PAPER

Dynamic Scheduling of Workflow for Makespan and Robustness Improvement in the IaaS Cloud

Haiou JIANG[†a)], ***Student Member***, **Haihong E**[†b)], ***and*** **Meina SONG**[†c)], ***Nonmembers***

SUMMARY The Infrastructure-as-a-Service (IaaS) cloud is attracting applications due to the scalability, dynamic resource provision, and pay-as-you-go cost model. Scheduling scientific workflow in the IaaS cloud is faced with uncertainties like resource performance variations and unknown failures. A schedule is said to be robust if it is able to absorb some degree of the uncertainties during the workflow execution. In this paper, we propose a novel workflow scheduling algorithm called Dynamic Earliest-Finish-Time (DEFT) in the IaaS cloud improving both makespan and robustness. DEFT is a dynamic scheduling containing a set of list scheduling loops invoked when some tasks complete successfully and release resources. In each loop, unscheduled tasks are ranked, a best virtual machine (VM) with minimum estimated earliest finish time for each task is selected. A task is scheduled only when all its parents complete, and the selected best VM is ready. Intermediate data is sent from the finished task to each of its child and the selected best VM before the child is scheduled. Experiments show that DEFT can produce shorter makespans with larger robustness than existing typical list and dynamic scheduling algorithms in the IaaS cloud.

key words: *dynamic workflow scheduling, the IaaS cloud, makespan, robustness*

1. Introduction

Over the decades, cloud computing has emerged as a new distributed computing system due to features like application scalability, heterogeneous resources, dynamic resource provisioning, and pay-as-you-go cost model. In the IaaS model of cloud computing, resources are provided in the form of VMs, which are dynamically managed, monitored, maintained, and governed by market principles. Applications can benefit from virtually unlimited resources with minimum hardware investment. As a result, scientific workflows, which are usually modeled as Directed Acyclic Graphs (DAG) with data dependencies among tasks and present applications of high throughput, computation, and complex large scale data analysis, are increasingly adopting cloud computing.

Workflow scheduling is regarded as one of the major challenges in scientific workflow management in the IaaS cloud. Scheduling is defined as mapping tasks to the computing resources by optimizing performance metrics such as execution time and cost [1]. Problem of workflow scheduling usually is NP-hard. Various heuristic and meta-heuristic approaches have been developed to solve the workflow scheduling in the heterogeneous computing system, Grids, and Cloud [1]–[4]. The majority of these approaches can produce good schedules given that the state of resources is static. However, in the IaaS cloud, there are uncertainties a scheduler needs to deal with. Resource performance variations affect the overall workflow performances. If a resource slows down, the tasks scheduled and queued on that resource have to wait to be executed, consequently impact the start time of other dependent tasks and increase the overall makespan. Failures caused by hardware failures, software failures, interferences between cohosted VMs [5], etc. are also inevitable in such large complex distributed systems and may affect the overall workflow performances. Dynamic scheduling can deal with the uncertainties by scheduling a task only when it becomes ready, namely, when all the tasks that the current task depends upon have completed their execution. It considers runtime state of the system and is adaptive in nature.

In this paper, we present a dynamic workflow scheduling algorithm in the IaaS cloud, called Dynamic Earliest-Finish-Time (DEFT). DEFT contains a set of list scheduling loops, which are invoked when some tasks complete and release resources. In each loop, the unscheduled tasks are sorted, a VM with minimum estimated earliest finish time is selected for each unscheduled task. Ready tasks are scheduled, while other tasks keep the VM selection records, and intermediate data is sent from finished tasks to their children according to the task-VM record. Coefficient of Variation (CV) is used in this paper for the robustness to measure how DEFT will schedule workflows with stable makespans in the IaaS cloud with uncertainties. Experiment results show that DEFT not only minimizes the workflow makespans but also produces schedules with larger robustness against uncertainties in the IaaS cloud.

The remainder structure of the paper is as follows. Section 2 discusses the related work. Section 3 describes the models of the system, problem definitions and attributes, and the measurement of robustness. Section 4 presents the details of DEFT, discusses the complexity of the algorithm, and illustrates the algorithm with an example. The simulation results will be presented in Sect. 5. And Sect. 6 gives the conclusion and future work.

2. Related Work

The DAG-based workflow scheduling problem keeps evolv-

Manuscript received August 14, 2016.
Manuscript revised November 29, 2016.
Manuscript publicized January 13, 2017.

[†]The authors are with Beijing University of Posts and Telecommunications, China.
a) E-mail: seagullwill@bupt.edu.cn (Corresponding author)
b) E-mail: ehaihong@bupt.edu.cn
c) E-mail: mnsong@bupt.edu.cn

DOI: 10.1587/transinf.2016EDP7346

ing with computational platforms, from the age of homogeneous systems, to heterogeneous systems, Grids, and Cloud [7]. Due to its NP-hard nature, most of algorithms are heuristic based and can be classified into three categories: list algorithms, clustering algorithms and task duplication based algorithms. List scheduling algorithms are generally preferred since they generate good quality schedules with less complexity [7]. HEFT (Heterogeneous Earliest Finish Time) [6], CPOP (Critical Path on a Processor) [6], PETS (low complexity Performance Effective Task Scheduling) [7], HCPT (Heterogeneous Critical Parent Trees) [8], HPS (High-Performance task Scheduling) [9], Lookahead [10], SDBATS (Standard Deviation-Based Algorithm for Task Scheduling) [11], and PEFT (Predict Earliest Finish Time algorithm) [12] are typical list algorithms. HEFT, CPOP, HCPT, SDBATS, Lookahead and PEFT have two phases, including a task prioritizing phase to decide scheduling sequence of tasks, and a processor selection phase to assign a processor to each task sequentially for a minimum makespan. HPS and PETS have a level sorting phase to sort tasks into groups that can be executed in parallel before the task prioritizing and the processor selection phase. HEFT is well-known and frequently referenced with its lower time complexity, and PEFT is the latest excellent DAG scheduling algorithm with the same time complexity as HEFT. The above algorithms perform well in the static environment supposing that the resource is always available and the performance is static.

In practice, especially in dynamic IaaS cloud, such static schedulers seldom lead to good performances since the actual task execution time at runtime usually deviate a lot from their static predictions estimated before the workflow execution. Some static DAG schedulers try to deal with the unpredictability by assuming the resource computing rate or task execution time variation to follow a specific distribution. H. Arabnejad and J. Barbosa [13] model the computing rate variation of each leased VM by introducing a degradation percentage based on a normal distribution. X. Tang et al. [14] assume the processing time and the communication time to be independent and follow the exponential distribution or the normal distribution estimated by building a historic table and using statistical profiling. D. Poola et al. [15] model the task execution time variation as a normal distribution with a mean value of zero. However, it is difficult to specify analytically the exact distribution of the VM computing rate or the task execution time in the IaaS cloud. W. Zheng and R. Sakellariou [16], [17] solve the problem by using a non-analytical Monte Carlo method. The method first samples from the input space consisting of the random task execution time predictions of the given application, then generates a long list of different static schedules by employing a specific static scheduling heuristic. The schedule with minimum mean makespan of all the input samples is selected. The result of the algorithm is affected by the range of sample space, the number of samples, and the number of static schedules.

Some dynamic scheduling algorithms are developed dealing with uncertainties by scheduling or rescheduling a task in the runtime. Q. Zheng [18] proposes an offline scheduler and makes adaptations during runtime based on the current information about the task start and completion times. It alleviates the chain effect caused by task overruns by rescheduling tasks to faster resources if the predecessors are likely to increase the makespan. However, it does not reschedule tasks to faster resources for smaller makespans since the rescheduling only happens after the overruns. P-HEFT (HEFT for parallel tasks) [19] applies dynamic scheduling for a batch of jobs described DAG with different arrival times. In each scheduling loop, ready tasks of different jobs are scheduled in parallel using an adaptation of HEFT. DCP-G [20] dynamically schedules ready tasks sequentially by identifying DCP (Dynamic Critical Path) of the unscheduled tasks in each scheduling loop, and selects the resource providing the minimum execution time for the task in Grids. The problem of P-HEFT and DCP-G is, all the intermediate data that should be sent as soon as a task is finished is delayed until its child is scheduled to a resource. In DEFT, intermediate data is sent when a task completes and its child has a record of a best VM. Waiting time for intermediate data will be diminished and the overall makespan will be minimized.

There is no consensus on a good definition of robustness for a schedule. Some definitions are related to the workflow deadline. L. Boloni and D. C. Marinescu [21] define the slack of a task as the amount of time that can be delayed without delaying the deadline of the workflow and define the slack of a schedule as the sum of the slacks of all the tasks of the workflow. D. Poola et al. [15] define robustness as 1) the likelihood of the workflow to finish before the given deadline or 2) the amount of time a workflow can be delayed without violating the deadline constraint. Z. Shi et al. [22] give two definitions of robustness based on tardiness of deadline and deadline miss rate. Other definitions are related to makespan, measuring the stability of the makespan for any realization of the same schedule. L. C. Canon et al. [23] measure the robustness by scheduling tasks of a DAG to a target environment several times and computing the makespan distribution. Makespan standard deviation, makespan differential entropy [21], probabilistic that the makespan is within two bounds [24], and the worst makespan it is possible to have in 99% of cases are defined and compared based on the makespan distribution. We use the same measurement by scheduling tasks of a DAG to a target environment several times and see how stable the makespans are. However, we use CV to measure robustness that is more representative and easy to compute.

3. Scheduling Problem Formulation

This section illustrates how we model the IaaS cloud, the workflow, and the scheduling problem. Then the robustness is defined.

3.1 Basic Models and Attributes

In the IaaS cloud, resources are provided by the IaaS cloud service providers to the users in the form of VMs. The configuration of VM differs in memory, CPU measured in million instructions per second (MIPS) and OS. When a scientific workflow is submitted to the cloud, certain type of VMs are provided as a resource pool according to the user's requirement or the workflow configuration. Performances of the network and VMs of each type are considered heterogeneous.

Scientific workflow is usually modeled as a DAG with data dependencies among tasks. The DAG of workflow is represented by $G = \{V, E\}$, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set of n tasks of a workflow, also called nodes in this paper, and $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$ is the data dependencies between these tasks. An edge $(v_i, v_j) \in E$ shows the precedence constraint that task v_i should complete its execution before task v_j starts, v_i is called the parent of v_j, donated as $parent(v_j)$, and v_j is called the child of v_i, donated as $child(v_i)$. The task without any parent is called the entry task and the task without any child is called the exit task. All the nodes from the entry nodes to the parents of v_i are the predecessors of v_i, and all the nodes from the children of v_i to the exit nodes are the successors of v_i.

There are some attributes used in the workflow scheduling we will refer to in this paper.

1) Computing cost is the estimated execution time to complete a task v_i on VM_j at time t, donated as w^t_{ij}. Average computing cost of task v_i on M VMs of the requested type at time t is defined as:

$$\bar{w}^t_i = \sum_{j=1}^{M} w^t_{ij} \Big/ M = \sum_{j=1}^{M} (L_i / R^t_j) \Big/ M \tag{1}$$

Where L_i is the length of task v_i in million instructions, and the R^t_j is computing rate of VM_j at time t.

2) Communication cost c^t_{ij} is the average estimated time spent on transferring intermediate data from task v_i to task v_j based on the network performance at time t, defined as:

$$c^t_{ij} = D + data_{ij} / B^t_{ij} \tag{2}$$

Where D is the average latency of data transfer and B^t_{ij} is data transfer rate between VM^t_i hosting task v_i and VM^t_j hosting task v_j at time t, $data_{ij}$ is the amount of the intermediate data that task v_i sends to task v_j. Note that $c^t_{ij} = 0$ if tasks v_i and v_j are assigned to the same VM or two VMs cohosted on the same server.

3) Makespan, which is also called schedule length in some research [6], [7], [12], of a workflow is the time span from the workflow submission to the completion of the last task of the workflow.

$$makespan = \max\{FT(n_{exit})\} - ST \tag{3}$$

Where ST is the submission time of the workflow and $FT(v_i)$ is the finish time of task v_i. Where there is more than one exit tasks, the makespan is the maximum finish time of all the exit tasks.

Makespan is normalized for comparison. Normalized makespan, called NSL (Normalized Schedule Length) for short, is defined as follows:

$$NSL = makespan / L_{CP_{min}} \tag{4}$$

$$L_{CP_{min}} = \sum_{v_i \in CP_{min}} \min_{VM_j \in VM} \{w_{ij}\} + \sum_{\substack{v_i \in CP_{min} \\ v_k \in CP_{min} \\ v_k \in child(v_i)}} (data_{ik} / \max_{\substack{VM_j \in VM \\ VM_l \in VM}} B_{jl}) \tag{5}$$

$L_{CP_{min}}$ is the theoretical minimum makespan of the workflow, which is the minimum length of critical path (CP) of DAG. CP is the longest path from the entry node to the exit node. The computation cost of each task is calculated by executing it on the fastest VM, and the communication cost is calculated by setting the data transfer rate to the maximum bandwidth of the network. Real makespan cannot be shorter than $L_{CP_{min}}$, and the NSL is always larger than 1.0.

4) Earliest Start Time (EST) of task v_i on VM_j at time t is defined as:

$$EST(v_i, VM_j; t) = \max\{Avail^t_j, \max_{v_p \in parent(v_i)} (FT(v_p) + c^{FT(v_p)}_{pi})\} \tag{6}$$

Where $FT(v_i)$ is the finish time of task v_i, and $Avail^t_j$ is the earliest time when VM_j is available and ready to execute task v_i from time t. Equation (6) means that a task starts when all its parents have finished, all the data is transferred, and the VM is ready to execute it. For the entry node $EST(v_i, VM_j; ST) = ST$, where Where ST is the submission time of the workflow.

5) Data Latest Transfer Time (DLTT) is the latest time that the parent v_p executed on VM^t_p should send intermediate data to the child v_i assigned to VM^t_i at time t in case of not delaying the EST of v_i. It is the EST of v_i subtracted by the data transfer cost from VM^t_p to VM^t_i, shown as follows:

$$DLTT(v_i, VM^t_i, v_p, VM^t_p; t) = EST(v_i, VM_i; t) - c^t_{pi} \tag{7}$$

6) Earliest Finish Time (EFT) of task v_i on VM_j at time t is calculated as the EST of a task v_i plus the computing cost of v_i on VM_j at time t:

$$EFT(v_i, VM_j; t) = EST(v_i, VM_j; t) + w^t_{ij} \tag{8}$$

3.2 Measurement of Robustness

Robustness of a schedule is the stability of the makespans for any realization of the same schedule. Schedule a workflow in the dynamic IaaS cloud for N times, M_i is the

makespan of the i^{th} scheduling result, CV (Coefficient of Variations) is used to measure the robustness of the scheduler in this paper, defined as the ratio of the Standard Deviation σ to the average makespan μ:

$$CV = \frac{\sigma}{\mu}$$
$$\mu = \frac{1}{N}\sum_{i=1}^{N} M_i, \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(M_i - \mu)^2} \quad (9)$$

CV shows the extent of variability of actual makespans in relation to the average makespan. Obviously, a smaller CV means that the makespans are more likely to be close to the average value, showing a more stable distribution of the makespans. The smaller the CV is, the more robust the scheduler is. CV performs over Standard Deviation in that it does not correlate to the average value and is easy to compute as well.

4. The Proposed DEFT Scheduling Algorithm

Static scheduling has obvious shortage in the dynamic IaaS cloud with uncertainties. It schedules all the tasks on the VMs at the start of the workflow. VM computing rate and network bandwidth may change in the runtime. When a VM slows down, tasks scheduled and queued on it may be delayed, consequently impact the start time of other dependent tasks and increases the overall makespans. Failed tasks may also delay successors as well as tasks queued after the failed tasks on the same VMs. In this section, we introduce a novel scheduling algorithm for the IaaS cloud with uncertainties, called Dynamic Earliest-Finish-Time algorithm (DEFT).

4.1 DEFT

DEFT contains a set of loops invoked whenever some tasks complete and release VMs. DEFT is based on a heuristic that if tasks considered in each loop can gain minimum makespans based on the current computing and network performance, makespan of the whole workflow can be minimized. Each scheduling scheduling loop is list based, and contains three phases, namely, tasks prioritizing, VM selection, data transfer phase.

Task prioritizing phase. In each scheduling loop, unscheduled tasks are prioritized using $rank_u$. It is calculated based on the current VM computing rate and network bandwidth. $rank_u$ of task v_i at time t, donated as $rank_u(v_i; t)$, is defined as follows:

$$rank_u(v_i; t) = \bar{w}_i^t + \max_{v_j \in child(v_i)} \left[rank_u(v_j; t) + c_{ij}^t \right] \quad (10)$$

where $\bar{w}_i^t$ is the current average computing cost of v_i, and c_{ij}^t is the current communication cost from v_i to its child v_j. For the exit task, $rank_u(v_{exit}; t) = 0$.

VM Selection Phase. Tasks are sorted by the non-increasing order of $rank_u$ values, and the task with the higher $rank_u$ is given the higher priority. Task with the higher priority can first select and record the "best" VM that allows for the minimum EFT according to Eq. (8). Insertion policy is used that tries to insert a task in the earliest idle time slot between two already recorded tasks on a VM, if the slot is large enough to accommodate the task.

A task is ready if all its parents are finished. When the best VM is available, the ready task can be scheduled and executed on it. Remaining tasks, including tasks which are not ready and ready tasks which are not scheduled just save the (*task*, *bestVM*) record. In the next scheduling loop, if the best VM changed based on the new runtime performance, including VM available time, task finish time, VM computing rate, network bandwidth, etc., the (*task*, *bestVM*) record is updated.

Data Transfer Phase. For a finished task v_i, intermediate data from v_i to each of its children v_j is sent at time DLTT. DLTT is calculated based on current network bandwidth and latency between VM_i hosting v_i and VM_j recorded as the best VM for v_j, according to Eq. (7). At each scheduling loop, record (*time*, *sourceTask*, *sourceVM*, *destinationTask*, *destinationVM*, *data*) is saved, where time is $DLTT(v_j, VM_j, v_i, VM_i; t)$, source task is v_i, source VM is VM_i, destination task is v_j, destination VM is VM_j, and data is the intermediate data from v_i to v_j. Data is sent according to the DLTT record in the runtime. If the (*task*, *bestVM*) record of v_j is changed after data is sent, old data is discarded and new data is sent at the new DLTT. If the new DLTT is earlier than the current time, v_j will be delayed, and $EST(v_j)$ is updated as well. When a VM receives some data, it checks the (*task*, *bestVM*) record list and the DLTT record list. The data is cached if the VM is the the destination, otherwise, the data is discarded. Considering the (*task*, *bestVM*) record may change before a task is scheduled, and the task may fail before complete, DLTT record will not be deleted until the task complete successfully.

In data transfer phase, DLTT is calculated for all the children of finished tasks. Best VMs for the children are selected in VM selection phase. Since tasks that are not the children of the finished tasks can neither be ready nor be assigned to the earlier idle time slot before the children, there is no use to consider them. Therefore, tasks considered in each scheduling loop are the unscheduled children of the finished tasks.

Initially, all the entry nodes are ready to schedule and stored in *Scope*. A scheduling loop is invoked at the beginning or when some tasks complete and release VMs. All the unscheduled children of finished tasks are added to *Scope* (Line 3). Then $rank_u$ of each task in *Scope* is calculated based on the current computing rate and network bandwidth according to Eq. (10) (Line 4-5). *Readys* is updated by adding tasks whose parents are all finished (Line 6-7). Then tasks are sorted by the non-increasing order of $rank_u$ (Line 9). Best VM is selected for each task in the sorted *Scope* (Line 11). New (*task*, *bestVM*) record is add to *Maps* (Line 12-13). *Maps* and *Datas* are updated if the (*task*, *bestVM*) record is changed in the current loop

Algorithm 1 Pseudo code of the DEFT algorithm

```
Input: The workflow DAG with n tasks {v_1, v_2, ..., v_n}
       VM = {VM_1, VM_2, ..., VM_m}: m VM in the IaaS cloud
       Fins: tasks finished before each loop
       Scope: list of tasks considered in each loop, initially is {v_entrance}
       Readys: list of ready tasks, initially is {v_entrance}
       Maps: list of (task, bestVM) map record
       Datas: list of data transfer record
 1:  while Scope is not null do
 2:   if Fins is not null at time t do
 3:     update Scope
 4:     for task v_i ∈ Scope do
 5:       compute rank_u(v_i; t) using equation(10)
 6:       if v_i is ready
 7:         Readys ← v_i
 8:     end for
 9:     sort tasks in Scope by non-increasing order of rank_u
10:     for task v_i ∈ Scope do
11:       find bestVM_i that minimize EFT(v_i, bestVM_i; t)
12:       if v_i ∉ Maps
13:         Maps ← (v_i, bestVM_i)
14:       else if (v_i, bestVM_i) ∉ Maps do
15:         update bestVM for v_i in Maps
16:         update records whose destination is v_i in Datas
17:       end if
18:       if v_i ∈ Readys and bestVM_i is available
19:         schedule v_i to bestVM_i
20:     end for
21:     delete scheduled tasks from Maps, Readys, Scope
22:     for task v_f ∈ Fins do
23:       for task v_c ∈ child(v_f) do
24:       compute DLTT(v_c, bestVM_c, v_f, VM_f; t) using equation(7)
25:       Datas ← (DLTT, v_f, VM_f, v_c, bestVM_c, data)
26:       end for
27:       delete record whose destination is v_f from Datas
28:     end for
29:   end if
30:   empty Fins
31:   wait()
32:   send data at time DLTT in Datas
```

(Line 14-16). Then ready tasks whose best VMs are available are scheduled (Line 18-19) and records are deleted from *Maps*, *Readys*, and *Scope* (Line 21). For each finished task v_f, DLTT from v_f to each child is calculated, and $(DLTT, v_f, VM_f, v_{child}, bestVM_{child}, data)$ is added to *Datas* (Line 22-25). Record with v_f as the destination task is delete from *Datas* (Line 27). Finally, data is transferred according to *Datas* in the runtime (Line 32).

4.2 Detailed Description of the DEFT Algorithm

In this section, we describe each step of the DEFT algorithm in detail. Then we analyze the time complexity.

Suppose a workflow with n tasks is scheduled to m VMs in the IaaS cloud. In each scheduling loop, DEFT compute $rank_u$ for the tasks in *Scope* and select the best VMs for the tasks. Since $rank_u$ defined in Eq. (10) is recursively calculated by traversing the DAG from the exit task to the target task, all the unscheduled tasks are considered in the task prioritizing phase. The time complexity of computing $rank_u$ in a scheduling loop is $O(k^2m)$, for k unscheduled tasks and m VMs. The time complexity of VM selection phase and data transfer phase is $O(s * m)$, where s is the number of tasks in *Scope* and $s \leq k$. So, the total time complexity of each loop is $O(k^2m)$. In the worst case where there is only one task scheduled in each loop, there are n scheduling loops, and the time complexity of the l^{th} loop is $O((n-l)^2 * m)$. The total time complexity is $O(1^2 * m + 2^2 * m + \cdots + (n-1)^2 * m)$, that is $O(n^3 * m)$.

4.3 An Example

To better illustrate DEFT, we show an example comparing DEFT to PEFT [6], DCP-G [12], and DCP-G [20].

HEFT and PEFT are typical static algorithms using static information to generate scheduling map at the scheduling time. HEFT uses the same definition of $rank_u$ and selection strategy as ours in prioritizing phase and VM selection phase. PEFT proposes the concept of OCT (Optimistic Cost Table) in the task prioritizing phases.The OCT value of a task v_i on VM_j is recursively defined by Eq. (11) by traversing the DAG from the exit task to the entry task:

$$OCT(v_i, VM_j) = \max_{v_k \in child(v_i)} \left[\min_{VM_p \in VM} \{OCT(v_k, VM_p) + w_{kp} + c_{ik}\} \right] \quad (11)$$

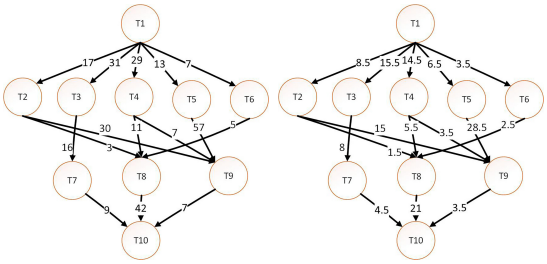
where w_{kp} is the computing cost of v_k running on VM_p, and c_{ik} is the communication cost from v_i to v_k. For the exit task, $OCT(v_{exit}, VM_p) = 0$ for all the $VM_p \in VM$. $c_{ik} = 0$ if $VM_j = VM_p$, or if VM_j and VM_p are hosted on the same server. In the VM selection phase, task v_i is assigned to VM_j minimizing $O_{ETF}(v_i, VM_j)$, defined by Eq. (12):

$$O_{EFT}(v_i, VM_j) = EFT(v_i, VM_j) + OCT(v_i, VM_j) \quad (12)$$

DCP-G selects and schedules tasks in the runtime. The task selected for scheduling is the one that is on the dynamic critical path, which has the same earliest start time and latest start time updated dynamically in the runtime, with no unmapped parent tasks and has the lowest earliest start time. The the VM with minimum execution time for that task is selected.

The scheduling results are shown in Fig. 1. There are 10 tasks of a DAG workflow scheduled on 3 fully connected heterogeneous VMs. Initially, network bandwidth between VMs are the same. Communication cost and the DAG is shown in Fig. 1 (a). Computing cost of each task on each VM is shown in the left four columns of Fig. 1 (c). At t=30s, computing rate of VM3 doubles, and the computing cost of each task on it is shown in the right column of Fig. 1 (c). Transfer rates between VM3 and the other two VMs also doubles, and the communication cost of each task on it is shown in Fig. 1 (b).

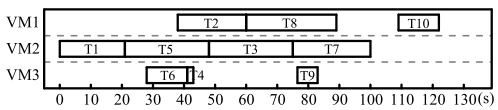
HEFT and PEFT use initial performance to schedule tasks. Tasks are sorted and queued on the same VMs after the performance variation. Final makespan of HEFT is 122s Fig. 1 (d), and PEFT is 122s Fig. 1 (e). DCP-G schedules each critical task dynamically in the runtime, and the final makespan is 116s Fig. 1 (f).



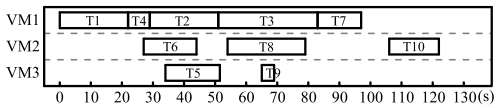
(a) Communication cost of DAG, t=0s. (b) Communication cost of DAG, t=30s.

	VM1	VM2	VM3(t=0)	VM3(t=30)
T1	22	21	36	18
T2	22	18	18	9
T3	32	27	43	21.5
T4	7	10	4	2
T5	29	27	35	17.5
T6	26	17	24	12
T7	14	25	30	15
T8	29	23	36	18
T9	15	21	8	4
T10	13	16	33	16.5

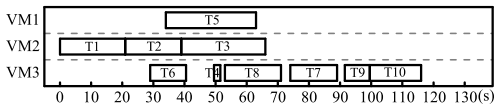
(c) Computing cost matrix.



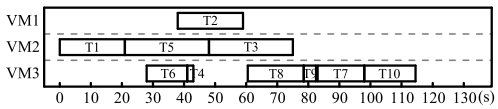
(d) HEFT, makespan=122s.



(e) PEFT, makespan=122s.



(f) DCP-G, makespan=116s.



(g) DEFT, makespan=114.5s.

Fig. 1 Scheduling examples of HEFT, PEFT, DCP-G, and DEFT.

DEFT contains scheduling loops invoked dynamically in the runtime. When T6 finishes at $t = 41s$, the scheduling loop starts with the remaining task sorted as {T4, T3, T8, T7, T9, T10}. *Scope* is {T4, T3}, *Ready* is {T4, T3}. Best VM for T4 is VM3, with EST=41s. Then VM3's available time for T3 is at t=43s (computing cost of T4 on VM3 is 2s), computing cost is 21.5s. In the scheduling loop when T1 finishes at $t = 21s$, best VM for T3 is VM2. There is no data transfer since T3 is recorded to be assigned to the same VM as T1. Therefore, if T3 is scheduled to VM3 in the scheduling loop at t=41s, data is sent from VM2 at $t = 41s$ and received by VM3 at $t = 56.5s$. In this case, $EST(T3, VM3; 41s) = \max\{56.5, 43\}+21.5 = 56.5+21.5 = 78s$. At $t = 41s$, VM2's available time for T3 is at t=48s, computing cost is 27s, and communication cost is 0s since T3 is recorded to be assigned to the same VM as T1. In this case, $EFT(T3, VM2; 41s) = \max\{41, 48\} + 27 = 48 + 27 = 75s$. Therefore, the best VM for T3 is VM2. The final scheduling of DEFT is shown in Fig. 1 (g), and the makespan is 114.5s.

5. Experimental Results

In this section, we present the comparative evaluations of DEFT and PEFT, DCP-G in the IaaS cloud. We first describe experimental settings. Then we show the comparative evaluations of the three algorithms.

5.1 Experimental Settings

We use the DynamicCloudSim [25] toolkit to simulate the IaaS cloud. DynamicCloudSim extends the CloudSim [26] simulation toolkit by introducing models for: (1) inhomogeneity in the performance of computational resources, (2) uncertainties and dynamic changes to the performance of VMs, and (3) straggler machines and failures during the task execution.

We use the default setting of the data center in DynamicCloudSim. The data center contains 100 servers of Intel Xeon E5430 2.66 GHz, 200 servers of AMD Opteron 270 2 GHz, and 200servers of AMD Opteron 2218 HE 2.6 GHz. 100 VMs are randomly hosted on the servers and allocated to random performance parameters. The default I/O throughput of VMs is 20 MB/s. The servers are fully connected. The external bandwidth of VMs is 0.25 MB/s, and the latency of data transfer is not considered.

In DynamicCloudSim, dynamic change of CPU rate of VMs, I/O throughput, are network bandwidth are simulated in the same way. The time of the next performance change is sampled from an exponential distribution with a given rate parameter, and the new value of the given characteristic is sampled from a normal distribution with the mean set to the default value of the given characteristic. Higher values in both the rate parameter of the exponential distribution and RSD (relative standard deviation) of the normal distribution correspond to higher levels of dynamics. We use the tuple $\langle change\_number, dynamic\_degree \rangle$ to donate the performance changes. Since the time of the next performance change is randomly sampled and the makespans of a workflow vary using different scheduling methods, we use the ranges of the number of changes instead of the fixed values, {1-10, 10-20, 20-30, 30-40}. The low dynamic degree is the default setting of RSD, which is 0.054 for CPU, 0.033 for I/O, and 0.04 for network bandwidth, and the high dynamic degree is RSD=0.5 for CPU, I/O, and network bandwidth. The values used in the experiments are marked: $\{\langle 10, l\rangle, \langle 10, h\rangle, \langle 20, l\rangle, \langle 20, h\rangle, \langle 30, l\rangle, \langle 30, h\rangle, \langle 40, l\rangle, \langle 40, h\rangle\}$ for short.

We use the typical scientific workflow Montage, which has been repeatedly utilized for evaluating scheduling mechanisms and computational infrastructures for scientific workflow execution in the past. The DAG structure of each workflow, including task dependencies, computational characteristics, the input and output files of the tasks, etc. is de-

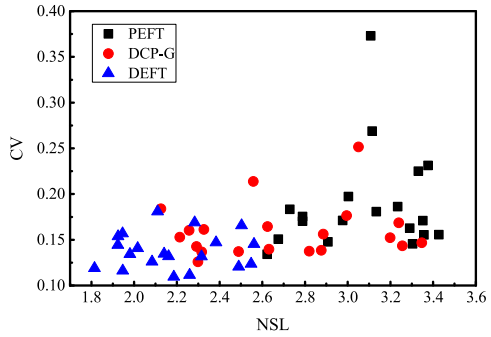


Fig. 2 Average NSL and CV in the default setting.

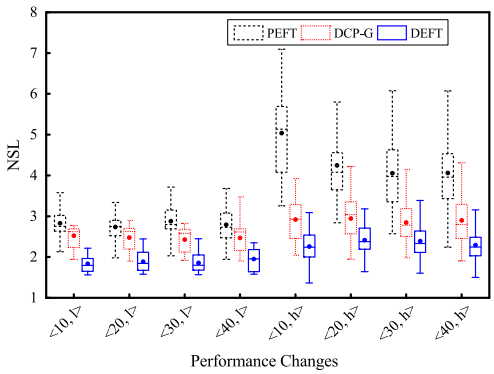


Fig. 3 NSL with respect to different settings of performance changes.

scribed in Pegasus DAX (Directed Acyclic Graph in XML) format. Each size of the workflow, from 50 to 1000, have 20 instances generated by WorkflowGenerator [27]. Each workflow is run 100 times in the experiment, and the mean makespan and CV of the makespans are computed.

5.2 Evaluation Results

In this section, we compare the makespan and robustness performances of PEFT, DCP-G and DEFT in the IaaS cloud.

Figure 2 shows the NSLs and robustness of the 20 workflows consisting 1000 task nodes using PEFT, DCP-G and DEFT in the default setting of performance changes. All the workflows using DEFT have smaller makespans than using PEFT, and 19 workflows using DEFT have smaller makespans than using DCP-G. DEFT can produce more robust schedules than PEFT for 19 workflows, and DCP-G for 18 workflows.

Figure 3 and Fig. 4 shows makespans and robustness of the three algorithms with respect to different settings of performance changes. The workflow with the minimum CV differences of the three algorithms in Fig. 2 is chosen. NSLs of the workflow are 2.56 of PEFT, 2.47 of DCP-G and 1.86 of DEFT in the static environment where CPU, I/O, and network bandwidth do not vary during execution of the workflow.

Figure 3 presents the boxplots showing the minimum, 25 percent, mean, 75 percent, and maximum values of the NSL of the three scheduling methods with respect to different settings of performance changes. DEFT can always have the smallest NSLs. The average NSLs of PEFT, DCP-G, and DEFT at the low dynamic degree are 2.83, 2.52, and 1.82, respectively, for performance of CPU, I/O, and network bandwidth changes less than 10 times during the execution of the workflow, and are 2.79, 2.47, and 1.95, respectively, for performance changes over 30 times. The improvement of DEFT over PEFT and DCP-G are 35.69% and 27.78% for performance changes less than 10 times, and 30.11% and 21.05% for performance changes more than 30 times. The average NSLs of PEFT, DCP-G, and DEFT at the high dynamic degree are 5.04, 3.91, and 2.92, respectively, for performance changes less than 10 times, and 4.06, 3.65, and 2.90, respectively, for performance changes over 30 times. The improvement of DEFT over PEFT and DCP-G are 42.06% and 32.89% for performance changes less than 10 times, and 28.57% and 20.55%for performance changes over 30 times.

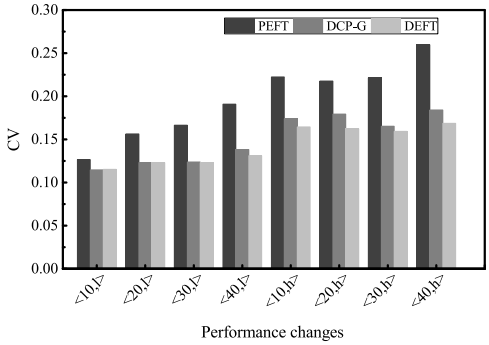


Fig. 4 CV with respect to different settings of performance changes.

Figure 4 shows CVs of PEFT, DCP-G, and DEFT with respect to different settings of performance changes. CVs of DCP-G and DEFT are similar as they schedule tasks dynamically in the runtime. The average CVs of the low dynamic degree are 0.16 of PEFT, 0.124 of DCP-G, and 0.123 of DEFT. The average CVs of the high dynamic degree are 0.23 of PEFT, 0.176 of DCP-G, and 0.164 of DEFT. The improvement of DEFT and DCP-G over PEFT are 9.36% and 8.86% for performance changes less than 10 times at the low dynamic degree, and are 29.05% and 35.03% for performance changes more than 30 times at high dynamic degree.

Scheduling time is considered as the scheduling overhead, which constitutes a significant amount of time used by the scheduler to generate the schedule. Figure 5 shows the total scheduling time of workflows with different sizes of the three scheduling methods running in the default setting of the dynamic IaaS cloud data center. It shows that the scheduling time overhead of the three methods are similar if the workflow size is small (consisting less than 300 tasks). With the workflow size increases, the overhead of DCP-G and DEFT increases linearly, while PEFT does not vary too much. However, it should be noticed that the average makespan of the workflows with 1000 tasks is around 7000 seconds, and the total scheduling time of the workflow

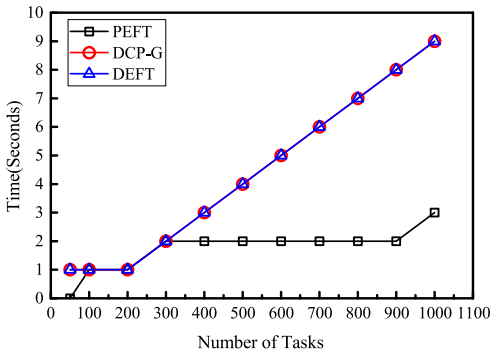


Fig. 5 Scheduling time of different workflow sizes.

is around 9 seconds. The scheduling time overhead is acceptable compared to the makespan of the workflows in our experiment. Moreover, DEFT provides a good choice for workflows with the execution time of tasks large enough to tolerant the overhead, which is about 10ms.

6. Conclusion and Future Work

Scientific workflows are increasingly adopting the IaaS cloud where computing resources are provided in the form of VMs, dynamically managed, monitored, maintained, and governed by market principles. In this paper, we deal with the challenges of scheduling workflows to the IaaS cloud with uncertainties like resource performance variations and unknown failures. A dynamic scheduling algorithm, called DEFT, is proposed to improve both makespan and robustness. DEFT contains a set of list scheduling loops invoked when some tasks complete and release resources. In each loop, runtime informations are used to sort tasks, select VMs, and send intermediate data. Experiments show that DEFT can produce shorter makespan than existing typical list-based scheduling, PEFT, and dynamic scheduling, DCP-G. It also produces more robust schedules in the IaaS cloud with uncertainties.

Existing algorithm uses the dynamic scheduling to alleviate impact of uncertainties like the resource performance variations and unknown failures. In the future work, we will research some monitor functions to discover potential failures and a predict model to better understand the performance variations.

Acknowledgments

This work is supported by the National Key project of Scientific and Technical Supporting Programs of China (Grant No.2014BAK15B01); the Cosponsored Project of Beijing Committee of Education; Engineering Research Center of Information Networks, Ministry of Education.

References

[1] F. Fakhfakh, H.H. Kacem, and A.H. Kacem, "Workflow scheduling in coud computing: a survey," IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, pp.372–378, 2014.

[2] E.N. Alkhanaka, S.P. Leea, R. Rezaeia, and R.M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and Grid computing: a review, classifications, and open issues," The Journal of Systems and Software vol.113, pp.1–26, 2016

[3] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," Journal of Grid Computing, vol.13, no.4, pp.457–493, 2015.

[4] D.I.G. Amalarethinam and A.M. Josphin, "Dynamic task scheduling methods in heterogeneous systems: a survey," International Journal of Computer Applications, vol.110, no.6, pp.12–18, 2015.

[5] G.P.J. Dejun, and C.H. Chi, EC_2: performance analysis for resource provisioning of service-oriented applications, Workshop on Service Oriented Computing (ICSOC/ServiceWave), pp.197–207, 2009.

[6] 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.

[7] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," Journal of Computer Sciences, vol.3, no.2, pp.94–103, 2007.

[8] T. Hagras and J. Janecet, "A simple scheduling heuristic for heterogeneous computing environments," Second International Symposium on Parallel and Distributed Computing, pp.104–110, 2003.

[9] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, "High performance task scheduling algorithm for heterogeneous computing system," Distributed and Parallel Computing, vol.3719, pp.193–203, 2005.

[10] L.F. Bittencourt, R. Sakellariou, and E.R.M. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pp.27–34, 2010.

[11] E.U. Munir, S. Mohsin, A. Hussain, M.W. Nisar, and S. Ali, "SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems," IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, pp.43–53, 2013.

[12] H. Arabnejad and J.G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," IEEE Transactions on Parallel and Distributed Systems, vol.25, no.3, pp.682–694, 2014.

[13] M.A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," IEEE Transactions on Cloud Computing, vol.2, no.2, pp.222–235, 2014.

[14] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A stochastic scheduling algorithm for precedence constrained tasks on Grid," Future Generation of Computing System, vol.27, no.8, pp.1083–1091, 2011.

[15] D. Poola, S.K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), pp.858–865, 2014.

[16] W. Zheng and R. Sakellariou, "A Monte-Carlo approach for full-ahead stochastic DAG scheduling," Proceedings of the 21st Heterogeneous Computing Workshop, pp.99–112, 2012.

[17] W. Zheng and R. Sakellariou, "Stochastic DAG scheduling using a Monte Carlo approach," Journal of Parallel and Distributed Computing, vol.73, no.12, pp.1673–1689, 2013.

[18] Q. Zheng, "Dynamic adaptation of DAGs with uncertain execution times in heterogeneous computing systems," IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp.1–8, 2010.

[19] J.G. Barbosa and B. Moreira, "Dynamic scheduling of a batch of

parallel task jobs on heterogeneous clusters," Parallel Computing, vol.37, no.8, pp.428–438, 2011.
[20] M. Rahman, R. Hassan, R. Ranjan, and R. Buyya, "Adaptive workflow scheduling for dynamic Grid and cloud computing environment," Concurrency and Computation: Practice and Experience, vol.25, no.13, pp.1816–1842, 2013.
[21] L. Bölöni and D.C. Marinescu, "Robust scheduling of metaprograms," Journal of Scheduling, vol.5, no.5, pp.395–412, 2002.
[22] Z. Shi, E. Jeannot, and J. Dongarra, "Robust task scheduling in non-deterministic heterogeneous computing systems," Cluster Computing, pp.1–10, 2006.
[23] L.-C. Canon and E. Jeannot, "Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments," IEEE Transactions on Parallel and Distributed Systems, vol.21, no.4, pp.532–546, 2010.
[24] V. Shestak, J. Smith, H.J. Siegel, and A.A. Maciejewski, "A stochastic approach to measuring the robustness of resource allocations in distributed systems," International Conference of Parallel Processing (ICPP), pp.459–470, 2006.
[25] M. Bux and U. Leser, "DynamicCloudSim: simulating heterogeneity in computational clouds," Future Generation Computer Systems, vol.46, pp.85–99, 2015.
[26] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," SoftwarePractice and Experience vol.41, no.1, pp.23–50, 2011.
[27] Pegasus, WorkflowGenerator, https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, Accessed on 27th Oct. 2016.

Haiou Jiang is currently pursuing Ph.D. in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a B.E. and a M.E. degree from College of Computer Science at Beijing University of Posts and Telecommunications. She is interested in cloud computing technology, cloud data center management, service computing.

Haihong E is currently an associate professor in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a B.E. and a M.E. degree from College of Electronic Engineering, a Ph.D. from College of Computer Science at Beijing University of Posts and Telecommunications. She is now working in PCN&CAD center of Beijing University of Posts and Telecommunications. Her research interests are distributed system, SSME (service science, management and engineering), internet of things, wireless city applications, and mobile internet applications.

Meina Song is currently a professor in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a M.E. and a Ph.D. degree from College of Electronic Engineering at Beijing University of Posts and Telecommunications. She is now the chairman of PCN&CAD center of Beijing University of Posts and Telecommunications, vice president of mobile internet application and terminal technology work committee of China communications standards association, member of technical committee of service computing in China computer federation, member of technical committee of network and data communications in China computer federation. Her main research areas include distributed system, P2P technology, service computing, big data technology etc.