2012 26th IEEE International Conference on Advanced Information Networking and Applications

ScaleStar: Budget Conscious Scheduling Precedence-Constrained Many-task Workflow Applications in Cloud

Lingfang Zeng, Bharadwaj Veeravalli
Department of Electrical and Computer Engineering
The National University of Singapore
Singapore 117576
{*elezengl,elebv*}*@nus.edu.sg*

Xiaorong Li
Institute of High Performance Computing
*A*STAR*
Singapore 138632
lixr@ihpc.a-star.edu.sg

***Abstract*—Traditionally, the "best effort, cost free" model of Supercomputers/Grids does not consider pricing. Clouds have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on "pay-as-you-go" model. Large scale many-task workflow (MTW) may be suited for execution on Clouds due to its scale-* requirement (scale up, scale out, and scale down). In the context of scheduling, MTW execution cost must be considered based on users' budget constraints. In this paper, we address the problem of scheduling MTW on Clouds and present a budget-conscious scheduling algorithm, referred to as *ScaleStar* (or Scale-*). ScaleStar assigns the selected task to a virtual machine with higher *comparative advantage* which effectively balances the execution time-and-monetary cost goals. In addition, according to the actual charging model, an adjustment policy, refer to as *DeSlack*, is proposed to remove part of slack without adversely affecting the overall makespan and the total monetary cost. We evaluate ScaleStar with an extensive set of simulations and compare with the most popular HEFT-based LOSS3 algorithm and demonstrate the superior performance of ScaleStar.**

***Keywords*-Cloud computing, budget, makespan, workflow, scheduling**

I. Introduction

Cloud creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable and standard worldwide network environment. In addition to this flexibility, Cloud infrastructure services provide the illusion of limitless resource allocation for most users [1]. Cloud Service Providers (CSPs) allow users to either allocate one such machine for many hours or many machines for one hour, while the costs remain the same. Without paying a premium for large scale, this resource elasticity is unprecedented in the history of IT [2].

The various commercial offers differ not only in pricing but also in the types of machines that can be allocated. For example, for Amazon's EC2, the choice is between "Small", "Large", "Extra Large", "High CPU (Medium)", and "High CPU (Large)" instance types, all coming in two to three flavors [3]. The efficient use of these payment-services is a crucial issue because different type machines may have different rent(pricing). Moreover, different services, belonging to different CSPs, may have different policies for charging. Users would like to pay a price which is commensurate the budget they can get. However, currently, the problem of allocating the right number of machines, of the right type, for the right time frame, strongly depends on the application program, and is left to the user.

Given this motivation, we focus on developing an efficient many-task workflow (MTW) scheduling algorithm based on user's time and budget constraints. In general, processing time and economic cost are two typical constraints for MTW execution on Cloud services. In this paper, we present a budget conscious scheduler, which minimizes MTW execution time within a certain budget. The proposed MTW scheduling approach can be used by both end-users and CSPs. End users can use the approach to orchestrated Cloud services, whereas CSPs can outsource computing resources to meet customers' service-level requirements.

The remainder of the paper is organized as follows. We introduce the problem overview and related work in Section 2. In Section 3 the system model and schedule problem definition are provided. In Section 4 we present the proposed scheduling algorithm. In Section 5 we conduct performance evaluation approaches and results. In Section 6 we conclude the paper with directions for further work.

II. Related work and motivation

Since MTW applications in scientific and engineering fields are the most typical application model, much work have been conducted on these before (e.g., [4]–[10], where, HEFT [4] is one of the well-known heuristics for heterogeneous resources). However, most efforts in task scheduling have focused on two issues, the minimization of application completion time (makespan/schedule length) and time complexity. It is only recently that much attention has been paid to economic cost in scheduling, particularly for Grids [11]–[13]. MTWs can be modeled as Directed Acyclic Graphs (DAGs) [14]–[16]. In this model, the existing approaches can address only certain variants of the multi-criteria workflow scheduling problem, usually considering up to two contradicting criteria ([8], [16] and [17] for example) being scheduled in some specific Grid environments.

1550-445X/12 $26.00 © 2012 IEEE
DOI 10.1109/AINA.2012.12

Clouds are emerging as a promising solution for computation and resource demanding application, MTW has the skew of the distribution [18], Clouds' distinctive elasticity feature suits MTW to attain efficient use of the computational resources. Users would like to pay a price which is commensurate to the budget they have available. So, the commercialization of the Cloud requires policies that can take into account user requirements, and budget considerations in particular. Most of existing work in the literature based on "best effort, free access" model Grid or some specific Grid environments, such as dynamic cost model based on Grid Economy and market equilibrium. There has been little work examining issues related to budget constraints in a Cloud context.

Yu et al. [11], [12] provided a workflow scheduling algorithm which minimizes execution monetary cost while meeting users' deadline constraint. Sakellariou et al. [19] developed scheduling approaches, LOSS and GAIN, to adjust a schedule which is generated by a time optimized heuristic and a cost optimized heuristic to meet users' budget constraints respectively. But this strategies should be supported by other scheduling algorithm. BaTS [10], a budget and time-constrained scheduler, can schedule large bags of *independent* tasks onto multiple Clouds. Zhu et al. [20] proposed a dynamic scheduling for fixed time-limit and resource budget constraint tasks. References [12] and [13] focus on using genetic algorithms to solve the scheduling problems considering the budget and deadline of entire network. Recently, HCOC [21] discuss workflow execution in a Cloud context. HCOC reduce monetary cost while achieving the established desired execution time by deciding which resources should be leased from the public Cloud or be used in the private Cloud. Byun et al. [22] provided PBTS (Partitioned Balanced Time Scheduling) which estimates the minimum number of computing hosts required to execute a workflow within a user-specified finish time. However, the computing hosts are used as the same monetary cost per time unit. For large graph processing in Cloud, Li et al. [23] designed a cost-conscious scheduling algorithm (CCSH) which is an extension of HEFT.

In general, the primary performance goal of Supercomputers or Grids has focused on reducing the execution time of applications whilst increasing throughput. This performance goal has been mostly achieved by the development of scale out or scale up cluster or multicluster systems. However, this free access model does not considered pricing. Clouds have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on "pay-as-you-go" usage model. With the development of Clouds, in the future, more and more large scale MTW may be suited for execution on Clouds due to its scale-* requirement (scale up — high performance node, scale out — more nodes for task parallelism, and scale down — short-term usage node of tasks should be released).

III. System Model, Problem Setting

A. System model

Our system model assumes that a physical cluster is built with a number of interconnected servers in a datacenter. Each physical server can be mounted with multiple *virtual machines* (VMs). VMs are logically connected together over several physical clusters. Without any loss of generality, we denote $\mathbb{V}$=v_1,v_2,...,v_V, where, V is the number of VMs, a set of computational VMs participating in a Cloud. Our monetary cost model assumes that all VM reservation cycles, expressed in hours, are identical for all VMs. Each VM $v_i \in \mathbb{V}$ has different VM rents with different configurations; in other words, VM can be chose according user's budget. Then, for each VM $v_i \in \mathbb{V}$, a set $\mathbb{P}$ of p_i the price per hour to execute a task on a VM v_i.

We assume that a MTW application is composed of a number of tasks. Our scheme is to schedule large number of such MTW tasks onto a given Cloud platform. Without loss of generality, a MTW application can be represented by a DAG $\mathbb{G} = (\mathbb{N}, \mathbb{E})$, where $\mathbb{N}$ is a set of tasks (the term task and node have been used interchangeably in the paper), and $\mathbb{E}$ is a set of edges. The edges usually represent precedence constraints: each edge $e_{i,j} = (n_i, n_j) \in \mathbb{E}$ represents a precedence constraint that indicates that task n_i should complete its execution before task n_j starts. $e_{i,j}$ also represents inter-task communication, which is the amount of data (in bytes) that task n_i must send to task n_j in order for task n_j to start its execution. We call n_j as the successor of n_i and n_i as the predecessor of n_j. Note that in addition to data communication itself, overhead may be involved for data redistribution, e.g., when task n_i is executed on a different number of VMs. However, when two tasks are assigned to different VMs, a communication cost is required. We ignore the communication cost when tasks are assigned to the same VM. A task n_i has a weight ω_i ($1 \leq i \leq N$, N is the number of tasks in a MTW), corresponding to the execution time of task n_i on a baseline VM platform, and an edge $e_{i,j}$ ($1 \leq i, j \leq N$) has a weight $\epsilon_{i,j}$, which corresponds to the amount of data to be transmitted from VM v_i to VM v_j.

We assume that $\mathbb{G}$ has a single entry task and a single exit task. A task with no predecessors is called an *entry task*, n_{entry}, whereas an exit task, n_{exit}, is one that does not have any successors. If there is more than one exit (entry) task, they are connected to a zero-cost pseudo exit (entry) task with zero-cost edge, which does not affect the schedule. A task is called a *ready task* if all of its predecessors have been completed. The longest path of a task graph $\mathbb{G}$ is the critical path (CP). The overall execution time of $\mathbb{G}$, or *makespan*, is defined as the time between the beginning of $\mathbb{G}$'s entry task and the completion of $\mathbb{G}$'s exit task.

As defined above, as N is the number of tasks in a MTW, it takes $w_{i,j}$ time units to execute a task n_i on the VM v_j

with price p_j. Therefore, the computation monetary cost of a task n_i on a VM v_j can be calculated as:

$$Cost_i = cost(n_i, v_j) = w_{i,j} \cdot p_j \tag{1}$$

where p_j is the monetary cost per hour to execute a task n_i on a VM v_j, $w_{i,j}$ is the execution time of a task n_i on the VM v_j.

The total computation monetary cost of a MTW $Cost_{comp}$ can be calculated as:

$$Cost_{comp} = \sum_{j=1}^{V}((\sum_{i=1}^{N} w_{i,j}) \cdot p_j) \tag{2}$$

where, $w_{i,j}$ is the execution time of a task n_i on the VM v_j, if the task n_i is assigned on the VM v_j with price p_j, $w_{i,j} = \omega_i$, else, $w_{i,j} = 0$, V is the number of VMs.

Specially, from the view of CSPs, the start time of pricing by a VM begins prior to its being ready and the finish time of charging by that VM is the time at which the last application task in this VM is executed. In practice, if a VM is not released to CSPs then the user will be priced even if the VM is idle. Thus, Eq.(2) can be redefined as follows:

$$Cost_{comp} = \sum_{k=1}^{V}((eft_{n_j,v_k} - est_{n_i,v_k}) \cdot p_k) \tag{3}$$

where eft_{n_j,v_k} is the earliest finish times of the last task n_j in the kth VM with price p_k, est_{n_i,v_k} is the earliest start times of the first task n_i in the kth VM, V is the number of VMs.

The total execution time of the MTW $T_{makespan}$ can be calculated as:

$$T_{makespan} = max\{eft(n_{exit})\} \tag{4}$$

where, *eft* is defined in Eq.(8).

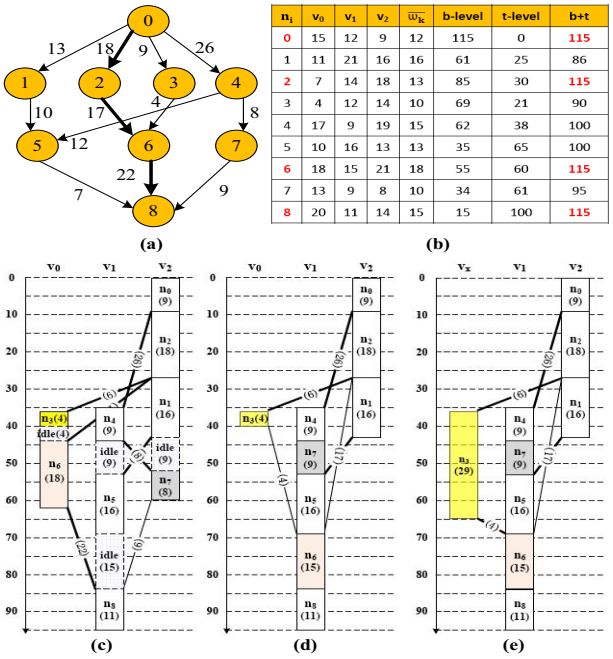
When a MTW is submitted to a Cloud, the scheduler allocates tasks one by one to a VM. Our goal is to model Cloud services considering financial cost and performance constraints.

$$minimize(T_{makespan}), subject\ to, Cost_{comp} + Cost_{tran} + Cost_{stor} + Cost_{spinup} \leq B \tag{5}$$

where B is the user-given constraints on monetary cost, $Cost_{tran}$ denotes the monetary cost of transferring input or output data products, $Cost_{stor}$ denotes the monetary cost of permanent storage for input or output data products, $Cost_{spinup}$ represents the monetary cost of system initialization. For other components in this equation, such as $Cost_{tran}$, $Cost_{stor}$, $Cost_{spinup}$ are also essential for user's economic constraints. Since they are not directly related to the MTW scheduling strategy, we do not give their details as the work presented here only focuses on the workflow tasks scheduling.

B. Many-task workflow scheduling problem

Given a DAG $\mathbb{G}$ and a Cloud system $\mathbb{C}$, the problem addressed is to find a feasible schedule, defined as a mapping from $\mathbb{G}$ onto $\mathbb{C}$: scheduling of a set of interdependent tasks onto a set of heterogeneous VMs dispersed across multiple data centers in a Cloud. A simple task graph is shown in Fig. 1. Usually, *b-level* and *t-level* attributes are used for assigning priority [24]. The *t-level* of a task n_i is the length of a longest path (there can be more than one longest path) from an *entry* task to n_i (excluding n_i). The *b-level* of a task n_i is the length of a longest path from n_i to an *exit* task. The *b-level* and *t-level* parameters are very important for assigning priority to tasks. For example, if we want to schedule tasks in CP first, tasks can be scheduled in a descending order of *b-level*. Consider the DAG shown in Fig. 1(a). In this DAG there exists only one CP which comprises tasks n_0, n_2, n_6, and n_8 (The edges in the CP are shown with thick arrows). Note that these nodes ($\in \mathbb{N}_{\mathbb{CP}}$) have the same biggest value of (*b-level* + *t-level*), as shown in Fig. 1(b).



n_i	v_0	v_1	v_2	$\overline{\omega_k}$	b-level	t-level	b+t
0	15	12	9	12	115	0	115
1	11	21	16	16	61	25	86
2	7	14	18	13	85	30	115
3	4	12	14	10	69	21	90
4	17	9	19	15	62	38	100
5	10	16	13	13	35	65	100
6	18	15	21	18	55	60	115
7	13	9	8	10	34	61	95
8	20	11	14	15	15	100	115

Figure 1. An example of scheduling in a DAG (assume the monetary cost per time unit of v_0, v_1, v_2, and v_x is 0.1, 0.5, 1, and 0.01 respectively): (a) A simple DAG representing precedence-constrained MTW; (b) Execution time of nodes on three different VMs, average execution time, *b-level*, *t-level*, and *b-level*+*t-level* of nodes; (c) HEFT algorithm ($T_{makespan}$=95, $Cost$=92.6 (including idle times), $Cost$=71.2 (excluding idle times)); (d) Budget-conscious algorithm ($T_{makespan}$=95, $Cost$=73.4 (including idle times), $Cost$=73.4 (excluding idle times)); (e) Budget-conscious algorithm ($T_{makespan}$=95, $Cost$=73.04 (including idle times), $Cost$=73.04 (excluding idle times)).

The makespan ($T_{makespan}$) is obtained after the scheduling of N tasks in a task graph $\mathbb{G}$ is completed. Since the two

objectives (minimization of makespan and monetary cost) in our scheduling model conflict with each other, scheduling decisions should be made accounting for the impact of each of those objectives on the quality of schedule. Before scheduling, the tasks are labeled with the average execution times:

$$\overline{\omega_k} = \frac{\sum_{j=1}^{V} w_{k,j}}{V} \tag{6}$$

The earliest start time $est(n_i, v_k)$ and earliest finish time $eft(n_i, v_k)$ of, a task n_i on a VM v_k are defined as:

$$est(n_i, v_k) = \begin{cases} 0 & \text{if } n_i = n_{entry}, \\ max\{avail(k), (max_{n_m \in pred(n_i)} \\ (eft(n_m, v_m) + \epsilon_{i,j} \cdot \lambda_{v_m, v_k}))\} \\ & \text{otherwise.} \end{cases} \tag{7}$$

$$eft(n_i, v_k) = est(n_i, v_k) + \omega_{i,k} \tag{8}$$

where v_k is the VM scheduled to task n_i, v_m is the VM scheduled to task n_m, $pred(n_i)$ is the set of immediate predecessor tasks of task n_i, $avail(k)$ is the earliest time at which VM v_k is ready for task execution, and λ_{v_m, v_k} is the inverse of the bandwidth of the link between VM v_m and VM v_k. If n_m is the last assigned task on VM v_k, then $avail(k)$ is the time that VM v_k completed the execution of the task n_m and it is ready to execute another task when we have a non-insertion-based scheduling policy. The inner *max* block in the *est* equation returns the *ready time*, i.e., the time when all data needed by n_i has arrived at VM v_k.

Traditional studies from the literature aim to assign tasks onto machines in such a way that the overall schedule length is minimized and precedence constraints are met. For an example of a DAG, the makespan produced using HEFT [4], is shown in Fig. 1(c). For the example, from Fig. 1(c) we note that there are two lines — bold line and fine line. Bold line signifies the fact that the earliest start time of a task can not be scheduled in advance if there is a bold line from other task. But, fine line signifies the fact that a task can be scheduled ahead of current time provided it satisfies the time constraint on the line and also if there is not any bold line from other task. In additional, the charging model of CSPs has an effect on calculating the monetary cost of MTW execution. From Fig. 1(c), CSPs charge fees (92.6) from user by lease time, but charge only 71.2 by "using" time. If the former, scheduling algorithm should employ or eliminate the idle time or user should revert the lease machine(s) to CSPs provided the installing machine time is far less than the idle time.

The slack of each task indicates the maximal value that can be added to the execution time of this task without affecting the overall makespan of the schedule. If CSPs charge fees even when a VM is idle, as far as possible, to eliminate idle times, some tasks should be delayed without affecting the schedule's makespan. Considering again the example in Fig. 1(c), the slack of node 8 is 0; the slack of node 6 is also zero (computed as the slack of node 8 plus the spare time between 6 and 8, which is zero). Node 4 has a spare time of 4 with node 6 and a spare time of 44 with node 8 (its two immediate successors in the DAG and the VM where it is executing). Since the slack of both nodes 6 and 8 is 0, and then the slack of node 4 is 4. Indeed, this is the maximal time that the finish time of node 4 can be delayed without affecting the schedule's makespan. In Fig. 1(d), compared with Fig. 1(c), we note that node 6 is reassigned from VM v_0 to VM v_1, the overall makespan remains unaltered while the monetary cost gets reduced. Similarly in Fig. 1(e), the makespan remains unaltered but the monetary cost is reduced further. In this case, the node 3 is assigned to the VM v_x which has a cheaper monetary cost per time unit than VM v_0.

IV. The Proposed Scheduling Algorithm

As discussed in the previous section, the total computation monetary cost of a MTW $Cost_{comp}$ can be calculated either by Eq.(2) or by Eq.(3). Eq.(2) gives an interpretation of charging model from users' viewpoint: pay for the use of VMs which conduce to their task computation. Eq.(3) provides a model which is viewed from CSPs: charge fees such as machine rent until users release/return those resource to CSPs.

It may be noticed that incorporation of budget consumption into task scheduling adds another layer of complexity to this already intricate problem. Unlike real-time systems, MTW applications in our study are not deadline-constrained; this indicates that evaluation of the quality of schedules is not straightforward, rather the quality of schedules should be measured explicitly considering both makespan and monetary cost. The time-cost metric devised and incorporated into ScaleStar effectively deals with this trade-off. Specially, we introduce a novel objective function referred to as *Comparative Advantage* (CA). We describe below the computations involved in determining the value of this function CA. The ScaleStar algorithm is composed of three phases:

A. Task priorities and initial assignment phase

When a task n_i is assigned to a VM v_j with price p_j, we refer to this as an *assignment*. For each assignment under consideration, its $CA1$ value is computed in addition to that of the best assignment seen up to that point of decision making; that is, the latter is recomputed with the current assignment being considered. The actual $CA1$ value computation starts from the second assignment due to the involvement of two assignments in each computation. A positive $CA1$ value indicates the finding of a new best scheduling. For a given task n_i, the $CA1$ value of an assignment of VM v_j (with p_j) with the best assignment

of v' (with p') is defined as:

$$CA1(n_i, v_j, v') = \frac{cost(n_i,v')-cost(n_i,v_j)}{cost(n_i,v_j)} + \frac{\frac{eft(n_i,v')-eft(n_i,v_j)}{eft(n_i,v_j)-est(n_i,v_j)}}{} \quad (9)$$

where $cost(n_i, v_j)$ and $cost(n_i, v')$ are calculated using Eq.(1) and the monetary cost of n_i on v_j (with p_j) and that of n_i on v' (with p'), respectively. Similarly, $eft(n_i, v_j)$, $eft(n_i, v')$, $est(n_i, v_j)$, and $est(n_i, v')$ refer to the earliest finish/start times of the two task-VM allocations. For a given ready task, its $CA1$ value with each pair of VM and price is computed using the current best assignment of VM v' (with p') for that task, and then the best assignment — from which the maximum $CA1$ value is obtained — is selected (steps 3 in Table I).

Table I
ALGORITHM **SCALESTAR**.

Input: A DAG $\mathbb{G} = (\mathbb{N}, \mathbb{E})$ with task execution time and communication. A set $\mathbb{V}$ of V VMs with cost of executing tasks and available Budget B.
Output: A DAG schedule S of $\mathbb{G}$ onto $\mathbb{V}$.
1 Compute *b-level* of $\forall n_i \in \mathbb{N}$,
  sort $\mathbb{N}$ in decreasing order by *b-level* value
2 Build an array $A[N][V]$, where N is the number of tasks, V is the number of VMs
3 For $\forall n_i \in \mathbb{N}$ and $\forall v_j \in \mathbb{V}$
  Compute $CA1(n_i, v_j, v')$ and $CA1(n_i, v', v_j)$ value with v' use Eq.(9)
  If $CA1(n_i, v_j, v') > CA1(n_i, v', v_j)$, Replace v' with v_j, assign n_i on v'
4 Compute $makespan_{old}$ and $cost_{old}$ according to S
5 For $\forall n_i \in \mathbb{N}$ and $\forall v_j \in \mathbb{V}$
  If task n_i is assigned to VM v_j, $A[i][j] = 0$, else,
    Compute $makespan_{new}$ assume n_i is assigned to VM v_j
    Compute $cost_{new}$ assume n_i is assigned to VM v_j
    $A[i][j] = CA2(i, j)$
6 Assign all the non-zero values of A to a set $\mathbb{A}$
7 Sort A in increasing order by $CA2$ value
8 $cost$ = the current total execution monetary cost
9 While ($\mathbb{A}$ is not empty)
  If $(cost > B)$, get the first element value $A[i][j]$ (and remove it) from $\mathbb{A}$
  If $(cost \leq B)$, get the last element value $A[i][j]$ (and remove it) from $\mathbb{A}$
  Reassign task i to VM j and calculate new monetary cost of schedule S
  If $(cost > B)$, invalidate previous reassignment for task i to VM j
10 If $(cost > B)$, use cheapest assignment for S
11 Call *DeSlack* policy

B. Task re-assignment phase

With above initial assignment of the tasks onto VMs we proceed to phase 2 compute the comparative advantage ($CA2$). Here, we compare the current total execution monetary cost with the budget. Let $makespan_{new}$ be the makespan to execute task n_i on the new assigned VM v_j, and let $makespan_{old}$ be the makespan to execute task n_i on the VM assigned by old schedule S. Also, let the cost associated with $makespan_{new}$ be denoted as $cost_{new}$, and let the cost associated with $makespan_{old}$ be denoted as $cost_{old}$, respectively, which are calculated by Eq.(3). Then, CA2 is computed using the following expression given below:

$$CA2(n_i, v_j) = \frac{makespan_{new} - makespan_{old}}{cost_{old} - cost_{new}} \quad (10)$$

From Eq.(10), we compute CA2 for all possible assignments by assigning n_i to all available VMs v_j and we tabulate all the CA2 values. If $cost_{old}$ is less than or equal to $cost_{new}$ the value of $CA2$ is considered zero. If $makespan_{new}$ is greater than $makespan_{old}$ we assign a $CA2$ value of zero. The algorithm keeps trying reassignment by considering the smallest or biggest values of the $CA2$ for all tasks and VMs (step 9 in Table I).

Table II
DESLACK POLICY.

1. Except the entry task and the exit task, compute eft and est for each task and each "idle" according to the schedule S;
2. Except the entry task and the exit task, find the first tasks or the last tasks, form a $Candidate_{task}$ set, and sort $Candidate_{task}$ in decreasing order by the length of execution time $(eft_{task} - est_{task})$;
3. Selected all "idle" and form a $Candidate_{idle}$ set, and sort $Candidate_{idle}$ in increasing order by the length of idle times $(eft_{idle} - est_{idle})$;
4. Compute the total monetary cost, $cost_{old}$;
5. while ($Candidate_{task}$ is not empty) **do** {
  $n \leftarrow$ first task (node) in $Candidate_{task}$;
  while ($Candidate_{idle}$ is not empty) **do** {
    $idl \leftarrow$ first "idle" in $Candidate_{idle}$;
    Compute the total monetary cost, $cost_{new}$, assume task n isassigned to a same VM (for a idle idl);
    if ($est_n \geq est_{idl}$, $\omega_{idl} \geq \omega_n$, $\omega_{idl} - \omega_n$ is the minimum value, and $cost_{new} \leq cost_{old}$) {
      Task n is allocated to the VM where the idle idl is, go to **(1)**; }
    Eliminate the idle idl from the $Candidate_{idle}$; }
  Remove task n from the $Candidate_{task}$; }

C. DeSlack *policy*

The key idea of *DeSlack* (step 11 in Table I) is to recompute the starting time of each node in the schedule S and the slack, in order to make a decision for adjustment. The input of this policy is a schedule S. The objective of the policy is to partially eliminate the idle times (shown as "idle" in the Fig. 1) of the schedule while minimizing the monetary cost.

The DeSlack policy (Table II) maintains a schedule which is based on the construction of the schedule S and keeps track of the tasks which have been assigned a VM. So, we obtain the earliest start/finish time of each task (est and eft). Also, we obtain the earliest start/finish time of each idle time slot. In Table II, est_n and est_{idl} are the earliest

start times of task (n) and idle times (*idl*), respectively. ω_j is the execution time of task (n) on the new selected VM v_j. Note that, task (n) and idle times (*idl*) are in the same VM v_j, and the step (4) in Table II calculates the total monetary cost using Eq.(3).

V. Experiments and Results

In this section, we compare the performance of the ScaleStar algorithm with LOSS3 [19]. Because LOSS3 must start with an initial assignment of the tasks onto VMs, we use HEFT [4] to generate the initial schedule.

A. Experimental Settings and Comparison Metrics

We evaluate our algorithm rigorously against complex workflows with various structures in order to cover the characteristics of all MTW applications. We used the procedure described in [25] to generated workflows. There are some parameters to define a workflow: number of tasks N, number of edges, range of computing time, range of VM requirements for an P-task, and percentage of P-task. For each type of workflow, we performed simulations using different parallelism factor α and communication to computation ratios (CCRs). Every set of the above parameters are used to generate several random graphs in order to avoid scattering effects. The results presented are the average of the results obtained for these graphs (Average of 100 random MTW applications with 30000 tasks).

In Fig. 3, each result graph shows 2 sets of bars. Each set presents results for different budget B values, varying from $1.1 \times OC$ to $1.4 \times OC$, where OC (see Eq.(11)) is the optimum monetary cost when resource allocation/release is fully elastic. Hence, the monetary cost is exactly proportional to the resource time used by a MTW application. The total optimum monetary cost of a MTW application OC can be calculated as:

$$OC = \sum_{i=1}^{N} (et_{n_i} \cdot count(n_i) \cdot p_h) \tag{11}$$

where, et_{n_i} is the execution time of a task n_i, typically, the execution of a task consists of three phases, downloading of input data from the storage system, running the task, and transferring output data to the storage system. If a task n_i simultaneously occupies throughout its execution, for example, a parallel processing task such as an MPI-task requires a fixed number of fully-connected computing CPUs at the same time, the required VMs are viewed as a special "virtual machine" with an equal price of all required VMs of the task. In this paper, we name a parallel processing task as an *P-task*. $count(n_i)$ is the number of required type of physical host h for task n_i, p_h is the monetary cost per hour of host h. This optimal monetary cost is the lower bound of cost for estimation algorithms. Because the ScaleStar aims to minimize the monetary cost by estimating the resource capacity for workflows.

In this study, because we consider budget constraint as another equally important performance measure, besides *makespan*, we also provide other two important metrics: BSR and AND, introduced below.

Budget Spend Ratio (BSR): For a given task graph, we normalize both its makespan and monetary cost to lower bounds — the makespan and monetary cost of the tasks along the critical path (CP) without considering communication costs.

$$BSR = \frac{Cost_{comp}}{\sum_{n_i \in \mathbb{N}_{\mathbb{CP}}} min_{v_j \in \mathbb{V}} \{w_{i,j} \cdot p_j\}} \tag{12}$$

where, $Cost_{comp}$ is calculated by Eq.(3), $\mathbb{N}_{\mathbb{CP}}$ is a set of critical path tasks of $\mathbb{G}$, $\mathbb{V}$ is the set of VMs, task n_i with execution time $w_{i,j}$ is assigned to VM v_j with price p_j.

Average Normalized Difference (AND) [19] was used as another performance metric for our comparison:

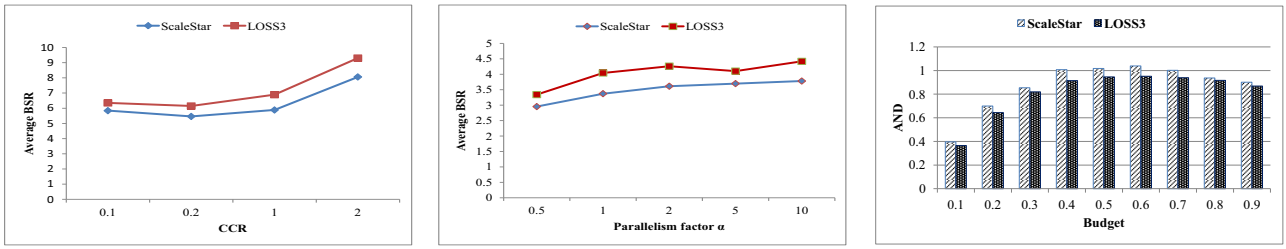
$$AND_x = \frac{1}{R_x} \sum_{i=1}^{R_x} \left(\frac{M_x^i - M_{cheapest}^i}{M_{base}^i - M_{cheapest}^i}\right) \tag{13}$$

where $M_{cheapest}$ is the makespan of the cheapest assignment, M_x is the makespan returned by our algorithm, or LOSS3 [19], M_{base} is the shortest makespan using HEFT [4], the superscript i denotes the i-th run. Because both ScaleStar and HEFT are greedy heuristics, occasional values which are better than the values obtained by those two heuristics may occur. Hence, for each case we take R_x runs. In each case, we considered nine values for the possible budget, consider values of budget that lie in ten equally distanced points between the monetary cost for the cheapest assignment and the monetary cost for the schedule generated by HEFT. Values for budget outside those two ends are looked as be trivial to handle since they indicate that either there is no solution satisfying the given budget, or both algorithms can provide a solution within the budget.

B. Results

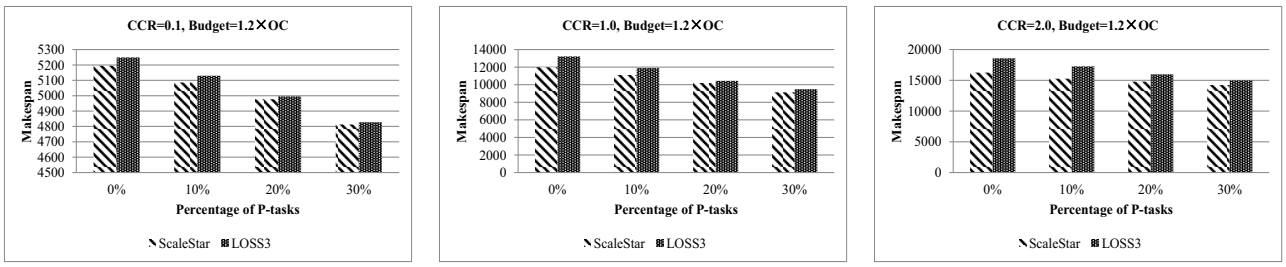
The overall comparative results from our evaluation study are showed in Fig.2 which clearly signify the superior performance of our algorithms over LOSS3, irrespective of different CCR, α, and budget types.

In Fig. 2(a), because smaller BSR is better, note that when task graph is communication intensive (CCR=2.0), the monetary cost of schedules generated by LOSS3 is comparable; this is due to good makespans resulted from only focusing on selected VMs' performance and not considering monetary cost. The comparison between ScaleStar and LOSS3 reconfirmed the favorable performance of our algorithms particularly in terms of monetary cost. The results reported in Fig. 2(b) reveal that ScaleStar outperforms the LOSS3 in terms of the BSR. As the parallelism factor α increases, improvement evince significantly. This is mainly because high parallel application can be exploited by scheduling algorithm. Results showing the AND for each different type



(a) Average *BSR* for *CCR* (b) Average *BSR* for α (c) *AND*

Figure 2. ScaleStar VS. LOSS3.



(a) Low communication (b) Medium communication (c) High communication

Figure 3. Comparison of the makespan of ScaleStar and LOSS3.(a) CCR=0.1; (b) CCR=1.0; (c) CCR=2.0.

of budget available are presented in Fig. 2(c) which shows the difference of the two approaches. The ScaleStar have a generally better makespan than the LOSS3.

Thus, the ScaleStar is over the baseline performance of HEFT, and the LOSS3 is capable of performing close to the baseline performance of HEFT for different values of the budget. This is due to the fact that, for both ScaleStar and LOSS3, the starting basis is a DAG scheduling heuristic, which already produces a short makespan.

The above experiments indicate that the algorithm proposed in this paper is able to find affordable assignments with better makespan when the ScaleStar approach is applied, instead with the LOSS3 approach. The ScaleStar approach applies *CA* to an assignment, moreover in the ScaleStar approach the *DeSlack* policy is used to partly eliminate the idle times; this may reduce the monetary cost. However, in cases where the available budget is close to the cheapest budget, both SclaeStar and LOSS3 give worse makespan than that close to the money cost for the schedule generated by HEFT. This observation can contribute to the optimization in the performance of the ScaleStar algorithm.

Fig. 3 shows the average makespan of 100 synthetic workflows with different percentage of P-task. The results show that ScaleStar is consistently better than LOSS3 up to 14.3%. Moreover, under the same CCR value, ScaleStar is better than LOSS3 and has far more superiorities when the percentage of P-task is smaller. And, for different CCR values, ScaleStar is still better than LOSS3 and has far more superiorities when the CCR value is higher.

VI. Conclusion

In this paper we addressed certain important issues related to handling large-scale MTW scheduling problem on Clouds. Our study made an important contribution that is timely and useful in handling MTW on Clouds. Specifically, we address the problem of scheduling MTW application on Clouds and present a budget-conscious scheduling algorithms, which is cared by both CSPs and users. Towards this end, we had proposed a budget conscious scheduler that is shown to improve the overall makespan and resource utilization of MTW applications in Cloud. Our extensive simulation studies reveal that our *ScaleStar* shows considerable improvement in makespan under various situations such as communication-intensive workflows, workflows with wider degree of parallelism etc. Most commercial cloud vendors have a minimum billable unit (time) and VM has spin-up costs (time). As a future research direction, it will be interesting to extend our approach to account for the impact of deploy/scale on the overall cost effectiveness. Also, it will be interesting to study the dynamic scheduling uncertainties, such as performance drops on the Cloud VMs or unknown task execution times and task size.

Acknowledgment

This work is supported by A*STAR SERC, Singapore, under grant project “Reliable and Adaptive MapReduce-Enabled Cloud Platform for Workflow Data Analytics Ser-

vices (DVCS)", $102-158-0036$. The authors are grateful to the anonymous reviewers for their valuable comments that helped in improving the paper.

REFERENCES

[1] J. Kupferman, J. Silverman, P. Jara, and J. Browne. (2009) Scaling into the cloud. [Online]. Available: http://cs.ucsb.edu/jkupferman/docs/ScalingIntoTheClouds.pdf

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010.

[3] Amazon.com. (2010) Amazon elastic compute cloud. [Online]. Available: http://aws.amazon.com/ec2/

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[5] H. Zhao and R. Sakellariou, "Scheduling multiple DAGs onto heterogeneous systems," in *Proc. 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[6] L. Ramakrishnan and B. Plale, "A multi-dimensional classification model for workflow characteristics," in *Workflow Approaches to New Data-centric Science, with ACM SIGMOD*, 2010, pp. 1–12.

[7] S. X. Sun, Q. Zeng, and H. Wang, "Process-mining-based workflow model fragmentation for distributed execution," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 41, no. 2, pp. 294–310, 2011.

[8] R. Prodan and M. Wieczorek, "Bi-criteria scheduling of scientific grid workflows," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 364–376, 2010.

[9] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Proc. 10th IEEE/ACM Int Grid Computing Conf*, 2009, pp. 50–57.

[10] A. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Proc. IEEE Second Int Cloud Computing Technology and Science Conf*, 2010, pp. 351–359.

[11] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proc. of First International Conference on e-Science and Grid Computing*, 2005.

[12] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proc. Workshop Workflows in Support of Large-Scale Science*, 2006, pp. 1–10.

[13] G. Gharooni-fard, F. Moein-darbari, H. Deldari, and A. Morvaridi, "Scheduling of scientific workflows using a chaos-genetic algorithm," *International Conference on Computational Science (ICCS)*, vol. 1, no. 1, pp. 1445–1454, May 2010.

[14] J. Ding, Y. Wang, J. Le, and Y. Jin, "Dynamic scheduling for workflow applications over virtualized optical networks," in *IEEE Conference on Computer Communications Workshops*, 2011, pp. 127–132.

[15] P.-F. Dutot, T. N'Takpe, F. Suter, and H. Casanova, "Scheduling parallel task graphs on (almost) homogeneous multicluster platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 7, pp. 940–952, 2009.

[16] Y. C. Lee and A. Y. Zomaya, "On effective slack reclamation in task scheduling for energy reduction," *Journal of Information Processing Systems*, vol. 5, no. 4, pp. 175–186, 2009.

[17] Y. C. Lee, R. Subrata, and A. Y. Zomaya, "On the performance of a dual-objective optimization model for workflow applications on grid platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 9, pp. 1273–1284, 2009.

[18] T. G. Armstrong, Z. Zhang, D. S. Katz, M. Wilde, and I. T. Foster, "Scheduling many-task workloads on supercomputers: Dealing with trailing tasks," in *Proc. IEEE Workshop Many-Task Computing Grids and Supercomputers*, 2010, pp. 1–10.

[19] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *in Integrated Research in Grid Computing*, ser. CoreGrid, S. Gorlatch and M. Danelutto, Eds. Pisa, Italy: Springer-Verlag, Nov. 2007.

[20] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, 2010, pp. 304–307.

[21] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[22] E.-K. Byuna, Y.-S. Keeb, J.-S. Kimc, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011–1026, Oct. 2011.

[23] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang, "Cost-conscious scheduling for large graph processing in the cloud," in *Proc. of the 13th International Conference on High Performance Computing and Communications (HPCC)*, 2011, pp. 808–813.

[24] Y. kwong Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[25] X. Tang, K. Li, Z. Zeng, and B. Veeravalli, "A novel security-driven scheduling algorithm for precedence constrained tasks in heterogeneous distributed systems," *IEEE Transactions on computers*, vol. 60, no. 7, pp. 1017–1029, July 2011.