Cloud YARN resource provisioning for task-batch based workflows with deadlines

Xiaoping Li, *Senior Member, IEEE,* Zhicheng Cai,*Student Member, IEEE,* Rubén Ruiz

Abstract—To meet the dynamic workload requirements in widespread task-batch based workflow applications, we developed the cloud YARN (C-YARN) architecture by integrating the YARN platform with cloud computing. The C-YARN could provision flexible resources. In terms of depths and functions, tasks of different task-batches were merged to task-units. Based on task-units, a unit-aware deadline division method was investigated for properly dividing workflow deadlines to task deadlines to minimize the utilization of rented intervals. Considering different factors affecting time slot allocation, several rules were introduced for scheduling tasks with the task deadlines. A rule-based task scheduling method was presented for allocating tasks to time slots of rented Virtual Machines (VMs) with a task right shifting operation and a weighted priority composite rule. A Unit-aware Rule-based Heuristic (URH) was proposed for elastically provisioning VMs to task-batch based workflows to minimize the rental cost in C-YARN. Effectiveness of the proposed URH methods was verified by comparing them against two adapted existing algorithms for similar problems on some realistic workflows.

Index Terms—cloud computing, workflow scheduling, resource provisioning, task-batch, interval based pricing model.

1 Introduction

TASK-BATCH based workflows are more general than traditional ones [1], [2]. They are widespread in many fields, especially in data analysis applications, such as mobile applications, e-commerce and scientific research (which often process large volumes of data through a series of connected operations [3]). In order to speed up the process for each operator, data is partitioned and processed by multiple parallel tasks which form a task-batch. For an application, the flow of different operations forms a workflow which is composed of many dependent task-batches (it becomes a traditional workflow if there is only one task in each task-batch). Workflows with such specified network structures are called task-batch based workflows. A typical example is the Fake-License Plate Detecting application (FLPD). FLPD is applied to detect cars with fake-license plates by analysing data obtained from road cameras. The FLPD consists of six dependent operations (Data partition, Record generation, Reducing by locations, Abnormity detection, Result combination and Final combination) as shown in Figure 1. Multiple parallel tasks of each operation form a task-batch. For example, v_2, v_3, v_4 and v_5 generate transportation records from different data partitions obtained from the same district and they form a task-batch.

- *Xiaoping Li and Zhicheng Cai work at the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China, and also with the Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, 211189, Nanjing, China. E-mail:{xpli, caizhicheng}@seu.edu.cn.*

- *Rubén Ruiz works at the Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain (e-mail: rruiz@eio.upv.es).*

Hadoop 1.0 based systems [4], [5] are not ideal platforms for task-batch based workflow applications. A MapReduce job [6] is usually adopted to process data partitions of one or more task-batches and the whole workflow might consist of many different MapReduce jobs. However, an on-demand cluster (Hadoop 1.0) [4] for MapReduce jobs cannot be resized until the tasks on it are all completed, incurring many idle nodes while processing slimmer stages (light workload) [4]. As stated in [4]: only one reduce task being executed in a node is enough to prevent claiming back the cluster while this is at the same time common, i.e., resources of different MapReduce clusters are hard to be shared to improve the utilization of resources. At the same time, MapReduce is not a suitable programming framework to describe workflow-based logics.

Recently, several more flexible cluster management systems (such as YARN [4] and Mesos [7]) have been developed. YARN gives the possibility of choosing more appropriate programming frameworks such as *Dryad* [8] and designing efficient task scheduling strategies according to application characteristics. *Dryad* allows users to specify arbitrary Directed Acyclic Graphs (DAGs) to describe the application's logic such as data communication patterns among different task-batches. Unlike adopting multiple separate on-demand Hadoop 1.0 clusters, resources of YARN with *Dryad* can be shared among task-batches of the same application to improve the utilization of rented Virtual Machines (VMs) [9]. Existing studies such as Tabaa et al. [5] usually adopted local clusters or private clouds as the resource pool for YARN, which had fixed capacities. However, task-batch based workflow applications always take hours or days to process large volumes of data. In different stages of task-batch based workflows, workloads change greatly. Nowadays many companies and institutes are

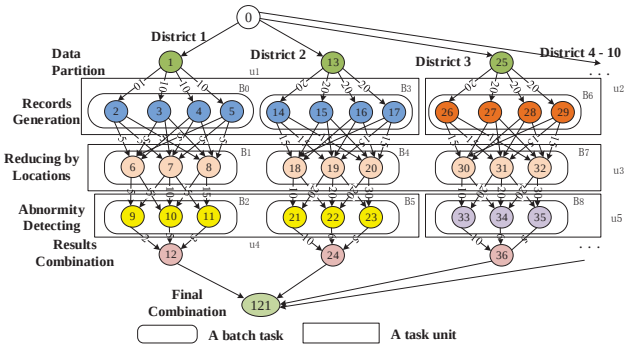

Fig. 1. An example of the task-batch based workflow

trying to migrate their applications to commercial public clouds in order to achieve adaptive capacities. Cardosa et al. [10] and Chen et al. [11] investigated scheduling of dynamically built MapReduce clusters based on virtual machines (VMs) to increase energy efficiency and to minimize cost. However, the MapReduce framework is not suitable for dynamic workload situations. Therefore, we proposed the cloud YARN (C-YARN) with DAG based programming framework for task-batch based workflows which was an extension of YARN by adding support of renting resources from clouds to YARN. Cloud resources were elastically provisioned in terms of dynamic workloads.

Developing smart algorithms to help application users request (or release) resources from (or to) public clouds and minimizing the resource rental cost are crucial to optimally provision cloud resources to applications in the considered C-YARN system. For such applications, tasks in the same batch usually have the same functionality (or operation) and the same software requirements. Tasks of different batches with the same functionality are merged to a single task-unit. Tasks of a task-unit can be executed in parallel to accelerate the execution process. They can also be fulfilled sequentially to improve the utilization of rented time intervals when the deadline is not very tight. In other words, tasks are executed by task-groups according to the number of rented virtual machines. Tasks of each task-group are sequentially executed on the same virtual machine. The size of a task-group is called clustering granularity as in [12]. Because the number of rented machines cannot be determined in advance (i.e., the involved data center has an elastic capacity) in C-YARN, the clustering granularity is scalable. Actually, the more sequenced tasks there are, the larger possibility of maximizing the utilization of the rented intervals (minimize the final rental cost). In other words, the final rental cost is closely related to clustering granularity. However, the scalable clustering granularity is constrained by task deadlines while task deadlines are determined by the workflow deadline. Therefore, it is essential to properly divide the workflow deadline to task deadlines to obtain proper clustering granularities, which improve the utilization of rented time intervals and minimize the rental cost in the elastic C-YARN. In existing methods for traditional workflows such as those proposed by Byun et al. [9], Mao et al. [13], Abrishami et al. [1] and Durillo et al. [2], task deadlines were generated for tasks separately without considering task-batches. If these methods were applied to task-batch based workflows, imbalance tasks deadlines would be resulted. For example, tight task deadlines miay be allocated to task-batches consisting of many short tasks or extremely loose task deadlines may be assigned to task-batches with fewer long tasks, which all lead to a lower utilization of rented intervals.

In this paper, a Unit-aware Rule-based Heuristic (URH) was proposed which distributed the workflow deadline to competitive task-units. The URH guided C-YARN users to rent the appropriate type and number of resources from public clouds and to schedule tasks to appropriate time slots on the rented virtual machines minimizing the VM rental cost. Key contributions we made in this paper were summarized as follows:

(i) Integrating cloud computing and the YARN platform, the C-YARN platform was presented for task-batch based workflows. C-YARN provisioned resources elastically .

(ii) A unit-aware deadline division method was developed for properly dividing the workflow deadline into task deadlines. Scalable clustering granularities of task-units were obtained which improved the utilization of rented time intervals.

(iii) A rule-based task scheduling strategy was proposed for the deadline constrained tasks. A unit-aware rule-based heuristic was investigated for the C-YARN resource provisioning problem in task-batch based workfolws.

The rest of the paper was organized as follows. Section 2 gave the related work and the problem was described in detail in Section 3. Section 4 presented the proposed heuristic which was evaluated in Section 5. Section 6 concluded the paper.

2 Related Work

Performance optimization of *MapReduce* and DAG based scheduling [14] on dedicated clusters with fixed capacities has been studied extensively in the literature. For example, algorithms for *MapReduce* mainly focused on improving system performances, such as makespan minimization [15] and energy saving [16]. For DAG based tasks, many researchers considered the improvement of system performances of distributed systems with fixed capacities, such as [17]–[21]. In traditional service-oriented computing, resources could be dynamically provisioned as services to workflow applications [22]–[24]. They used economic models which were different from those in public clouds. For example, cloud resources were usually priced by intervals and were shareable among tasks of the same workflow while services in service-oriented computing were priced per

task. Therefore, algorithms for private cluster or service-oriented computing are not suitable for task-batch based workflow scheduling in public clouds.

Since most companies and research institutes lack resources to operate large private clouds, studies on provisioning resources for *MapReduce* and DAG-based applications in public clouds are of great interest. For example, Chen et al. [11] proposed a method to help users make decisions about resource provisioning for running MapReduce programs in public clouds. Performance of MapReduce (Hadoop) on a 100-node cluster of Amazon EC2 with various levels of parallelism was studied by Jiang et al [25]. The Balanced Time Scheduling (BTS) algorithm, proposed by Byun et al. [26], minimized the client-oriented resource renting cost of DAG-based applications. BTS was based on the assumption that there was a fixed number of homogeneous resources during the whole workflow execution horizon. In a later work by Byun et al. [9], the Partitioned Balanced Time Scheduling (PBTS) was developed for elastic resource provision patterns, which found the minimum number of resources for each time partition rather than the whole workflow execution horizon. However, it still assumed that homogeneous resources were used, i.e., only one kind of VM existed in the system. Recently, Mao et al. [13] proposed an approach to support the running of workflow applications on auto-scale cloud VMs, which took advantage of deadline division and task consolidation. Abrishami et al. [1] and Juan et al. [2] considered the resource provisioning of scientific workflows with heterogeneous resources in IaaS (Infrastructure as a Service) clouds. In order to mitigate effects of performance variation of cloud resources, Calheiros et al. [27] proposed an algorithm that used the idle time of provisioned resources and budget surplus to replicate tasks. They considered the data transfer among tasks and choices among multiple types of VMs. However, the operating system loading time and the software setup time were not considered. In our previous work [28], a Multiple-Rules based Heuristic (MRH) was developed to provision resources to workflow applications, in which the data locality and software locality were all considered. These works were all designed for traditional workflows without considering task-batches.

To the best of our knowledge, there is only one existing work [6] on resource provisioning for task-batch based workflows. There each task-batch was considered as a whole and executed by a cluster (traditional Hadoop 1.0) with a specific number of VMs determined by its modes. Resources of each task-batch could not be released until the whole task-batch is finished (this is common in traditional Hadoop 1.0 clusters), which led to a low resource utilization.

3 Problem Formulation

3.1 The C-YARN system

In this paper, we introduced the C-YARN system for task-batch based workflow applications by deploying the existing YARN to public clouds, which included two roles: the cloud provider and the cloud user. The IaaS cloud provider supplied Virtual Machines (VMs) to cloud users. Cloud users rented VMs from cloud providers to establish their own C-YARN systems to serve their applications. Distinct from the traditional YARN system built on local clusters described in [4], there was an additional component the Elasticity Controller (EC) in the Resource Manager (RM) of the C-YARN as shown in Figure 2.

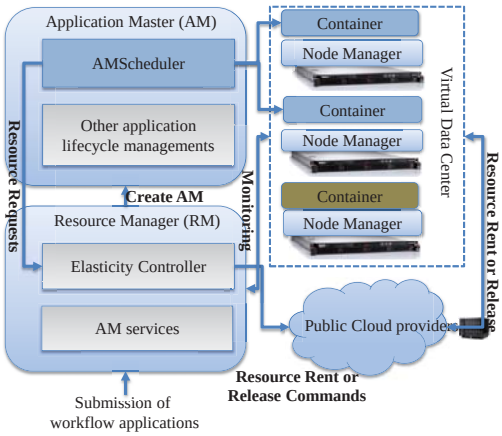

Fig. 2. Architecture of the developed C-YARN

The EC of the RM was in charge of fulfilling the request of workflow applications in the system by dynamically renting and releasing resources from and to public clouds. Whenever a task-batch based workflow was submitted to the C-YARN system, the RM created an Application Master (AM) for the application which was in charge of managing all life-cycle aspects including dynamic resource scaling, data flow management and fault handling, etc. The most important component in the AM was the AMScheduler in charge of making dynamic resource consumption plans (issued Resource Requests to the RM) and scheduling tasks to the received containers (built on virtual machines in this paper) according to specific applications. In this paper, heuristics were designed for the AMScheduler to provision VMs to the task-batch based workflows. The objective was to minimize the total VM rental cost while meeting the workflow deadline D.

3.2 Task-batch based workflow applications

Different types of Virtual Machines (with distinct prices) are provided by many commercial clouds, in which interval-based (monthly, hourly or minute) pricing models are offered. The whole interval is paid even if only part of the interval is used. For example, Table 1 shows details of different VM types at Amazon EC2. Let P_δ be

the price of VM type δ per interval unit. $\mathbf{L}$ is the pricing interval length. A task-batch based workflow is defined by a Directed Acyclic Graph (DAG) $G = \{V, E\}$ and a set $\mathcal{B}$. $V = \{v_0, v_1, \ldots, v_{N+1}\}$ is the set of all tasks where the source node v_0 and the sink node v_{N+1} are dummy nodes. $E = \{(i,j)|i < j\}$ is the precedence constraints of tasks (such as data transfer dependencies), the arc (i,j) indicates that v_j cannot start until v_i completes. $\mathcal{P}_i$ represents the immediate predecessor set of v_i. $\mathcal{B} = \{B_0, B_1, \ldots, B_Q\}$ is the set of task-batches, in which B_i is the i^{th} task-batch. The depth of v_i is defined as the minimum number of nodes (tasks) along the path from v_0 to v_i, e.g., the depth of v_2 of the workflow shown in Figure 1 is 3. Tasks of the same task-batch have the same depth. For the workflow in Figure 1, $\mathcal{B} = \{B_0, B_1, B_2, B_3, B_4, B_5, ...\}$ in which batches $B_0 = \{v_2, v_3, v_4, v_5\}$, $B_1 = \{v_6, v_7, v_8\}$, $B_2 = \{v_9, v_{10}, v_{11}\}$, etc.

Different types of VMs have different configurations, which are suitable for different types of tasks, e.g., high-CPU VM instances for computational-intensive tasks. We let $T^e_{i,\delta}$ denote the execution time of task v_i on VM instances of type δ. The fact that execution times of tasks are different on various VM instances makes allocation very difficult. Several existing methods [29], [30] can be used to estimate execution times. Though sometimes cloud providers limit each user the amount of rented resources, we can still obtain sufficient amount of resources from multiple clouds. Therefore, it is reasonable to assume that there is no limitation on the amount of VM instances. Besides execution times of tasks, data transfer times are non-negligible. In fact, data transferring among tasks is always time-consuming and is dependent on the volume of data and the system network bandwidth w^B. Because VM instances are usually rented from the same data center of a cloud provider, we assume that bandwidths of different VM instances are the same. $\mathcal{Z}_{i,j}$ denotes the volume of intermediate data transferred from v_i to v_j. Different system software (such as the operating system, middle-ware and professional software components) are required to execute tasks, which need times to setup (including VM image transfer times, OS loading times, professional software downloading and installing times). In this paper, we assume that VM setup times for the same type of VM instances are equal (which can be estimated according to experiences). T^m_δ denotes the VM setup time for the VM type δ, ϖ_i being the software component for v_i with the setup time T_{ϖ_i}.

3.3 Challenges for the problem under study

In this paper, we considered the C-YARN resource provisioning problem for task-batch based workflows. Though resource provisioning for task-batch based workflows was studied in our previous work [6] using Hadoop 1.0 clusters, there are many new challenges for this problem using the proposed C-YARN platform. The main challenges are:

(i) In C-YARN, cloud resources are dynamically provisioned to adapt dynamic workload demands in different stages of task-batch based workflows. Adaptive workflow deadline division is required to properly divide workflow deadlines to task deadlines.

(ii) Execution modes of task-batches in this paper are adaptively determined involving in many factors (such as the number of tasks in each task-unit, task processing times, the pricing interval length and the deadline division method). In [6], execution modes were predefined.

(iii) Task-batches in C-YARN are decomposable and they are regrouped into task-groups for execution with scalable clustering granularity. The clustering granularity closely depends on the number of rented Virtual Machines which is the optimization objective. Task-batches in [6] are undecomposable and each task-batch was scheduled as a whole. Therefore, workflow deadline division for task-level scheduling in this paper is much more complicated than that for batch-level scheduling in [6].

4 Proposed Heuristic

To minimize the total VM rental cost, all tasks of a workflow application are scheduled with the workflow deadline D and tasks' precedence constraints. Similar to existing workflow scheduling, there are two phases in the resource provisioning problem under study in this paper: deadline division and task scheduling. Workflow deadlines are divided into task deadlines with which tasks are scheduled. There are many factors exert influences on the two phases, such as the complexity of workflows (precedence constraints between tasks), task types (task-batches or tasks), and resource pricing models (priced by per usage or time intervals). Because of the above challenges, existing workflow scheduling algorithms [6], [22], [28], [31] are not suitable for resource provisioning for task-batch based workflows in C-YARN.

In this paper, a Unit-aware Rule-based Heuristic (URH) was developed for the problem under study. We called the amount of time distributed to a task as the task float duration (TFD), that distributed to or a task-unit as the task-unit float duration (UFD). Estimated wasted cost (EWC) was defined as the total of the wasted rental cost of unused fractions on the rented intervals and the software setup cost of a task-unit with a given execution mode. The URH mainly consisted of the following two steps:

(i) The workflow deadline D was divided into task deadlines in terms of task-units. Task-batches with the same depth and function were merged to a task-unit. Then, UFDs were initialized by the cheapest then slowest VM types. If the length of the critical path generated by the UFDs was longer than D, some tasks were reassigned to faster VM types. Next, D was distributed to competitive task-units in such a way so that the total of their EWCs was decreased.

TABLE 1
Configurations and Prices (per hour) for VMs at Amazon EC2

	VM Type	Configuration	Price
Normal type	Small (N_S)	1.7 GB MEM, 1 EC2 CU	$0.06
	Medium (N_M)	3.75 GB MEM, 2 EC2 CU	$0.12
	Large (N_L)	7.5 GB MEM, 4 EC2 CU	$0.24
	Extra Large(N_EL)	15 GB MEM, 8 EC2 CU	$0.48
High-memory type	Extra Large (M_EL)	17.1 GB MEM 6.5 EC2 CU	$0.41
	Double Extra Large (M_DEL)	34.2 GB MEM 13 EC2 CU	$0.82
	Quadruple Extra Large (M_QEL)	68.4 GB MEM 26 EC2 CU	$1.64
High-CPU type	Medium (C_M)	1.7 GB MEM 5 EC2 CU	$0.145
	Extra Large (C_EL)	7 GB MEM 20 EC2 CU	$0.58

(ii) Tasks were scheduled to time slots of rented VMs with task deadline constraints. Tasks were selected based on their priorities while meeting the topological requirements. Time slots for the tasks were determined by an introduced right shifting operation and a weighted composite rule.

4.1 Task-unit generation

Utilization of rented resource intervals could be improved by combining or consolidating tasks (either inter- or intra-batches) with the same depth and functionality. All tasks in a task-batch based workflows were merged to task-units according to their depths and functionalities while their logical precedence constraints among them were still kept, i.e., there were precedence constraints between task-units. Basically, tasks of the same task-unit were allocated to the same deadline. For example, tasks of B_0 and B_3 in Figure 1 had the same depth and the same functionality. They were merged to task-unit u_1. Though the task depth of B_6 was identical to that of B_0, different kinds of software were needed. Therefore, B_0 and B_6 were not consolidated to a task-unit. Other consolidated task-units were shown in Figure 1 labeled by rectangles. Tasks of each task-unit were assigned the same deadline. These dependent task-units competed for the time interval $[0, D]$.

A larger UFD usually meant a higher possibility of serializing more tasks, choosing more suitable types of VMs and consolidating tasks more freely. However, a larger UFD of task-unit u also usually led to smaller UFD of its predecessor or successor task-units. In other words, there were many strategies for allocating the time interval $[0, D]$ to task-units. Different strategies had various impacts on the final resource rental cost, which gave different opportunities to task serialization and consolidation of task-units. Therefore, it was crucial to determine how much time (UFD) was allocated to each task-unit under the D constraint.

4.2 Unit-aware deadline division

Based on the generated task-units, the workflow deadline D was distributed to UFDs. TFDs were obtained in terms of UFDs. The earliest finish times of tasks were calculated by TFDs and they were regarded as the final task deadlines. In this paper, a unit-based deadline division method was proposed which determined execution modes of task-units. Task-batch attributes (the number of tasks, task execution times and task execution requirements), the length of the adopted pricing interval and deadline competition between task-batches were taken into account in order to distribute the workflow deadline properly.

4.2.1 Estimated wasted cost

Generally, distributing the same UFD to different task-units exerts a great influence on the final rental cost. We adopted the estimated wasted cost (EWC) to measure of the benefit of assigning a UFD to a task-unit. For a task-unit u, a given VM type δ and a given UFD σ was called an execution mode $\Upsilon_{\delta,\sigma}$ of u. For each $\Upsilon_{\delta,\sigma}$, the minimum number of needed VMs and rented intervals for u were estimated based on the assumption that all VMs were newly rented and only tasks of u were scheduled to them. During the estimation procedure, task processing times of the task-unit were supposed to be identical, which were equal to the largest task processing time of the task-unit. For each VM instance, only the software setup time of the first task was taken into account. Figure 3 showed an example of the estimation of the minimum number of VM instances and rented intervals in which there were five tasks with the largest processing time of 20s on M_EL VMs and a given UFD of 80. Three M_EL VMs were needed and one time interval was newly rented on each VM instance when the software setup time was 10s. We defined that the EWC of u with the execution mode $\Upsilon_{\delta,\sigma}$ was the total of the wasted rental cost of unused fractions on the rented intervals and the software setup cost, which was computed as follows.

$$\mathcal{W}_{u,\delta,\sigma}=\frac{\{g_\sigma[M^m_{u,\delta,\sigma}(I^m_{u,\delta,\sigma}\mathbf{L}-T_{\varpi_i})-T^e_{u,\delta}]+T_{\varpi_i}M^m_{u,\delta,\sigma}\}P_\delta}{\mathbf{L}} \quad (1)$$

where $T^e_{u,\delta}$ was the total processing time of unit u, $M^m_{u,\delta,\sigma}$ and $I^m_{u,\delta,\sigma}$ were the minimum number of VM instances and rented intervals for $\Upsilon_{\delta,\sigma}$ respectively. g_σ was a binary variable. If $\sigma \leq \mathbf{L}$, $g_\sigma=1$. Otherwise, $g_\sigma=0$, i.e., when $\sigma>\mathbf{L}$, only the software setup cost was considered as the EWC. For the example in Figure 3, $\mathcal{W}_{u,M\_EL,80}=\{1\times[3\times(1\times120-$

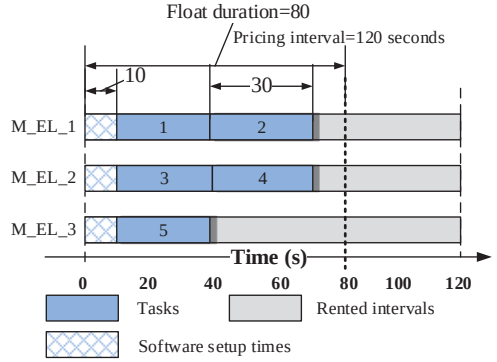

Fig. 3. Estimating the minimum numbers of VM instances and rented intervals for a given mode and UFD.

$10)-20\times5]+10\times3\}\times0.41/120=0.89$. A larger EWC usually means a lower utilization of rented intervals. Minimizing task-unit EWSs was therefore the optimization objective of the designed deadline division method.

4.2.2 UFD initialization

Initially, tasks chose VM types with the cheapest execution costs (if there were more than one VM type with the same execution cost, the one with the minimum EWC was chosen to break the tie), based on which UFDs were initialized. The execution cost of task v_i on the VM type δ was defined as $c_{i,t}=T^e_{i,\delta}\times P_\delta/\mathbf{L}$ without considering the waste resulting from interval based pricing models. Initially, the VM type with the cheapest execution cost was preferred for each task because a cheaper execution cost generally meant a higher match between the task and the VM type. For example, the execution time of a computation-intensive task on a high-memory VM was much longer than that on a high-CPU VM while the high-memory VM might not be cheaper than the high-CPU VM. Therefore, cheaper execution costs were translated into higher execution efficiency. Though better configured VMs had higher prices, execution cost could be the same because task execution was accelerated and the execution time was saved, e.g., task execution costs on the M_QEL, M_DEL, M_EL VMs might be the same. In this paper, the slowest VM type (usually the cheapest price per hour) was preferred because it helped to reduce the EWC. The reason for this was that it needed to consolidate more tasks to fully use the rented time intervals for better configured VMs.

The slowest VM type was chosen from the cheapest ones as the initial VM type for each task v_i, which is labeled by $\delta_i^{'}$. We called such VM type selection strategy as the cheapest-slowest selection (CSS for short) rule. Since tasks of the same task-unit had the same functionality and the same VM type preference, the current VM type of u was assigned as $\delta_i^{'}$, i.e., $\delta_u^{'}=\delta_i^{'}$ $(\forall v_i\in u)$. The longest processing time among tasks in u was calculated by

$$T^l_{u,\delta_u^{'}}=\max_{v_i\in u}\{T^e_{i,\delta_i^{'}}+\max_{j\in\mathcal{P}_i}\{\frac{\mathcal{Z}_{j,i}}{w^B}\}\} \quad (2)$$

Since all tasks in task-unit u used the same software, we let T^{ϖ}_u be the software setup time, i.e., $T^{\varpi}_u=T_{\varpi_i},\forall v_i\in u$. The UFD σ_u of task-unit u was initialized as

$$\sigma_u=T^l_{u,\delta_u^{'}}+T^{\varpi}_u \quad (3)$$

In addtion, the TFD σ_{v_i} of task v_i was initialized as $\sigma_{v_i}=\sigma_u, v_i\in u$.

4.2.3 UFD adjustment

After the initialization, the critical path generated by TFDs might be longer than the workflow deadline D. Some tasks on the critical path were needed to adjust by reassigning faster VM types to shorten the critical path. For a given task v_i, the current VM type $\delta_i^{'}$ was updated to the next slowest VM type $\delta_i^{''}$ using the CSS rule. At the same time, VM types of all the tasks of the task-unit were updated, i.e., $\delta_u^{''}=\delta_i^{''}$ $(v_i\in u)$ and the UFD of u was updated to $\sigma_u^{'}=T^l_{u,\delta_u^{''}}+T^{\varpi}_u$. It was obvious that $\mathcal{W}_{u,\delta_u^{''},\sigma_u^{'}}>\mathcal{W}_{u,\delta_u^{'},\sigma_u}$. For each task-unit u, we defined the increased ratio r_u as follows, which was the ratio of the increased EWC to the reduced UFD when the VM type was changed from $\delta_u^{'}$ to $\delta_u^{''}$

$$r_u=(\mathcal{W}_{u,\delta_u^{''},\sigma_u^{'}}-\mathcal{W}_{u,\delta_u^{'},\sigma_u})/(\sigma_u-\sigma_u^{'}) \quad (4)$$

If there is no faster VM type for u, r_u is set as $+\infty$.

The current TFDs determined a critical path CP. We let U_{CP} be the set of task-units with with at least one task on CP. Based on the calculated increased ratios, some UFDs were adjusted if the length of CP was longer than D by the following method: The task-unit $u^{'}$ was selected if $r_{u^{'}}<+\infty$ and $r_{u^{'}}$ was the minimal among U_{CP}. If there was more than one task-units with the same minimal increased ratio, the one with the largest EWC was selected to break the tie. If there was no $u^{'}$ satisfying the two conditions, no feasible solution could be found. Otherwise, the next slowest VM type is chosen by the CSS rule and $\sigma_{u^{'}}$ was updated by Equation (3). In addition, task TFDs of $u^{'}$ were updated and a new critical path CP' was generated. If the length of CP' is longer than D, the procedure was iterated until the length of the critical path was not longer than D. The UFD adjustment algorithm was formally described in Algorithm 1.

4.2.4 Adaptive workflow deadline division for task-units

The UFD initialization and adjustment procedures generated TFDs with a pessimistic way. The UFD of each task-unit u was assumed to be the largest processing time of its tasks under the VM type determined above. In addition, all tasks of u was supposed to be executed in parallel, i.e., the clustering granularity was 1. Therefore, there were many gaps between the earliest and the latest finish times of tasks in most of the time. An example was shown in Figure 4. EWCs of task-units could be decreased by distributing these gaps to UFDs (such as $\{v_4,v_5,v_6\}$ and $\{v_9,v_{10},v_{11}\}$).

In this paper, we proposed a new iterated gap distributing method which decreased task-unit EWCs by

Algorithm 1: UFD Adjustment Algorithm (UAA)

begin
  Initialize CP with TFDs;
  while $\sum_{v_i \in CP} \sigma_{v_i} > D$ **do**
    for each $u \in U_{CP}$ **do**
      Calculate r_u with Equation (4);
    Select u' with $r_{u'} = \min_{u \in U_{CP}} \{r_u\} \wedge r_{u'} \neq +\infty$;
    if $u' \neq$ *null* **then**
      Find the next slowest VM type $\delta''_{[1]}$ for the first task $v_{[1]} \in u'$ using the CSS rule;
      for each $v_i \in u'$ **do**
        $\delta'_i \leftarrow \delta''_{[1]}$;
      Update $\delta'_{u'}$ with $\delta'_{[1]}$;
      Calculating $\sigma_{u'}$ using Equation (3);
      Update the task TFDs of u' with $\sigma_{v_i} = \sigma_{u'}, \forall v_i \in u'$;
    else
      return *No feasible solution.*
    Update the new critical path CP and U_{CP} by the current TFDs ;
  return.

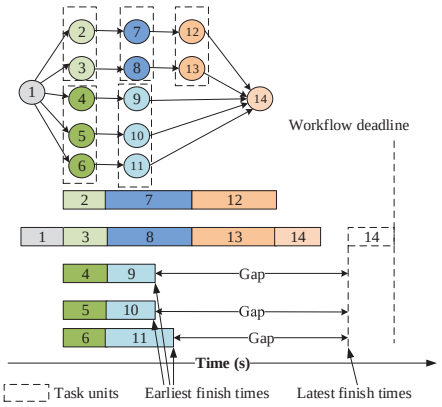

Fig. 4. An example of gaps for UFD increasing

increasing UFDs and kept VM types unchanged. In every iteration, we selected a task-unit u and increased the UFD from σ_u to $\sigma_u = \sigma_u + T^l_{u,\delta'_u}$, i.e.,

$$\sigma_u = T^{\varpi}_u + T^l_{u,\delta'_u} \times n_u \tag{5}$$

where $n_u \in \{1, 2, 3, ..., \lambda_u\}$ and λ_u was the number of tasks in u. Using this way of increasing UFDs, the minimum number of VMs for u was $M^m_{u,\delta'_u,\sigma_u} = \lceil \lambda_u / n_u \rceil$. Figure 5 showed an example of calculating EWCs when n_u ranged from 1 to 5. It was assumed that the length of the pricing interval was 120 seconds and the price of M_EL was 0.41 per interval. There were five tasks in unit $u = \{v_1, v_2, v_3, v_4, v_5\}$. For simplification, processing times of all tasks were assumed to be 30 seconds ($T^l_{u,M\_EL} = 30$) and the software setup time $T_{\varpi_i}, i \in \{1, 2, 3, 4, 5\}$ was 10 seconds. Figure 5 (a), (b), (c), (d) and (e) showed the procedure of estimating the minimum number of VM

Algorithm 2: Largest Return-Rate First (LRRF)

begin
  Initialize $\mathbf{F} \leftarrow \varnothing$;
  for each $u \in U$ **do**
    Calculate φ_u according to Equation (6);
  Initialize u' with $\varphi_{u'} = \min_{u \in U/\mathbf{F}} \{\varphi_u\}$;
  while $u' \neq$ *null* **do**
    Update $n_{u'} \leftarrow n_{u'} + 1$;
    Update $\sigma_{u'}$ according to Equation (5);
    Generate the current critical path CP;
    if $\sum_{v_i \in CP} \sigma_{v_i} > D$ **then**
      $\mathbf{F} \leftarrow \mathbf{F} \cup \{u'\}$, $n_{u'} \leftarrow n_{u'} - 1$ and update $\sigma_{u'}$;
    else
      Update $\varphi_{u'}$ according to Equation (6);
    Update u' with $\varphi_{u'} = \min_{u \in U/\mathbf{F}} \{\varphi_u\}$;
  return.

instances for $n_u = 1, 2, 3, 4$ and 5, respectively. According to Equation (1), EWCs of different n_u were:

(i) $\{1[5(1 \times 120 - 10) - 50 \times 3] + 10 \times 5\} 0.41/120 = 1.54$
(ii) $\{1[3(1 \times 120 - 10) - 50 \times 3] + 10 \times 3\} 0.41/120 = 0.72$
(iii) $\{1[2(1 \times 120 - 10) - 50 \times 3] + 10 \times 2\} 0.41/120 = 0.18$
(iv) $\{0[2(1 \times 120 - 10) - 50 \times 3] + 10 \times 2\} 0.41/120 = 0.08$
(v) $\{0[1(1 \times 120 - 10) - 50 \times 3] + 10 \times 1\} 0.41/120 = 0.03$

It was clear that $\mathcal{W}_{u,\delta'_u,\sigma_u}$ was a non-increasing function of n_u according to Equation (5). Since different task-units had different EWC functions, different EWCs were decreased by allocating the same length of gap to different task-units. When the UFD of u was increased from σ_u to σ'_u ($\sigma'_u - \sigma_u = T^l_{u,\delta'_u}$), the decreasing speed of the EWC of u was called return-rate defined as

$$\varphi_u = (\mathcal{W}_{u,\delta'_u,\sigma_u} - \mathcal{W}_{u,\delta'_u,\sigma'_u}) / (\sigma'_u - \sigma_u) \tag{6}$$

In terms of the obtained return-rates, all gaps were distributed to task-units by the proposed Largest Return-rate First method (LRRF). Initially, $n_u = 1$ for all task-units. Return-rates of all task-units were calculated according to Equation (6). The task-unit u' with the largest return-rate was chosen. We updated the UFD $\sigma_{u'}$ using Equation (5) by $n_{u'} \leftarrow n_{u'} + 1$. If the length of the critical path determined by the new TFDs was longer than D, it meant that the UFD increasing of u' resulted in a deadline violation and must be rolled back by $n_{u'} \leftarrow n_{u'} - 1$. In addition, u' was added to the tabu task-unit set $\mathbf{F}$, of which task-units' UFDs could not be increased any more. Otherwise, the return-rate $\varphi_{u'}$ was updated. Note that, there were no change on return-rates of the other task-units (except u'). Then, the task-unit with the largest return-rate in $U/\mathbf{F}$ was chosen as u' and we repeated the above process. If no task-unit was chosen, the procedure terminated. The procedure LRRF was formally described in Algorithm 2.

After increasing UFDs using Algorithm 2, we further distributed gaps between the earliest and the latest finish

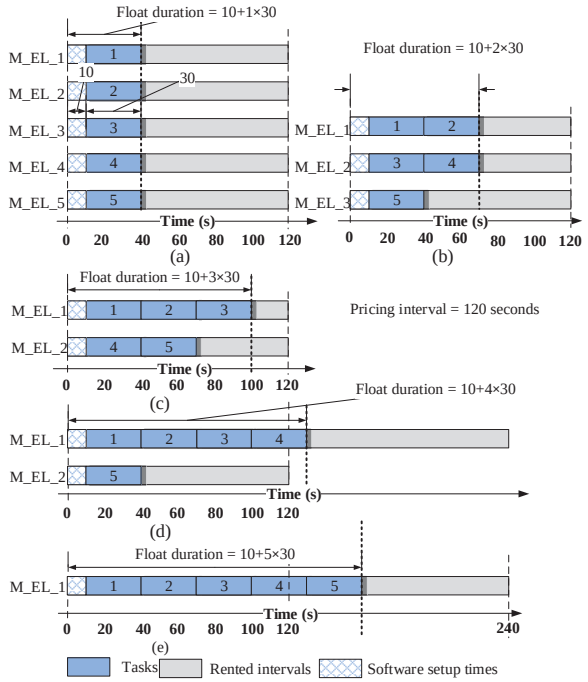

Fig. 5. Calculating EWCs

times to all tasks of the task-batch based workflow in proportion to their current TFDs and the TFDs were updated. Based on the updated TFDs, the earliest finish times of tasks were set as task deadlines, which were constraints for the following task scheduling problem.

4.3 Rule-based task scheduling

In this paper, we proposed a rule-based task scheduling strategy (RTSS). Tasks constrained by the obtained task deadlines were scheduled to appropriate time slots of existing or newly rented VM instances. Tasks were scheduled according to the topological order of the workflow. RTSS first got the task to be scheduled. Each task selected the appropriate time slot in terms of a hybrid rule which was based on four rules. By a right shifting method, tasks were scheduled to the best positions of the selected time slots.

The next task to be schedule was selected in terms of the following rules: Task-units with the smallest depth had the highest priority. If there are more than one task-units having the same depth (with different functionality), the task-unit with the largest total processing time had the highest priority. In the same task-unit, the task with the largest processing time is selected first. Let V^R be the set of ready tasks (all of their predecessors were scheduled).

Let $\Psi_{\mathcal{I}}^{v_i}$ be the set of time slots available (in the interval from the earliest start time to the deadline of v_i) for task v_i in the VM set $\mathcal{I}$. If v_i was assigned to the VM instance to which time slot s belonged, $x_{v_i,s}=1$; otherwise, $x_{v_i,s}=0$. The data transfer time of task v_j on time slot s was $T^d_{v_j,s}=\max_{i\in\mathcal{P}_j}\{\mathcal{Z}_{i,j}(1-x_{v_i,s})/w^B\}$. The bandwidth was assumed to be infinitely large and the data transfer time was 0 if the two tasks were on the same VM instance. In other words, only the data transfer between different VM instances was considered. If a new VM was needed to launch at time slot s, $y_s=1$; otherwise, $y_s=0$. Let T^m_s be the VM setup time for slot s. We obtained $T^m_s=y_sT^m_\delta$. If v_i was assigned to a time slot s without software ϖ_i, $z_{i,s}=0$; otherwise, $z_{i,s}=1$. $T^{\varpi}_{v_i,s}=(1-z_{i,s})T_{\varpi_i}$ was the software setup time of task v_i on time slot s.

To reduce the number of total newly rented intervals, the selected task v_i needing more than one newly rented time interval was tried to perform the right shifting on each time slot in $\Psi_{\mathcal{I}}^{v_i}$. Initially, v_i started at the earliest available start time on each time slot. It was tried to right shifted to start at the beginning of the second newly rented interval. Figure 6 illustrated an example of a reduction in the number of newly rented intervals by task right shifting, in which the scheduled v_1 was the only predecessor of v_2. The earliest start time of v_2 was 40. When calculating the combined priority value on a new high-CPU VM instance (C_EL), two intervals were newly rented when v_2 was assigned to start at 40 as shown in Figure 6 (a). If v_2 was moved right to start at 60, only one interval was needed, which was shown in Figure 6 (b). As a result, before calculating the priority value of a time slot, it should be tested to see whether the task right shifting could reduce the number of rented intervals or not.

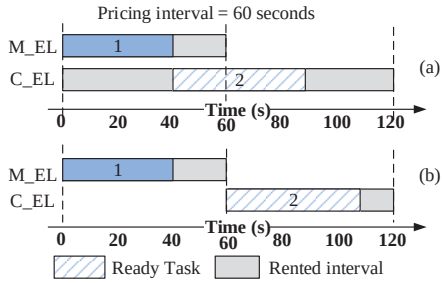

Fig. 6. Task right shifting

Time slots allocation is affected by several factors, such as the number of newly rented intervals, the price per interval, the execution time, the data transfer time, the software setup time, the VM loading time, the length-match between the task processing time and the length of the remaining rented intervals, and the utilization of rented intervals. These factors interact with each other [28]. Taking into account these factors, four rules were introduced to the RTSS. The first three were identical to those proposed in [28], which were described as follows.

(i) **FNIF** (Fewest amount of newly rented time intervals first): Let $R^s_{v_i}$ denote the number of newly rented intervals when v_i is assigned to $s\in\Psi_{\mathcal{I}}^{v_i}$. $\bar{R}^s_{v_i}$ is the maximum number of time intervals needed by v_i, which is determined by $\bar{R}^s_{v_i}=\lceil(T^e_{i,\delta_s}+T^d_{v_i,s}+T^m_s+T^{\varpi}_{i,s})/\mathbf{L}\rceil+1$ where δ_s is the VM type of s. The priority $\alpha^s_{v_i}$ is determined by the normalization $R^s_{v_i}/\bar{R}^s_{v_i}$.

(ii) **LACF** (Lowest Actual Cost First): The actual cost of v_i on s is $C^s_{v_i}=(T^e_{i,\delta_s}+T^d_{v_i,s}+T^m_s+T^{\varpi}_{v_i,s})\times P_{\delta_s}$. The priority value of this rule is normalized as $\beta^s_{v_i}=C^s_{v_i}/\max_{\acute{s}\in\Psi^{v_i}_{\mathcal{I}}} C^{\acute{s}}_{v_i}$. The time slot with cheaper $\beta^s_{v_i}$ is preferred.

(iii) **BMF** (Best match between the slot length and the task processing time first): The matching of the processing time of v_i and the length of s is defined as $\gamma^s_{v_i}=W^s_{v_i}/(2\times\mathbf{L})$ where $W^s_{v_i}$ is the remaining time (new produced smaller fractions) of v_i on s. The smallest $\gamma^s_{v_i}$ the first.

These rules are effective for task scheduling in workflows [28]. However, most parts of rented intervals were wasted because of the task deadline constraints for some cases. For example, in Figure 7, v_2, v_3, v_4, v_5, v_6 and v_7 belonged to unit u_k of which tasks had a unified deadline of 78 and the pricing interval was assumed to be 60. v_1 was the predecessor of all tasks in u_k. In Figure 7 (a), v_1, v_2 and v_3 were scheduled tasks and v_4 was the current ready task. When v_4 was scheduled, there were two choices: i) Rent a new interval on M_EL_1 and assign v_4 just following v_3 as shown in Figure 7 (b). ii) Rent a new VM instance and assign v_4 at the finish time of v_1 as shown in Figure 7 (c). According to the above three rules, the first choice was preferred because of the lower processing cost and better matching. However, most parts of the newly rented interval on M_EL_1 in Figure 7 (b) cannot be reused by later tasks of unit u_k because of the task deadlines. Another new VM instance M_EL_2 must be rented for v_5, v_6 and v_7. Therefore, the maximal predicted utilization first (**MPUF**) rule was proposed: We tried to pre-schedule as many as possible tasks in the task-unit to the newly rented interval to maximize utilization. The rate of unused fractions was called the predicted waste rate of the newly rented interval. For the example in Figure 7 (b), the fraction after v_4 was wasted since no more tasks of u_k could be assigned to it because of the task deadlines. In Figure 7 (c), after v_4 was assigned, v_5 could still be pre-scheduled after v_4. It was clear that the utilization of the rented intervals in Figure 7 (c) was significantly higher than that in Figure 7 (b). Let $\xi^s_{v_i}$ be the predicted waste rate for v_i on *slot*.

Since different rules had various advantages, we proposed a weighted priority rule for allocating tasks to time slots below.

$$\psi^s_{v_i}=\alpha^s_{v_i}\times a+\beta^s_{v_i}\times b+\gamma^s_{v_i}\times c+\xi^s_{v_i}\times d \tag{7}$$

The rule-based task scheduling method was formally described in Algorithm 3. For each ready task v_i, one VM instance $VM_{\delta'_i}$ of the type δ'_i was first added to the data center $\mathcal{I}$ temporally. Then we updated the earliest start time EST_{v_i} of v_i and the set of candidate time slots $\Psi^{v_i}_{\mathcal{I}}$. For each time slot, it was tested to see whether v_i could be right shifted to reduce the number of newly rented intervals. Then, the combined priority value $\psi^s_{v_i}$ on each time slot was calculated by Equation (7) and the time slot s' with the highest priority was chosen. Next, v_i was assigned to s' and $VM_{\delta'_i}$ was removed from the data center if it was not chosen. Finally, we updated V^R by adding tasks of which the predecessors were scheduled and got the next ready task. The procedure iterated until all tasks were scheduled.

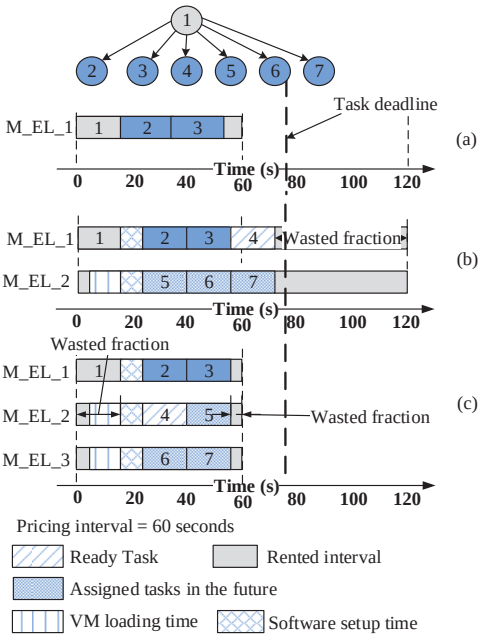

Fig. 7. Waste prediction on the rented intervals.

Algorithm 3: RTSS(a,b,c,d) /*Rule-based Task Scheduling Strategy*/

```
begin
    Initialize V^R ← {v_0} and s' ← null ;
    v_i ← Get the next ready task from V^R;
    while v_i ≠ null do
        Add a new VM instance VM_{δ'_i} to I;
        Update EST_{v_i} and Ψ^{v_i}_I;
        ψ^{low}_{v_i} ← +∞;
        foreach s in Ψ^{v_i}_I do
            Perform task right shifting on s;
            Calculate ψ^s_{v_i} according to Equation (7);
            if ψ^s_{v_i} < ψ^{low}_{v_i} then
                s' ← s;
                ψ^{low}_{v_i} ← ψ^s_{v_i};
        Assign v_i to s' and remove VM_{δ'_i} from I if it
        is not used;
        Update V^R;
        v_t ← Get the next ready task from V^R;
    return
```

4.4 Proposed Unit-aware Rule-based Heuristic

Based on the above procedures, the proposed unit-aware rule-based heuristic (URH) was formally described as

Algorithm 4: Unit-aware Rule-based Heuristic (URH)

```
1 begin
2     Combine tasks into task-units by the task-unit
      generation process;
3     Initialize VM types using the UFD initialization
      procedure;
4     Call UAA;
5     Call LRRF;
6     Distribute gaps to tasks in proportion to their
      TFDs;
7     Calculate the earliest finish times of tasks by
      TFDs and set them as task deadlines;
8     Call RTSS(a, b, c, d);
9     return
```

Algorithm 4. At first, tasks were merged into task-units according to their depths and functions. UFDs of the task-units were initialized by the cheapest and slowest VM types. Then, the UAA was called to adjust configurations of some task-units to obtain feasible UFDs. The LRRF procedure was conducted to fully decrease EWCs of all task-units by increasing their UFDs. Next, if there were still gaps, they were distributed to tasks in proportion to their TFDs. The earliest finish times of tasks were refreshed by the current TFDs. By adopting the earliest finish times as task deadlines, tasks were scheduled to appropriate time slots by the RTSS procedure.

The time complexity of the URH mainly depended on three parts, i.e., the UAA, the LRRF and the RTSS. In the UAA, time complexities of generating critical paths, calculating increased ratios and adjusting configuration of a task-units were $O(N^2)$, $O(N)$ and $O(M)$ respectively. There were at most $N \times M$ iterations at Step 2 of the UAA because each task could be updated M times at most. Therefore, the time complexity of UAA was $O(MN^3)$. For the LRRF, the time complexity of calculating the return-rate of all task-units was $O(N)$. The UFD of each task-unit u could be increased λ_u times, i.e., all units could be increased N times at most. Therefore, the time complexity of LRRF was $O(N^3)$. The time complexity of the RTSS was $O(M_s N)$, where M_s was the maximum number of available time slots in the system. Because tasks were non-preemptive, $M_s \leq N$. In other words, the time complexity of RTSS was no more than $O(N^2)$. Therefore, the time complexity of the proposed URH was O(MN^3).

5 Performance Evaluation of URH

To evaluate the performance, the URH was compared with the MRH [28] and the IC-PCP [1]. All these algorithms were coded in JaveEE and ran with a developed C-YARN simulator that extended the CloudSim [32] by adding C-YARN simulation components which supported software locality, DAG based modeling and elastic resource provisioning. The C-YARN simulator modeled the Amazon EC2 instance types shown in Table 1.

5.1 Tested workflows

Since the realistic workflows (including Montage, CyberShake, Epigenomics, LIGO, and SIPHT) studied by Bharathi et al. [33] were typical task-batch based workflows, they were used for the algorithms' evaluation in this paper. Therefore, the testing workflow instances were produced by the Workflow Generator of Bharathi et al. [33] (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator). The generated workflows were saved in XML formats, which provided network structures, task names and seed running times. They were extended according to the workflows' characteristics. The software needed by each task was determined by the task name described in the XML files. Tasks with the same type of software name and the same depth implied that they belonged to the same task-batch. In order to generate the execution time on different type of VMs, tasks with the same software requirement were assigned a unified category chosen from {*Normal*, *High-memory*, *High-CPU*}. Let $Q^M_{v_i}$ and $Q^C_{v_i}$ be the total memory and CPU workload of v_i. F^M_δ and F^C_δ represented the memory and CPU configurations of VM type δ described in Table 1. The execution time $seed_{v_i}$ described in the XML file was assumed to be the execution time on the N_S, M_EL and C_M types of instances when the category was *Normal*, *High-memory* and *High-CPU* respectively. $Q^M_{v_i} = F^M_{\delta_{t'}} \times seed_{v_i}$ and $Q^C_{v_i} = F^C_{\delta_{t'}} \times seed_{v_i}$ ($\delta_{t'}$ = N_S, M_EL and C_M for *Normal*, *High-memory* and *High-CPU*, respectively). The execution time of v_i on any VM type δ was $T^e_{i,\delta} = \max\{Q^M_{v_i}/F^M_\delta, Q^C_{v_i}/F^C_\delta\}$. Data transfers were described in the XML files which consisted of the volume of the data and the transferring directions. For the tested workflows, the number of tasks took values as follows: $\{50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$. 100 instances were randomly generated for each value and thus testing a total of 1100 instances. The test-bed generated in this paper is available on the website (http://www.seu.edu.cn/lxp/bb/06/c12114a113414/page.psp).

In the cloud simulation platform, the bandwidth took values from $\{1, 10, 100, 1000\}$ (MBps), the software setup time took values from $\{0, 5, 10, 15, 20\}$ (seconds) and the VM loading time was set as 30s. VM instances were priced by hours as done in the Amazon EC2. We let T^{min}_w be the shortest execution time of the tested workflow w with the fastest and sufficient amount of VM instances and T^{max}_w was the longest execution time of w with tasks being executed sequentially on only the slowest VM instance. We took the deadlines of each tested workflow from $T^{min}_w \times 2^n$, $n = 1, 2, 3, ..., L$ and $T^{min}_w \times 2^L < T^{max}_w$. 2^n was named as the deadline factor.

5.2 The competing algorithms

There is no other existing workflow scheduling algorithm proposed exactly for the same problem under studied in this paper. The IC-PCP proposed by Abrishami et al. [1] and the MRH in Cai et al. [28] were developed for traditional workflows, which are the most similar to the task-batch based worklows under study. For the sake of comparison, both IC-PCP and MRH were adapted for the considered applications without taking into account task-batches. In addition, the IC-PCP was modified to be aware of the operating system loading time and the software setup time as done in [28]. For a workflow instance w, the Relative Decreased Percentage (RDP) of the rental cost obtained by an algorithm A compared with IC-PCP was calculated by

$$RDP_w^A = (C_w^I - C_w^A) \times 100 / C_w^I \quad (8)$$

where C_w^I and C_w^A were the rental costs obtained by algorithms IC-PCP and A on instance w respectively. A larger RDP meant a lower resource rental cost.

There were several components (unit-aware deadline division, task right-shifting and RTSS) and parameters (a, b, c and d in RTSS) in the proposed URH. Because the same realistic workflows were verified, we adopted the same parameter values of a, b and c to those calibrated in [28], i.e., a=100, b=10, c=1. The weight d of MPUF was tested taking values from {0,1,10,100}. If d=0, it meant that the MPUF rule was not adopted. In addition, there was two variants for the right shifting operation, *true* or *false*. Based on levels of parameters or variants of components, we constructed six heuristics as shown in Table 2.

TABLE 2
Parameter settings of URH heuristics

Name	a	b	c	right shifting	d
URH_{00}	100	10	1	*false*	0
URH_{10}	100	10	1	*true*	0
URH_{03}	100	10	1	*false*	100
URH_{11}	100	10	1	*true*	1
URH_{12}	100	10	1	*true*	10
URH_{13}	100	10	1	*true*	100

5.3 Impacts of components and parameters on URH

5.3.1 Effectiveness of unit-aware deadline division

The experimental results were analyzed by the multi-factor analysis of variance (ANOVA) method [34]. The three main hypotheses (normality, homoskedasticity and independence of the residuals) were checked and accepted. The instance type and the deadline factor had the largest $F-$ratios which indicated that they had a statistically significant impact on the response variable RDP. Figure 8 showed the means plots of RDP (with 95% Turkey Honest Significant Difference (HSD) confidence intervals) with different deadline factors. In Figure 8, URH obtained a statistically significant larger RDP than MRH on CyberShake and Montage workflows, a little larger than MRH on LIGO and SIPHT workflows and similar with the MRH on Epigenomics workflows.

The reason for the better performance obtained by the URH on CyberShake, Montage, LIGO and SIPHT workflows was that these types of workflows had unbalanced workloads among different stages (batches), i.e., a significantly different number of tasks in each stage and the proposed unit-aware deadline division method could handle these unbalanced structures better. Figure 9 gave the structure of Montage workflows, of which the mProjectPP, mDiffFit and mBackground task-units had heavier workloads. The task deadlines (the largest one of the same unit) obtained by the MRH and the task-unit deadlines by URH_{00} on a realistic Montage workflow were given on the right of Figure 9, which showed that the MRH gave mProjectPP, mDiffFit and mBackground task-units extremely tighter deadlines than URH_{00}. This was because tasks deadlines in the MRH were determined separately according to the processing time of each task rather than by units as done in the URH_{00}. Gantt charts of the MRH and the URH_{00} on the Montage example were also given in Figure 9, which illustrated that properly allocated task-unit deadlines of mProjectPP and mDiffFit decreased the number of rented VM instances significantly (decreasing the VM rental cost from $12.16 to $1.87).

For the Epigenomics workflows, the URH_{00} and the MRH got similar results, which was because most tasks of Epigenomics workflows needed more than one hour (one pricing interval) and there was no space for the unit-aware deadline division method to improve, i.e., the proposed URH_{00} was more suitable for tasks with smaller execution times than the pricing interval length. If the rental interval was far smaller than the execution time, consolidation of tasks to improve the utilization of rented intervals was no longer important. However, the task-unit based deadline division still gave more chances to reuse software and saved the VM loading time. Figure 10 showed the means plots of RDP with 95% Tukey HSD confidence intervals, the interaction with the software setup time and the system bandwidth. The improved percentages of the URH_{00} compared with MRH for different software setup times and different bandwidths were similar, which indicated that the two factors (software setup time and the bandwidth) had little effect on the performance of both algorithms. As a result, these two factors were not discussed in the rest of this paper.

5.3.2 Effectiveness of task right shifting

Experimental results showed that the task right shifting worked better for workflows with longer tasks than others. This was because longer tasks gave more chances to apply the task right shifting. In the five types of tested workflows, right shifting only worked better on Epigenomics and SIPHT workflows. Figure 11 was the means plot of MRH, URH_{00} and URH_{10} (with 95% Tukey HSD confidence intervals) on Epigenomics and SIPHT

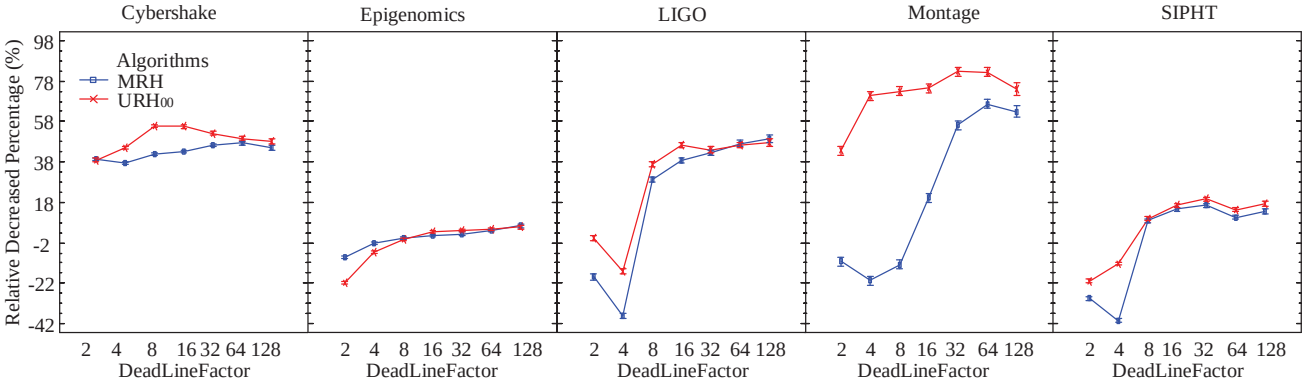

Fig. 8. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH and URH as a function of the DeadLinefactor and the workflow type.

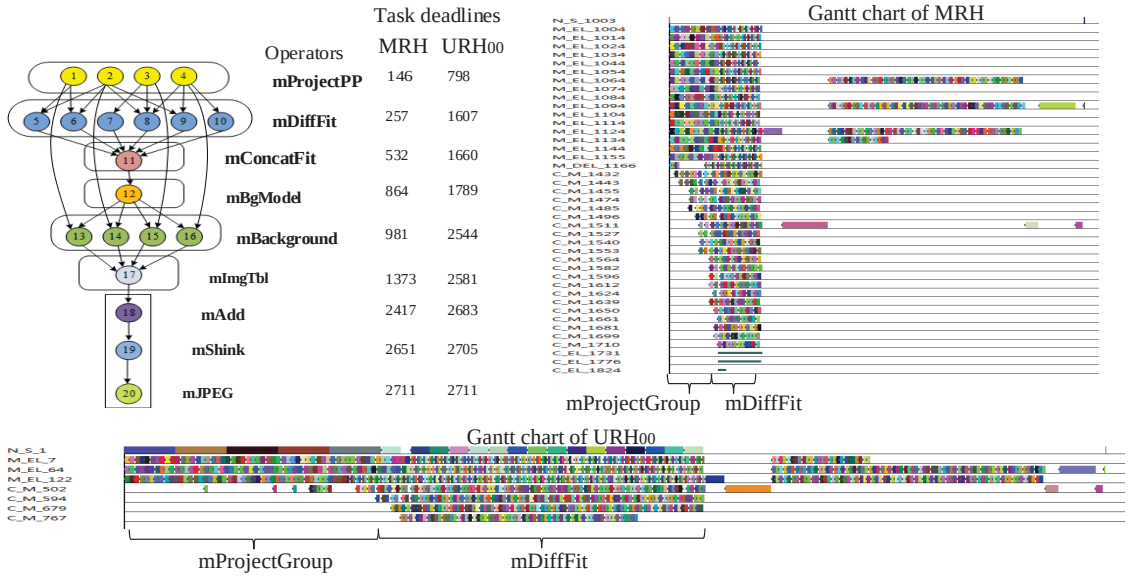

Fig. 9. Task deadlines and Gantt charts obtained by MRH/URH$_{00}$ for a Montage workflow.

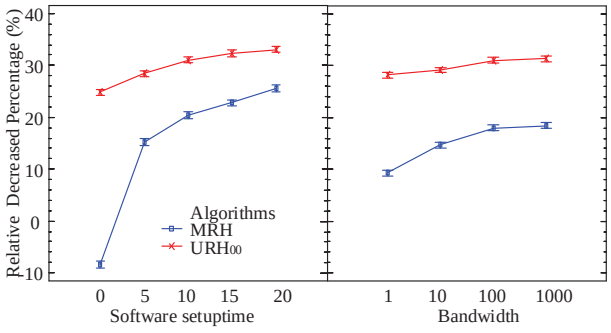

Fig. 10. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of the MRH and URH$_{00}$ as a function of software setup times and bandwidths.

workflows, which showed that the URH$_{10}$ got larger RDP, i.e., task right shifting helped in decreasing the rental cost. For example, Figure 12 showed the Gantt charts obtained by the URH$_{00}$ and the URH$_{10}$ for an Epigenomics workflow, in which tasks were scheduled as early as possible by the URH$_{00}$ while they were moved (delayed) to start at the beginning of the next pricing interval in the URH$_{10}$. About 5.5% of rental cost was saved by the URH$_{10}$ compared with URH$_{00}$ on this instance by adopting the task right shifting when selecting time slots.

5.3.3 *Effectiveness of the MPUF rule*

Experimental results showed that the MPUF rule worked better on workflows with deadlines which were longer than one pricing interval and especially for those with many shorter tasks such as CyberShake, LIGO and Montage workflows. Figure 13 showed the means plot of RDP obtained by URH$_{00}$ and URH$_{03}$ on CyberShake and LIGO workflows. It showed that the MPUF rule worked when the scheduling horizon was at least longer than one pricing interval (Deadlinefactor was larger than 8 for CyberShake and LIGO workflows). Figure 14 showed

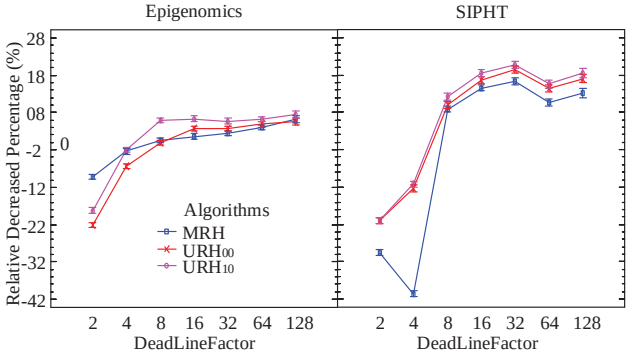

Fig. 11. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH, URH_{00} and URH_{10} as a function of the DeadLineFactor and the workflow type.

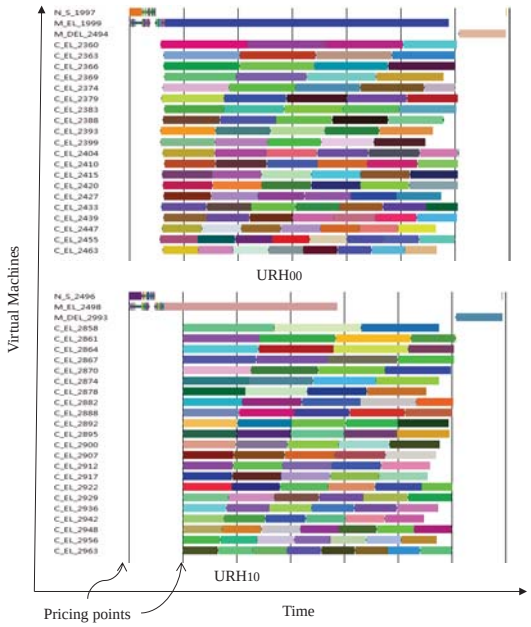

Fig. 12. Gantt chats of a Epigenomics workflow obtained by URH_{00} and URH_{10}

the Gantt chats obtained by URH_{00} and URH_{03} on a CyberShake workflow instance. Most parts of the second intervals were wasted because of the deadline constraint in the solution given by URH_{00} while the MPUF rule helped URH_{03} in avoiding renting the second intervals by predicting the utilization of newly rented intervals. The resource utilization of URH_{00} and URH_{03} on the example in Figure 14 were 47.9% and 73.7%, respectively. On the contrary, when the scheduling horizon was too loose, there was little chance for the MPUF rule to improve performance. This was because loose deadlines usually allowed all compared algorithms to fully consolidate tasks to improve the utilization of rented intervals.

Increasing the weight of the MPUF rule had a small impact on performance. The means plot obtained by URH_{11}, URH_{12} and URH_{13} were shown in Figure 15, which showed that only a small improvement was obtained when the weight of waste prediction was increased from 1 to 100 on CyberShake and LIGO workflows. This was because the MPUF rule rule was usually needed in scenarios where the priority values of the first three rules were almost the same for the candidate intervals. Therefore, the MPUF rule rule worked whenever its weight was larger than a specific value and performance could not be improved further even a larger weight was given.

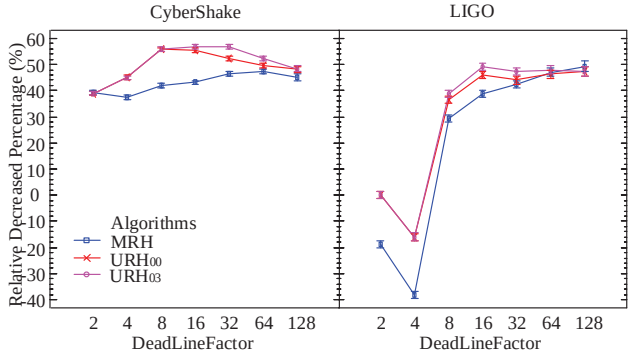

Fig. 13. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of algorithms as a function of the DeadLineFactor for the CyberShake and LIGO workflows.

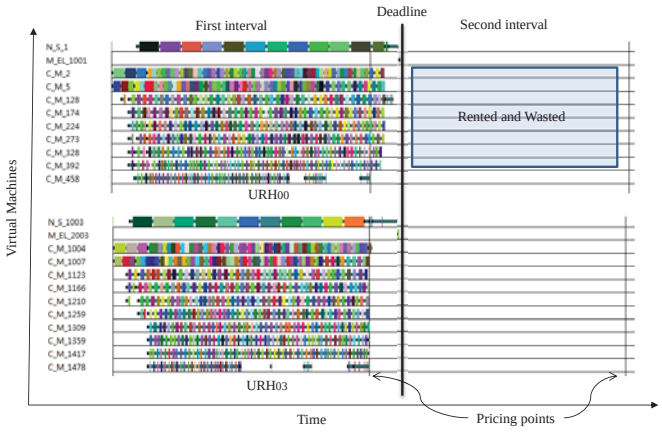

Fig. 14. Gantt chats of a CyberShake workflow obtained by URH_{00} and URH_{03}

5.4 Combined results

Experimental results indicated that there were interactions among task right shifting and the MPUF rule, i.e., the joint use of them could decrease the resource rental cost further. Figure 16 was the means plot of RDP of Epigenomics workflows, which showed that URH_{10} and URH_{03} were similar or a bit worse than the MRH when the deadline factor was 4. However, URH_{13} with the joint use of task right shifting and the MPUF rule was better than MRH for those instances. The means plot of all types of workflows in Figure 17 showed that URH_{13} got a statistically significant higher RDP than MRH in almost all cases, which also indicated that the combined use of unit-aware deadline division, task right shifting

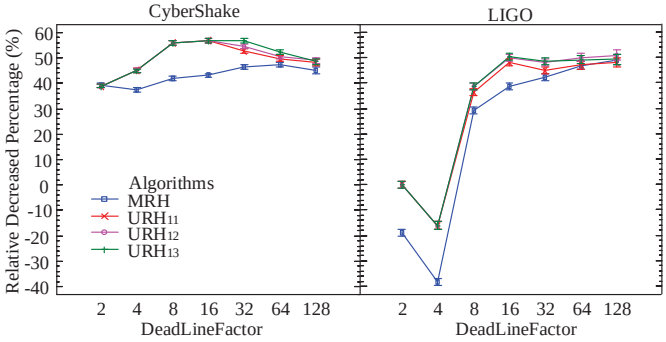

Fig. 15. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of URH with different predicting weights as a function of the DeadLineFactor for the CyberShake and LIGO workflows.

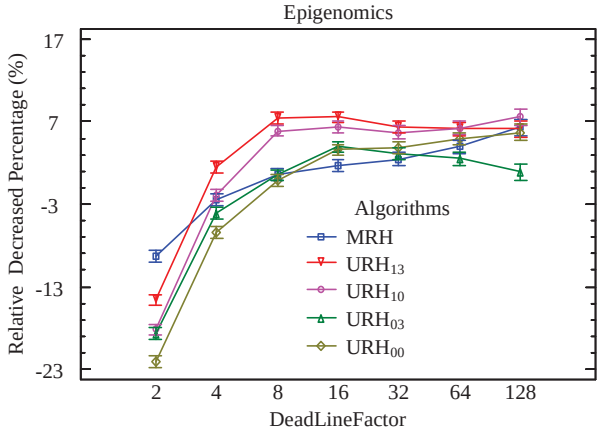

Fig. 16. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of algorithms as a function of the DeadLineFactor for the Epigenomics workflows.

and MPUF could decrease the VM rental cost in a very statistically significant way.

6 Conclusions

For elastically provisioning VM instances to task-batch based workflows in C-YARN, a fast and effective heuristic URH was proposed which took advantage of the task-unit based structures of realistic workflows to divide the workflow deadline. Experimental results showed that the unit-aware deadline division method was significantly better than traditional ones. In addition, a rule-based task scheduling method which consisted of task right shifting and waste prediction was developed to improve the performance even further. Experimental results indicated that task right shifting worked better on workflows with longer tasks while the MPUF rule could improve the performance of the proposals on workflows with scheduling horizons longer than the pricing interval. The combined use of the unit-aware deadline division, task right shifting and waste prediction resulted in a much better performance.

Comparing URH heuristics against the MRH, it could be concluded that deadline division had a statistically significant impact on the results. Therefore, developing appropriate deadline division methods according to the application characteristics was a promising line for future research.

7 Acknowledgments

This work has been supported by the National Natural Science Foundation of China (61272377) and the Doctoral Program of Higher Education (20120092110027). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "RESULT - Realistic Extended Scheduling Using Light Techniques" (No. DPI2012-36243-C02-01).

References

[1] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for iaas clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[2] J. J. Durillo and R. Prodan, "Multi-objective workflow scheduling in amazon EC2," *Cluster Computing*, vol. 17, no. 2, pp. 169–189, 2013.

[3] P. Uthayopas and N. Benjamas, "Impact of I/O and execution scheduling strategies on large scale parallel data mining," *Journal of Next Generation Information Technology*, vol. 5, no. 1, pp. 78–88, 2014.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[5] Y. Tabaa and A. Medouri, "Towards a next generation of scientific computing in the cloud," *International Journal of Computer Science Issues*, vol. 9, no. 6, pp. 177–183, 2012.

[6] Z. Cai, X. Li, and J. N. D. Gupta, "Heuristics for provisioning services to workflows in XaaS clouds," *Services Computing, IEEE Transactions on, doi: 10.1109/TSC.2014.2361320*, 2014.

[7] Mesos, "Apache mesos," *http://mesos.apache.org/*.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, 2007.

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

[10] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," *Computers, IEEE Transactions on*, vol. 61, no. 12, pp. 1737–1751, 2012.

[11] K. Chen, J. Powers, S. Guo, and F. Tian, "Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*.

[12] W. Chen, R. F. da Silva, E. Deelman, and R. Sakellariou, "Using imbalance metrics to optimize task clustering in scientific workflow executions," *Future Generation Computer Systems, doi:10.1016/j.future.2014.09.014*, 2014.

[13] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–12.

[14] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] A. Verma, L. Cherkasova, and R. H. Campbell, "Orchestrating an ensemble of mapreduce jobs for minimizing their makespan," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 5, pp. 314–327, 2013.

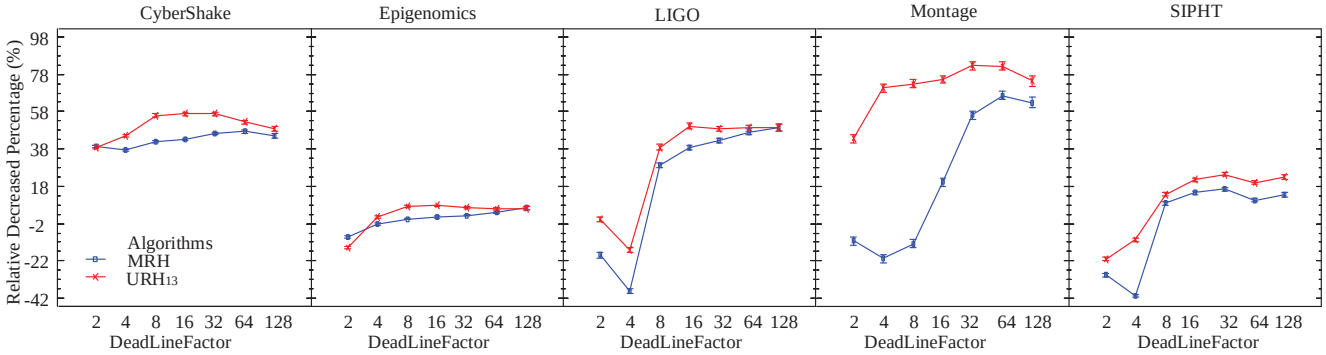

Fig. 17. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH and URH_{13} as a function of the DeadLineFactor for all workflows.

[16] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 129–139, 2010.

[17] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 107–118, 2004.

[18] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1107–1117, 2010.

[19] H. Hsiao, H. Chung, H. Shen, and Y. Chao, "Load rebalancing for distributed file systems in clouds," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 5, pp. 951–962, 2013.

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

[21] Q. Wu and Y. Gu, "Supporting distributed application workflows in heterogeneous computing environments," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*. IEEE, 2008, pp. 3–10.

[22] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1400–1414, 2012.

[23] E. Demeulemeester, W. Herroelen, and S. Elmaghraby, "Optimal procedures for the discrete time/cost trade-off problem in project networks," *European Journal of Operational Research*, vol. 88, no. 1, pp. 50–68, 1996.

[24] Y. Yuan, X. Li, Q. Wang, and X. Zhu, "Deadline division-based heuristic for cost optimization in workflow scheduling," *Information Sciences*, vol. 179, no. 15, pp. 2562–2575, 2009.

[25] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The performance of mapreduce: an in-depth study," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 472–483, 2010.

[26] E.-K. Byun, Y.-S. Kee, J.-S. Kim, E. Deelman, and S. Maeng, "Bts: Resource capacity estimate for time-targeted science workflows," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 848–862, 2011.

[27] R. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1787–1796, 2014.

[28] X. Li and Z. Cai, "Elastic resource provisioning for cloud workflow applications," *Technical report of southeast university 2014, http://www.seu.edu.cn/lxp/bb/06/c12114a113414/page.psp*.

[29] L. David and I. Puaut, "Static determination of probabilistic execution times," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE, 2004, pp. 223–230.

[30] M. A. Iverson, F. Ozguner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*. IEEE, 1999, pp. 99–111.

[31] J. Yu, R. Buyya, and C. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *e-Science and Grid Computing, 2005. First International Conference on*. IEEE, 2005, pp. 8–pp.

[32] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[33] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*. IEEE, 2008, pp. 1–10.

[34] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, *Experimental methods for the analysis of optimization algorithms*. Springer, 2010.