
```

to the current schedule but with less increase in cost. To determine such re-map, $GainWeight$ values for each task T_i scheduled to each of its candidate nodes in $V_{Gain-candidate}(T_i)$ are computed as Eq. 12:

$$GainWeight(j) = \frac{TP_{New} - TP_{Cur}}{Cost_{New} - Cost_{Cur}} \quad (12)$$

where TP_{Cur} and $Cost_{Cur}$ are the throughput and cost of current schedule, respectively; TP_{New} and $Cost_{New}$ are the throughput and cost of T_i re-mapped to node R_j which is a candidate node in $V_{Gain-candidate}(T_i)$ for T_i, respectively. The algorithm keeps re-mapping by considering the greatest values of $GainWeight$ for all tasks and their candidate nodes. The complexity of this algorithm is $O(mns)$, where s is the number of swaps.

5 Performance Evaluation

We design and implement our experiments based on the GridSim [29] toolkit. The four algorithms are implemented as four separate schedulers, which can generate scheduling results for given workflows and networks. The workflow tasks are submitted to a Grid resources as advance reservations in GridSim. The cost and throughput are recorded after simulations are finished in GridSim.

5.1 Experimental Settings

Workflow and Grid Network Configurations Given that different workflow applications and networks may have different impact on the performance of the scheduling algorithms, we develop a workflow and network generator which can randomly create varying parameters of the workflows and networks that follows a similar experimental approaches used by some previous published articles [6, 20], and within a suitably selected range of values: (i) the number of tasks and the complexity of each task; (ii) the number of inter-task communications and the data transfer size between two tasks; (iii) the number of nodes and the processing power of each node; (iv) the unit execution price of each node and network link; (v) the number of network links as well as the bandwidth and the minimum link delay of each link.

In our experiments, the cost that a user needs to pay for a workflow execution (i.e. user charge) comprises of two parts, namely cost of executing tasks on nodes, and cost of transfer data of dependency edges over network links.

We represent the problem size in Tab. 2 for workflow scheduling as a four-tuple $(m, |E_T|, n, |E_R|)$: m tasks and $|E_T|$ dependency edges in the workflow, and n nodes with $|E_R|$ links in the network.

Performance Metrics and Experimental Scenarios We consider the two performance metrics of throughput and execution cost, and evaluate our algorithm from the following experimental scenarios:

– Impact of budget constraint

Table 2. Workflow Configurations

Problem Index	Workflow ID	Network ID	Problem Size $m, \lvert E_T \rvert, n, \lvert E_R \rvert$
1	1	1	10, 20, 5, 19
2	2	2	15, 25, 10, 89
3	3	4	20, 42, 15, 209
4	4	4	25, 52, 20, 379
5	5	5	30, 60, 25, 425
6	6	6	35, 72, 30, 630
7	7	7	40, 79, 35, 855
8	8	8	45, 93, 40, 1250
9	9	9	50, 96, 45, 1600
10	10	10	60, 122, 50, 2200

– Impact of throughput constraint
– Impact of workflow size
– Impact of network size

Incomplete network graphs are simulated due to network connectivity and facility accessibility. To conduct thorough comparison, we select different budget constraints and simulate several different sizes of workflows and networks. To set up baselines for comparison, we also developed some representative workflow scheduling algorithms for maximizing throughput (due to no existing algorithm for maximizing throughput under budget constraint) including Streamline [7] and LDP [6] (which is used to find an initial schedule in B-SWAP).

5.2 Analysis of Results

Budget Constrained Approaches In order to compare the performance of the two algorithms for maximizing throughput under budget constraint, namely B-RATE and B-SWAP, we conduct the above-mentioned 10 sets of workflows and networks with problem sizes from small to large and give part of the results in Fig. 3. For each set, various budget constraints are considered. Generally, more budget is provided when problem size becomes larger due to more computation and communication efforts. We calculate the throughput and cost for comparison. The performance of the two proposed algorithms is further compared with Streamline [7] and LDP [6].

Fig. 3 shows the throughput and cost comparison among the four algorithms, the x axis represents the budget constraints; the y axis on the left and the various bars denote the throughput value, the y axis on the right and the lines represent the actual cost of the schedule. The throughput and cost of Streamline and LDP remains constant for each budget constraint as a baseline (since they do not consider budget). We observe that in most cases, B-SWAP results in higher throughput with larger cost than that of B-RATE. This may be due

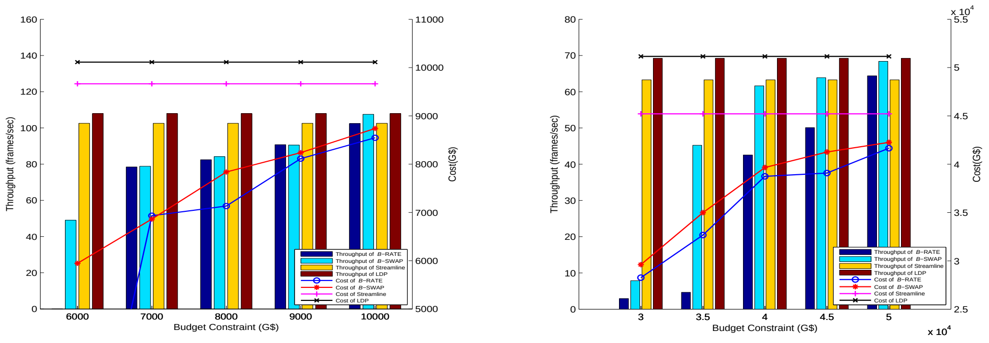
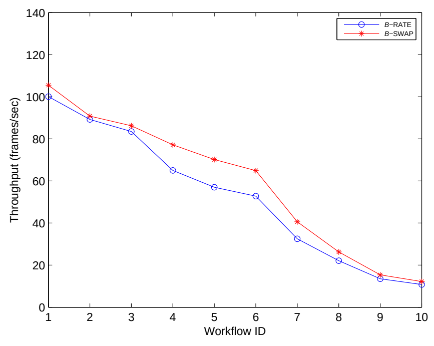


Fig. 3. Throughput and cost comparison under different budget constraints (left: problem index = 2, right: problem index = 6).

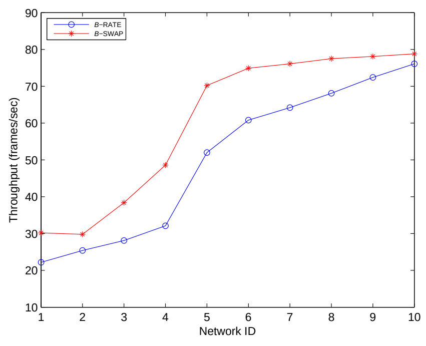
to the fact that B-SWAP starts with a schedule optimized for throughput, then keep re-mapping for the largest savings in cost with the minimum throughput loss; While the B-RATE algorithm starts with a rough and un-precise distribution of budget constraint value for each layer. It is noted that under smaller budget constraints, B-RATE may fail to compute a schedule because the budget constraints for some layers might not be possible under mapping strategy. With larger budget constraints, the two algorithms achieve more similar throughput values due to sufficient budget to play with. In comparison with Streamline and LDP algorithms, since the two algorithms aim for optimization for throughput, we observe that LDP produces the highest throughput with the highest cost though. With the increased budget, B-RATE and B-SWAP are able to achieve comparable throughputs as those from LDP. When budget constraint is set high enough, B-SWAP has the same throughput as that of LDP because no swapping procedure is needed. The costs of B-RATE and B-SWAP are much lower than that of Streamline and LDP even when their throughput values are similar. From Fig. 3, it can be seen that B-RATE's cost decreases by 8%-40% in comparison with Streamline, and decreases by 18%-45% in comparison with LDP; B-SWAP's cost decreases by 7%-35% in comparison with Streamline, and decreases by 17%-40% in comparison with LDP. The variation is due to different problem sizes and budget constraints.

The throughput measurements in Fig. 4(a) shows that B-SWAP consistently achieves higher throughput than B-RATE under the scenario of above-mentioned 10 workflows from small to large executed in the same network with budget constraint set to 80% of LDP's cost. A larger workflow size obviously results in a smaller throughput, which explains the decreasing trend in each curve.

In order to evaluate the impact of network size on the performance of B-RATE and B-SWAP, we compare their throughputs under the scenario of the same workflow executed in the above-mentioned 10 networks from small to large with budget constraint set to 80% of LDP's cost. Fig. 4(b) shows that B-SWAP

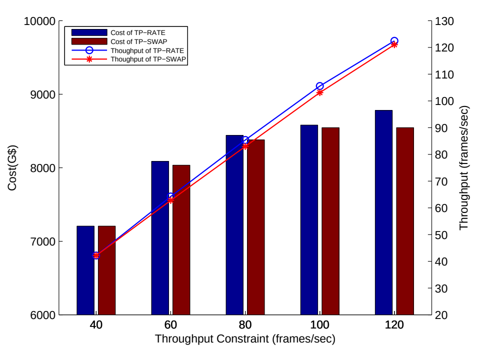


(a) Impact of workflow size (budget constraint = 80% of LDP's cost, network ID = 8)



(b) Impact of network size (budget constraint = 80% of LDP's cost, workflow ID = 5)

Fig. 4. Impact of workflow size and network size



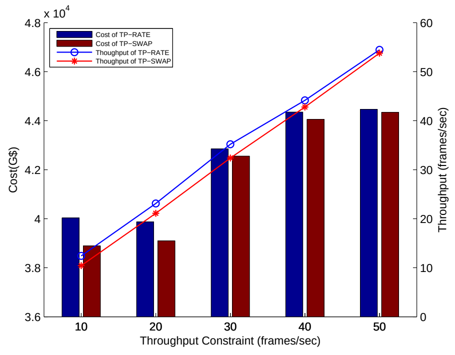
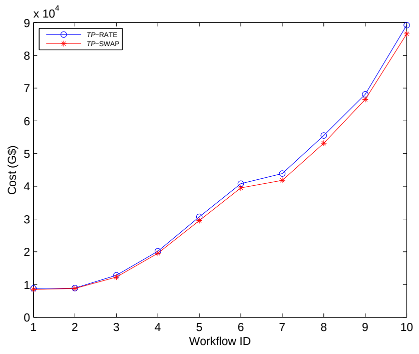


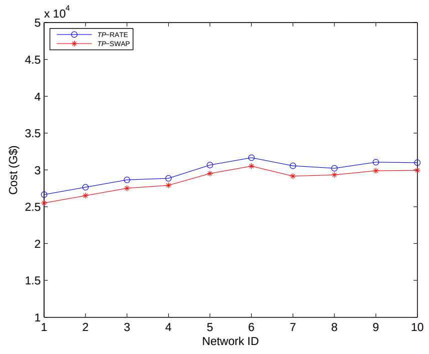
Fig. 5. Cost and throughput comparison under different throughput constraints (left: problem index = 1, right: problem index = 7).

consistently achieves higher throughput than B-RATE. A smaller network size results in a smaller throughput due to higher resource sharing, and the curves increases quickly as network becomes larger, but the increasing trend will slow down after the network is large enough for the workflow.

Throughput Constrained Approaches In order to compare the performance of the two algorithms for minimizing execution cost under throughput constraint, namely TP-RATE and TP-SWAP, we conduct the above-mentioned 10 sets of workflows and networks with problem sizes from small to large and give part of the results in Fig. 5. For each set, various throughput constraints are considered. We calculate the throughput and cost for comparison. To our best knowledge, since no other algorithm considers minimizing execution cost under throughput



(a) Impact of workflow size (throughput constraint = 70% of LDP's throughput, network ID = 8)



(b) Impact of network size (throughput constraint = 70% of LDP's throughput, workflow ID = 5)

Fig. 6. Impact of workflow size and network size

constraint, we only provide performance comparison between the two proposed approaches.

Fig. 5 shows the cost and throughput comparison, the x axis represents the throughput constraints, the y axis on the left and the bars represent the actual cost, the y axis on the right and the lines represent the throughput of the schedule. We observe straightforwardly that cost gets larger when throughput constraint becomes higher. In most cases, TP-SWAP produces lower cost than TP-RATE, but its throughput is relatively smaller. This may be due to the fact that TP-SWAP starts with a greedy schedule optimized for cost, then keep re-mapping for the maximum gain in throughput for the least increase in cost whereas the TP-RATE algorithm has a partial-optimized schedule for each layer that may not be optimal for cost as an entire schedule. Therefore, TP-SWAP is more likely to produce throughput closer to the throughput constraint than TP-RATE. The throughput of the two algorithms gets similar when the throughput constraint becomes higher due to less available resources to produce higher throughput. With larger throughput constraints, the two algorithms achieve more similar costs because higher throughput requirement limits the selection of nodes. From Fig. 5, it can be seen that compares with TP-RATE, TP-SWAP's cost is about 0%-3% lower, and throughput is about 0%-20% smaller. The variation is due to different problem sizes and throughput constraints.

The throughput measurements in Fig. 6(a) shows that TP-SWAP consistently achieves lower cost than TP-RATE under the scenario of above-mentioned 10 workflows from small to large executed in the same network with throughput constraint set to 70% of LDP's throughput. A larger workflow size obviously results in a larger cost, which explains the increasing trend in each curve.

In order to evaluate the impact of network size on the performance of TP-RATE and TP-SWAP, we compare their costs under the scenario of the same workflow executed in the above-mentioned 10 networks from small to large with throughput constraint set to 70% of LDP's throughput. Fig. 6(b) shows that TP-SWAP consistently achieves higher throughput than TP-RATE. A smaller network size results in a relatively smaller cost because higher resource sharing decreases the data transfer cost (since the data transfer costs of adjacent tasks scheduled on the same node are negligible).

6 Conclusions

In this paper, we considered a workflow scheduling problem for streaming applications with budget and throughput requirements for streaming applications in heterogeneous Grid environment. We proposed two algorithms, namely B-RATE and B-SWAP for budget constrained objective, and two algorithms, namely TP-RATE and TP-SWAP for throughput constrained objective. Thorough simulation experiments under GridSim were conducted with randomly generated workflow and Grid network cases. From our simulation experiments, it could be seen that for budget constrained objective, B-SWAP algorithm outperformed the B-RATE algorithm but with a higher complexity. Compared with throughput optimized only algorithms such as Streamline and LDP, our two proposed algorithms achieved much lower execution cost with similar throughput. For throughput constrained objective, TP-SWAP outperformed TP-RATE in execution cost, but with disadvantage of a higher complexity and smaller throughput. In the future, real-life scientific workflows and real Grid networks with more dynamic scenarios for execution of the workflow will be considered.

References

1. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor - A Distributed Job Scheduler. The MIT Press, MA, USA (2002)
2. Blythe, J., Jain, S., Deelman, E., Gi, Y., Vahi, K., Mandal, A., Kennedy, K.: Task Scheduling Strategies for Workflow-based Applications in Grids. In: IEEE International Symposium on Cluster Computing and the Grid (CCGrid). (2005) 759–767
3. Cao, J., Jarvis, S., Saini, S., Nudd, G.: Gridflow:workflow management for grid computing. In: 3rd International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan (2003)
4. Abramson, R.B.D., Venugopal, S.: The Grid Economy. the IEEE **93(3)** (2005) 698–714
5. Foster, I.: Globus Toolkit Version 4: Software for Service-oriented Systems. Journal of Computer Science and Technology (2006) 513–520
6. Gu, Y., Wu, Q.: Maximizing Workflow Throughput for Streaming Applications in Distributed Environments. In: 19th International Conference on Computer Communications and Networks (ICCCN). (2010)
7. Agarwalla, B., Ahmed, N., Hilley, D., Ramachandran, U.: Streamline: A Scheduling Heuristic for Streaming Application on the Grid. In: The 13th Multimedia Computing and Networking Conf. (2007) 69–85

8. Condor. http://research.cs.wisc.edu/htcondor.
9. DAGMan. http://research.cs.wisc.edu/htcondor/dagman/dagman.html.
10. Globus. http://www.globus.org.
11. Deelman, E., Singh, G., Su, M.H., Blythe, J., e. a. Gil, Y.: Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming **13** (2005) 219–237
12. Yu, J., Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid Computing. SIGMOD Record **34(3)** (2005) 44–49
13. Topcuoglu, S., Wu, M.: Task Scheduling Algorithms for Heterogeneous Processors. In: 8th IEEE Heterogeneous Computing Workshop (HCW99). (1999) 3–14
14. Sonmez, O., Yigitbasi, N., Abrishami, S., Iosup, A., Epema, D.: Performance analysis of dynamic workflow scheduling in multicluster grids. In: the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10). (2010)
15. Dongarra, J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. In: the 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '07). (2007) 280–288
16. Wu, Q., Gu, Y.: Supporting Distributed Application Workflows in Heterogeneous Computing Environments. In: 14th International Conference on Parallel and Distributed Systems (ICPADS08). Volume 47. (2008) 8–22
17. Wu, Q., Zhu, M., Lu, X., Brown, P., Lin, Y., Gu, Y., Cao, F., Reuter, M.: Automation and Management of Scientific Workflows in Distributed Network Environments. In: the 6th Int. Workshop on Sys. Man. Tech., Proc., and Serv.,. (2010) 1–8
18. Wu, Q., Zhu, M., Gu, Y., Brown, P., Lu, X., Lin, W., Liu, Y.: A Distributed Workflow Management System with Case Study of Real-life Scientific Applications on Grids. Journal of Grid Computing **10(3)** (2012) 367–393
19. Wu, Q., Gu, Y., Lin, Y., Rao, N.: Latency Modeling and Minimization for Large-scale Scientific Workflows in Distributed Network Environments. In: the 44th Annual Simulation Symposium (ANSS 2011). (2011) 205–212
20. Gu, Y., Wu, Q., Liu, X., Yu, D.: Improving Throughput and Reliability of Distributed Scientific Workflows for Streaming Data Processing. In: The 13th IEEE International Conference on High Performance and Communications (HPCC). (2011) 347–354
21. Yu, J., Buyya, R.: A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. In: Workshop on Workflows in Support of Large-Scale Science (WORKS). (2006) 1–10
22. Yuan, Y., Wang, K., Sun, X., Guo, T.: An Iterative Heuristic for Scheduling Grid Workflows with Budget Constraints. In: International Conference on Machine Learning and Cybernetics. (2009) 1700–1705
23. Abrishami, S., Naghibzadeh, M., Epema, D.: Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths. IEEE Transaction on Parallel and Distributed Systems **23(8)** (2012) 1400–1414
24. Y, Y., Liu, J., Ma, L.: Efficient cost optimization for workflow scheduling on grids. In: International Conference on Management and Service Science (MASS). (2010) 1–4
25. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.: Scheduling Workflows with Budget Constraints. In: Intigrated Research in Grid Computing. (2007) 189–202

26. Yu, J., Buyya, R.: Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms. Scientific Programming **14(3-4)** (2006) 217–230
27. Yu, J., Buyya, R., Tham, C.: Cost-based Scheduling of Scientific Workflow Applications on Utility Grids. In: First International Conference one-Science and Grid Computing. (2005) 139–147
28. Sakellariou, R., Zhao, H.: A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. In: 13th IEEE Heterogeneous Computing Workshop (HCW'04), Santa Fe, New Maxico, USA (2004)
29. Buyya, R., Murshed, M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE **14**(13) (2002) 1175–1220