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

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

Abstract—To meet the dynamic workload requirements in widespread task-batch based workflow applications, we develop the cloud YARN (C-YARN) architecture by integrating the YARN platform with cloud computing. The C-YARN is able to provision flexible resources. In terms of depths and functions, tasks of different task-batches are merged into task-units. A unit-aware deadline division method is investigated for properly dividing workflow deadlines to task deadlines so as to minimize the utilization of rented intervals. A rule-based task scheduling method is 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) is proposed for elastically provisioning VMs to task-batch based workflows to minimize the rental cost in C-YARN. Effectiveness of the proposed URH methods is 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, 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 domains, 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; it turns to be the problem in [4] if there is only one constraint for each 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.

Recently, several more flexible cluster management systems (such as YARN [5] and Mesos [6]) have been developed. YARN gives the possibility of choosing more appropriate programming frameworks such as *Dryad* [7] 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. Different from 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) [8]. 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. Cardosa et al. [9] and Chen et al. [10] investigated the 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 propose the cloud YARN (C-YARN) with DAG based programming framework for task-batch based workflows. Cloud resources are elastically provisioned in terms of dynamic workloads.

Minimizing the rental cost and maximizing the resource utilization are crucial for cloud resource provisioning in the considered C-YARN system. 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 into a single task-unit. In existing methods for traditional workflows such as those proposed by Byun et al. [8], Mao et al. [11], Abrishami et al. [1] and Durillo et al. [2], task deadlines were generated for tasks separately

- *Zhicheng Cai work at the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: caizhicheng@njust.edu.cn).*
- *Zhicheng Cai and Xiaoping Li work at the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China (e-mail: xpli@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).*

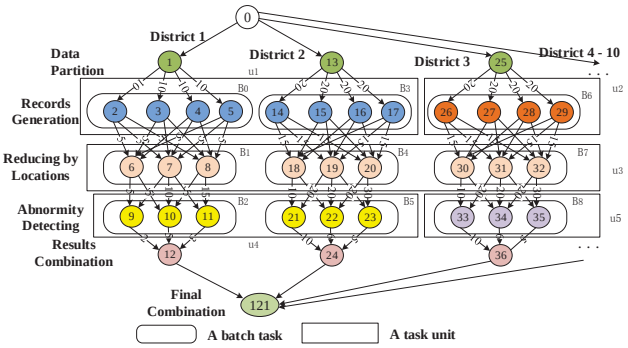


Fig. 1. An example of the task-batch based workflow

without considering task-batches. If these methods were applied to task-batch based workflows, imbalance in tasks deadlines would result. For example, tight task deadlines may 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) is proposed which distributes the workflow deadline to competitive task-units. The URH guides 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.

The key contributions of this paper are summarized as follows:

(i) By integrating cloud and the YARN platform, the C-YARN platform is presented for task-batch based workflows. C-YARN provisions elastic resources.
(ii) A unit-aware deadline division method is developed for properly dividing the workflow deadline into task deadlines. A rule-based task scheduling strategy is proposed.
(iii) A unit-aware rule-based heuristic is investigated for C-YARN resource provisioning in task-batch based workflows.

The rest of the paper is organized as follows. Section 2 summaarizes the related work and the problem is described in detail in Section 3. Section 4 presents the proposed heuristic which is evaluated in Section 5. Section 6 concludes the paper.

2 Related Work

Performance optimization of *MapReduce* and DAG based scheduling [12] on dedicated clusters with fixed capacities has been studied extensively in the literature [13], [14]. For DAG based tasks, many researchers considered the improvement of system performances of distributed systems with fixed capacities [15]–[19]. In traditional service-oriented computing, resources could be dynamically provisioned as services to workflow applications [20]–[22]. The adopted economic models 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 clusters or service-oriented computing are not suitable for task-batch based workflow scheduling in public clouds.

Resource provisioning for *MapReduce* and DAG-based applications in public clouds is of great interest. Chen et al. [10] 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 [23]. The Balanced Time Scheduling (BTS) algorithm, proposed by Byun et al. [24], minimized the client-oriented resource renting cost of DAG-based applications. The BTS assumed that there was a fixed number of homogeneous resources during the whole workflow execution horizon. In a later work by Byun et al. [8], 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 resources were homogeneous. Recently, Mao et al. [11] 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. [25] 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 [26], 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 no existing work on resource provisioning for task-batch based workflows. Though the considered problem is similar to that in [4] to some extend, they are completely different. The main differences are shown in Table 1.

TABLE 1
Comparing the considered problem against that in [4].

Problem	In this paper	In [4]
Tasks in a batch	each task except the source node has at least one predecessor	each batch except the source node has at least one predecessor
Cloud platform	cloud YARN	Hadoop 1.0
Scheduling unit	task	batch
Precedence	between tasks	between batches

3 Problem Formulation

3.1 The C-YARN system

In this paper, we introduce the C-YARN system for task-batch based workflow applications by deploying the existing YARN to public clouds, which includes two roles: the cloud provider and the cloud user. The IaaS cloud provider supplies Virtual Machines (VMs) to cloud users. Cloud users rent 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 [5], there is an additional component: the Elasticity Controller (EC) in the Resource Manager (RM) of the C-YARN.

The EC of the RM is 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 is submitted to the C-YARN system, the RM creates an Application Master (AM) for the application which is 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 is the AMScheduler in charge of making dynamic resource consumption plans (issuing 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 are designed for the AMScheduler to provision VMs to the task-batch based workflows. The objective is to minimize the total VM rental cost while meeting the workflow deadline $\mathcal{D}$.

3.2 Task-batch based workflow applications

Different types of Virtual Machines 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. Let P_δ be the price of VM type δ per interval unit. $\mathbb{L}$ is the pricing interval length. A task-batch based workflow is defined by a Directed Acyclic Graph (DAG) $G = (V, E, B)$. $V = \{v_0, v_1, \ldots, v_{N+1}\}$ is the set of 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. Assume there are Q batches, $\mathcal{B} = \{B_0, B_1, \ldots, B_Q\}$ 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, $\mathcal{B} = \{B_0, B_1, B_2, B_3, ...\}$ in which batches are: $B_0 = \{v_2, v_3, v_4, v_5\}$, $B_1 = \{v_6, v_7, v_8\}$, $B_2 = \{v_9, v_{10}, v_{11}\}$, etc.

Let $T^e_{i,\delta}$ denote the execution time of task v_i on VM instances of type δ. The fact that execution times (which can be estimated by some existing methods [27], [28]) of tasks are different on various VM instances makes allocation very difficult. We can assume that there is no limitation on the amount of VM instances in cloud. In addition, data transfer times are always non-negligible which are time-consuming and dependent on both 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 are required to execute tasks, which need times to setup. 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 setup time T_{ϖ_i}.

3.3 Challenges for the problem under study

In this paper, we consider 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 [4] 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 many factors. In [4], execution modes are predefined.

(iii) Task-batches in C-YARN are decomposable while those in [4] are undecomposable and each task-batch is scheduled as a whole. Workflow deadline division for task-level scheduling in this paper is much more complicated than that for batch-level scheduling in [4].

4 Proposed Heuristic

To minimize the total VM rental cost, all tasks of a workflow application are scheduled with the workflow deadline $\mathcal{D}$ and tasks' precedence constraints. Similar to existing workflow scheduling, the resource provisioning problem under study is divided into two sub-problems: deadline division and task scheduling. Workflow deadlines are partitioned into task deadlines with which tasks are scheduled. Because of the above challenges and new characteristics, existing workflow scheduling algorithms [4], [20], [26], [29] are not suitable for resource provisioning for task-batch based workflows in C-YARN.

In this paper, a Unit-aware Rule-based Heuristic (URH) is developed. We refer to the amount of time distributed to a task as the task float duration (TFD) and the amount of time distributed to or a task-unit

as the task-unit float duration (UFD). Estimated wasted cost (EWC) is 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 consists of two steps: Unit-aware deadline division and Rule-based task scheduling.

4.1 Unit-aware deadline division

All tasks in a task-batch based workflows are merged into task-units according to their depths and functionalities while their logical precedence constraints among them are still kept, i.e., there are precedence constraints between task-units. Basically, tasks of the same task-unit are allocated to the same deadline. For example, tasks of B_0 and B_3 in Figure 1 have the same depth and the same functionality. They are merged to task-unit u_1. Though the task depth of B_6 is identical to that of B_0, different kinds of software are needed. Therefore, B_0 and B_6 are not consolidated into a task-unit. Tasks of each task-unit are assigned the same deadline.

Tasks of a task-unit can be executed both in parallel to accelerate the execution process and sequentially to improve the utilization of rented time intervals when the deadline is not very tight. In other words, tasks are regrouped into 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 [30]. 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 sequential tasks there are, the larger possibility of maximizing the utilization of the rented intervals (minimize the final rental cost). 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. A larger UFD usually means 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 leads to smaller UFD of its predecessor or successor task-units. Therefore, it is crucial to determine how much time (UFD) is allocated to each task-unit under the $\mathcal{D}$ constraint. TFDs are obtained in terms of UFDs. The earliest finish times of tasks are calculated by TFDs and they are regarded as the final task deadlines.

4.1.1 Estimated wasted cost

Generally, distributing the same UFD to different task-units exerts a great influence on the final rental cost. We adopt the estimated wasted cost (EWC) to measure 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 is 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 are estimated based on the assumption that all VMs are newly rented and only tasks of u are scheduled to them. During the estimation procedure, task processing times of the task-unit are supposed to be identical, which are equal to the largest task processing time of the task-unit. For each VM instance, only the software setup time of the first task is taken into account. Figure 2 shows an example of the estimation of the minimum number of VM instances and rented intervals in which there are five tasks with the largest processing time of 20s on M_EL VMs and a given UFD of 80. Three M_EL VMs are needed and one time interval is newly rented on each VM instance when the software setup time is 10s. We define that the EWC of u with the execution mode $\Upsilon_{\delta,\sigma}$ is the total of the wasted rental cost of unused fractions on the rented intervals and the software setup cost, which is computed as follows.

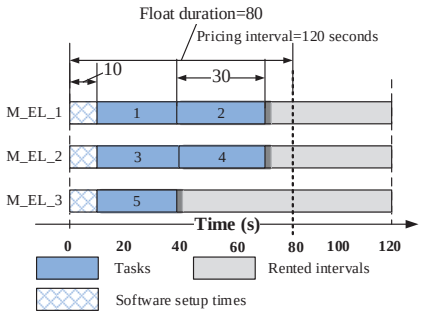


Fig. 2. Estimating the minimum numbers of VM instances and rented intervals for a given mode and UFD.

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

where $T^e_{u,\delta}$ is the total processing time of unit u, $M^m_{u,\delta,\sigma}$ and $I^m_{u,\delta,\sigma}$ are the minimum number of VM instances and rented intervals for $\Upsilon_{\delta,\sigma}$ respectively. g_σ is a binary variable. If $\sigma\leq\mathbb{L}$, $g_\sigma=1$. Otherwise, $g_\sigma=0$, i.e., when $\sigma>\mathbb{L}$, only the software setup cost is considered as the EWC. For the example in Figure 2, $\mathcal{W}_{u,M\_EL,80}=\{1\times[3\times(1\times120-10)-20\times5]+10\times3\}\times0.41/120=0.89$. A larger EWC usually means a lower utilization of rented intervals, i.e., task-unit EWSs are to be minimized.

4.1.2 UFD initialization

Initially, tasks choose VM types with the cheapest execution costs (if there are more than one VM type with the same execution cost, the one with the minimum EWC is chosen), based on which UFDs are initialized. The execution cost of task v_i on the VM type δ is defined as $c_{i,t}=T^e_{i,\delta}\times P_\delta/\mathbb{L}$ without considering the waste resulting from interval based pricing models. The VM type with the cheapest execution cost is preferred for each task because a cheaper execution cost generally means a higher match between the task and the VM type. Though better configured VMs have higher prices, execution cost might be the same because task execution is accelerated and the execution time is saved.

The slowest VM type is chosen from the cheapest ones as the initial VM type for each task v_i, which is labeled by $\delta_i^{'}$. We call such VM type selection strategy as the cheapest-slowest selection (CSS for short) rule. Since tasks of the same task-unit have the same functionality and the same VM type preference, the current VM type of u is assigned as $\delta_i^{'}$, i.e., $\delta_u^{'}=\delta_i^{'}$ $(\forall v_i \in u)$. The longest processing time among tasks in u is 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 use the same software, we let T_u^{ϖ} be the software setup time, i.e., $T_u^{\varpi}=T_{\varpi_i}$ $(\forall v_i \in u)$. The UFD σ_u of task-unit u is initialized as

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

The TFD σ_{v_i} of task v_i is initialized as $\sigma_{v_i}=\sigma_u$ $(v_i \in u)$.

4.1.3 UFD adjustment

After the initialization, the critical path generated by TFDs might be longer than $\mathcal{D}$. Some tasks on the critical path need adjustment by reassigning faster VM types to shorten the critical path. For a given task v_i, the current VM type $\delta_i^{'}$ is 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 are updated, i.e., $\delta_u^{''}=\delta_i^{''}$ $(v_i \in u)$ and the UFD of u is updated to $\sigma_u^{'}=T^l_{u,\delta_u^{''}}+T_u^{\varpi}$. It is obvious that $\mathcal{W}_{u,\delta_u^{''},\sigma_u^{'}}>\mathcal{W}_{u,\delta_u^{'},\sigma_u}$. For each task-unit u, we define the increased ratio r_u of the increased EWC to the reduced UFD when the VM type is changed from $\delta_u^{'}$ to $\delta_u^{''}$ as follows

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

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

The current TFDs determine a critical path *CP*. Let U_{CP} be the set of task-units with at least one task on *CP*. Based on the calculated increased ratios, some UFDs are adjusted if the length of *CP* is longer than $\mathcal{D}$ by: The task-unit $u^{'}$ is selected if $r_{u^{'}}<+\infty$ and $r_{u^{'}}$ is the minimal among U_{CP}. If there is more than one task-units with the same minimal increased ratio, the one with the largest EWC is selected. If there is no $u^{'}$ satisfying the two conditions, no feasible solution can be found. Otherwise, the next slowest VM type is chosen by the CSS rule and $\sigma_{u^{'}}$ is updated by Equation (3). In addition, task TFDs of $u^{'}$ are updated and a new critical path *CP*′ is generated. If the length of *CP*′ is longer than $\mathcal{D}$, the procedure is iterated until the length of the critical path is no longer than $\mathcal{D}$. The UFD adjustment algorithm is formally described in Algorithm 1.

4.1.4 Adaptive workflow deadline division for task-units

The UFD initialization and adjustment procedures generate TFDs in a pessimistic way. The UFD of each task-unit u is assumed to be the largest processing time of its tasks under the VM type determined above. In addition, all tasks of u is supposed to be executed in parallel, i.e., the clustering granularity is 1. Therefore, there might be many gaps between the earliest and the latest finish times of tasks most of the time. An example is shown in Figure 3. EWCs of task-units can be decreased by distributing these gaps to UFDs.

Algorithm 1: UFD Adjustment Algorithm (UAA)

```
begin
    Initialize CP with TFDs;
    while ∑_{v_i∈CP} σ_{v_i} > D do
        for each u∈U_CP do
            Calculate r_u with Equation (4);
        Select u' with r_{u'} = min_{u∈U_CP}{r_u} ∧ r_{u'} ≠ +∞;
        if u' ≠ null then
            Find the next slowest VM type δ''_[1] for the
            first task v_[1]∈u' using the CSS rule;
            for each v_i∈u' do
                δ'_i ← δ''_[1];
            Update δ'_{u'} with δ''_[1];
            Calculating σ_{u'} using Equation (3);
            Update the task TFDs of u' with
            σ_{v_i} = σ_{u'}, ∀v_i∈u' ;
        else
            return No feasible solution.
        Update the new critical path CP and U_CP by
        the current TFDs ;
    return.
```

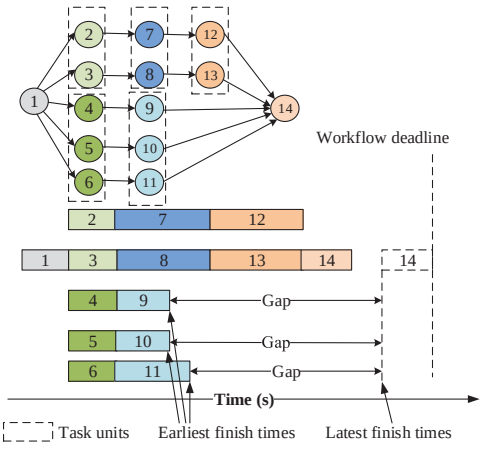


Fig. 3. An example of gaps for UFD increasing.

In this paper, we propose a new iterated gap distributing method which decreases task-unit EWCs by increasing UFDs while keeping VM types unchanged. In every iteration, we select a task-unit u and increase the UFD from σ_u to $\sigma_u=\sigma_u+T^l_{u,\delta_u^{'}}$, i.e.,

$$\sigma_u=T_u^{\varpi}+T^l_{u,\delta_u^{'}}\times n_u \quad (5)$$

where $n_u \in \{1,2,3,...,\lambda_u\}$ and λ_u is the number of tasks in u. Using this way of increasing UFDs, the minimum number of VMs for u is $M^m_{u,\delta_u^{'},\sigma_u}=\lceil \lambda_u/n_u \rceil$. Figure 4 shows

an example of calculating EWCs when n_u ranges from 1 to 5. It is assumed that the length of the pricing interval is 120s and the price of M_EL is 0.41 per interval. There are five tasks in unit $u=\{v_1, v_2, v_3, v_4, v_5\}$. For simplification, processing times of all tasks are assumed to be 30s ($T^l_{u,M\_EL}=30$) and the software setup time $T_{\varpi_i}, i \in \{1,2,3,4,5\}$ is 10s. Figure 4 (a), (b), (c), (d) and (e) show the procedure of estimating the minimum number of VM instances for $n_u=1,2,3,4$ and 5 respectively. According to Equation (1), their EWCs are $1.54, 0.72, 0.18, 0.08$ and 0.03 respectively. It is clear that $\mathcal{W}_{u,\delta'_u,\sigma_u}$ is a non-increasing function of n_u according to Equation (5). Since different task-units have different EWC functions, different EWCs are decreased by allocating the same length of gap to different task-units. When the UFD of u is increased from σ_u to σ'_u ($\sigma'_u-\sigma_u=T^l_{u,\delta'_u}$), the decreasing speed of the EWC of u is referred to as return-rate and defined as

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

In terms of the obtained return-rates, all gaps are 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 are calculated according to Equation (6). The task-unit u' with the largest return-rate is chosen. We update 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 is longer than $\mathcal{D}$, it means that the UFD increasing of u' results in a deadline violation and must be rolled back by $n_{u'} \leftarrow n_{u'}-1$. In addition, u' is added to the tabu set F, of which task-units' UFDs cannot be increased any more. Otherwise, the return-rate $\varphi_{u'}$ is updated. Note that, there are no change on return-rates of the other task-units (except u'). Then, the task-unit with the largest return-rate in $U \backslash F$ is chosen as u' and we repeat the above process. If no task-unit is chosen, the procedure terminates. The LRRF procedure is formally described in Algorithm 2.

Algorithm 2: Largest Return-Rate First (LRRF)

```
begin
    Initialize F ← ∅;
    for each u ∈ U do
        Calculate φ_u according to Equation (6);
    Initialize u' with φ_u' = min_{u∈U\F} {φ_u};
    while u' ≠ null do
        Update n_u' ← n_u' + 1;
        Update σ_u' according to Equation (5);
        Generate the current critical path CP;
        if Σ_{v_i∈CP} σ_{v_i} > D then
            F ← F ∪ {u'}, n_u' ← n_u' − 1 and update σ_u';
        else
            Update φ_u' according to Equation (6);
        Update u' with φ_u' = min_{u∈U\F} {φ_u};
return.
```

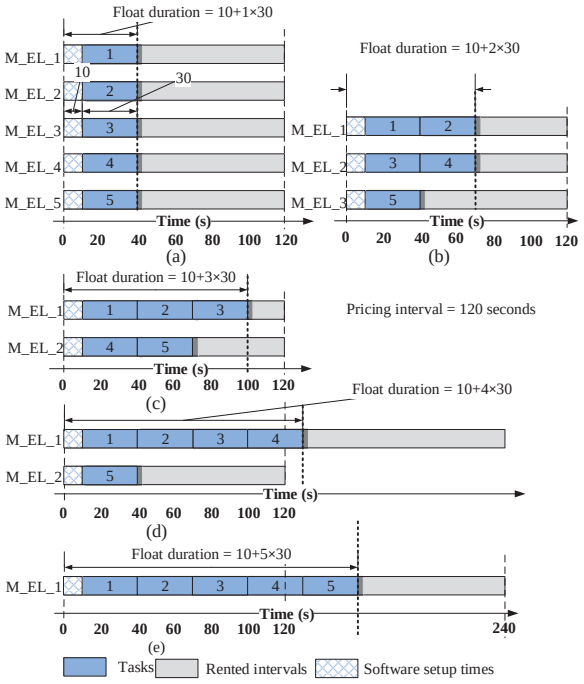


Fig. 4. The EWC calculating process of an instance.

After increasing UFDs using Algorithm 2, we further distribute gaps between the earliest and the latest finish times to all tasks of the task-batch based workflow in proportion to their current TFDs and the TFDs are updated. Based on the updated TFDs, the earliest finish times of tasks are set as task deadlines, which are constraints for the following task scheduling problem.

4.2 Rule-based task scheduling

In this paper, we propose a rule-based task scheduling strategy (RTSS). Tasks with the obtained deadlines are scheduled to appropriate time slots of existing or newly rented VM instances. Tasks are scheduled according to their topological order in the workflow. Through a right shifting method, tasks are scheduled to the best positions of the selected time slots.

The next task to be scheduled is selected in terms of the following rules: Task-units with the smallest depth have 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 has 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 are already scheduled).

Let $\Psi^{v_i}_{\mathcal{I}}$ 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 is assigned to the VM instance to which time slot s belongs, $x_{v_i,s}=1$; otherwise, $x_{v_i,s}=0$. The data transfer time of task v_j on time slot s is $T^d_{v_j,s}=\max_{i\in\mathcal{P}_j}\{\mathcal{Z}_{i,j}(1-x_{v_i,s})/w^B\}$. The bandwidth is assumed to be infinitely large and the data transfer time is 0 if the

two tasks are on the same VM instance. In other words, only the data transfer between different VM instances is considered. If a new VM is needed to launch at time slot s, $y_s=1$; otherwise, $y_s=0$. Let T_s^m be the VM setup time for slot s. We obtain $T_s^m=y_s T_\delta^m$. If v_i is assigned to a time slot s without software ϖ_i, $z_{i,s}=0$; otherwise, $z_{i,s}=1$. $T_{v_i,s}^{\varpi}=(1-z_{i,s})T_{\varpi_i}$ is 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 is right shifted on each time slot in $\Psi_{\mathcal{I}}^{v_i}$. Initially, v_i starts at the earliest available start time on each time slot. It is right shifted to the beginning of the second newly rented interval. Figure 5 illustrates an example of a reduction in the number of newly rented intervals by task right shifting, in which the scheduled v_1 is the only predecessor of v_2. The earliest start time of v_2 is 40. When calculating the combined priority value on a new high-CPU VM instance (C_EL), two intervals are newly rented when v_2 is assigned to start at 40 as shown in Figure 5 (a). If v_2 is moved right to start at 60, only one interval is needed, which is shown in Figure 5 (b). As a result, before calculating the priority value of a time slot, it should be tested to see whether the task right shifting can reduce the number of rented intervals or not.

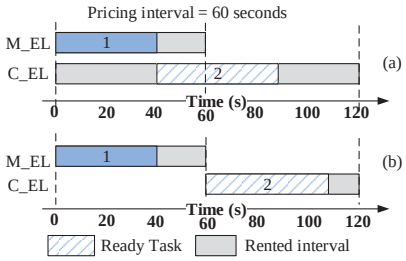


Fig. 5. An example for the task right shifting strategy.

Time slot allocation rules are affected by several factors and they interact with each other [26]. Though the three existing rules (FNIF, LACF and BMF) proposed in [26] are effective for task scheduling, most parts of rented intervals would be wasted because of task deadlines for some cases. Therefore, the Maximal Predicted Utilization First (MPUF) rule is proposed in this paper: We try to pre-schedule as many tasks as possible in the task-unit to the newly rented interval to maximize utilization. The rate of unused fractions is called the predicted waste rate of the newly rented interval.

To illustrate effectiveness of the MPUF rule, we give an example as shown in Figure 6. v_2, v_3, v_4, v_5, v_6 and v_7 belong to unit u_k of which tasks have a unified deadline of 78 and the pricing interval is assumed to be 60. v_1 is the predecessor of all tasks in u_k. In Figure 6 (a), v_1, v_2 and v_3 are scheduled tasks and v_4 is the current task. According to the three existing rules, a new interval is rented on M_EL_1 and assigned v_4 just following v_3 because of the lower processing cost and better matching as shown in Figure 6(b). However, most part of the newly rented interval on M_EL_1 cannot be reused by later tasks of unit u_k because of the task deadlines and another new VM instance M_EL_2 must be rented. If we adopt the MPUF, the tasks are scheduled as shown in Figure 6(c). It is clear that the utilization of the rented intervals in Figure 6(c) is significantly higher than that in Figure 6(b). We use $\xi_{v_i}^s$ to denote the predicted waste rate for v_i on s.

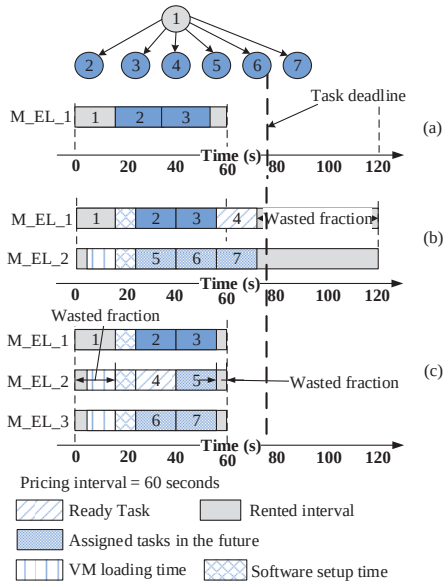


Fig. 6. Waste prediction on the rented intervals.

Since different rules have various advantages, we obtain a weighted priority for allocating tasks to time slots as below.

$$\psi_{v_i}^s=\alpha_{v_i}^s\times a+\beta_{v_i}^s\times b+\gamma_{v_i}^s\times c+\xi_{v_i}^s\times d \tag{7}$$

The rule-based task scheduling method is formally described in Algorithm 3. For each ready task v_i, one VM instance $VM_{\delta_i'}$ of the type δ_i' is first added to the data center $\mathcal{I}$ temporally. We update the earliest start time EST_{v_i} of v_i and the set of candidate time slots $\Psi_{\mathcal{I}}^{v_i}$. For each time slot, we test whether v_i can be right shifted to reduce the number of newly rented intervals. Then, the weighted priority value $\psi_{v_i}^s$ on each time slot is calculated by Equation (7) and the time slot s' with the highest priority is chosen. Next, v_i is assigned to s' and $VM_{\delta_i'}$ is removed from the data center if it is not used. Finally, we update V^R by adding tasks of which the predecessors are scheduled and get the next ready task. The procedure iterates until all tasks are scheduled.

4.3 Unit-aware Rule-based Heuristic

Based on the above procedures, the proposed unit-aware rule-based heuristic (URH) is formally described as Algorithm 4. At first, tasks are merged into task-units according to their depths and functions. UFDs of the task-units are initialized by the cheapest and slowest VM

Algorithm 3: RTSS(a,b,c,d)

begin
  Initialize $V^R \leftarrow \{v_0\}$ and $s^{'} \leftarrow$ null ;
  $v_i \leftarrow$Get the next ready task from V^R;
  while $v_i \neq$ *null* **do**
    Add a new VM instance $VM_{\delta_i^{'}}$ to $\mathcal{I}$;
    Update EST_{v_i} and $\Psi_{\mathcal{I}}^{v_i}$;
    $\psi_{v_i}^{low} \leftarrow +\infty$;
    foreach s *in* $\Psi_{\mathcal{I}}^{v_i}$ **do**
      Perform task right shifting on s;
      Calculate $\psi_{v_i}^{s}$ according to Equation (7);
      if $\psi_{v_i}^{s} < \psi_{v_i}^{low}$ **then**
        $s^{'} \leftarrow s$, $\psi_{v_i}^{low} \leftarrow \psi_{v_i}^{s}$;
    Assign v_i to $s^{'}$ and remove $VM_{\delta_i^{'}}$ from $\mathcal{I}$ if it is not used;
    Update V^R;
    $v_t \leftarrow$Get the next ready task from V^R;
  return

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

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

types. Then, the UAA is called to adjust configurations of some task-units to obtain feasible UFDs. The LRRF procedure is conducted to fully decrease EWCs of all task-units by increasing their UFDs. Next, if there are still gaps, they are distributed to tasks in proportion to their TFDs. The earliest finish times of tasks are refreshed by the current TFDs. By adopting the earliest finish times as task deadlines, tasks are scheduled to appropriate time slots by RTSS.

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

5 Performance Evaluation of URH

To evaluate the performance, the URH is compared with the MRH [26] and the IC-PCP [1]. All these algorithms are coded in JaveEE and ran with a developed C-YARN simulator that extended the CloudSim [31] 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.

5.1 Tested workflows

The realistic workflows Montage, CyberShake, Epigenomics, LIGO, and SIPHT studied by Bharathi et al. [32] are typical task-batch based workflows. Therefore, they are used for the algorithms' evaluation in this paper. The testing workflow instances are produced by the Workflow Generator of Bharathi et al. [32] (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator). The generated workflows are saved in XML formats, which provided network structures, task names and seed running times. They are extended according to the workflows' characteristics. The software needed by each task is determined by the task name described in the XML files. Tasks with the same type of software name and the same depth imply that they belong to the same task-batch. In order to generate the execution time on different type of VMs, tasks with the same software requirement are assigned a unified category chosen from {*Normal*, *High-memory*, *High-CPU*}. Let $Q_{v_i}^M$ and $Q_{v_i}^C$ be the total memory and CPU workload of v_i. F_δ^M and F_δ^C represent the memory and CPU configurations of VM type δ. The execution time $seed_{v_i}$ described in the XML file is assumed to be the execution time on the N_S, M_EL and C_M types of instances when the category is *Normal*, *High-memory* and *High-CPU* respectively. $Q_{v_i}^M = F_{\delta_{t'}}^M \times seed_{v_i}$ and $Q_{v_i}^C = F_{\delta_{t'}}^C \times seed_{v_i}$ ($\delta_{t'} =$ 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 is $T_{i,\delta}^e = \max\{Q_{v_i}^M / F_\delta^M, Q_{v_i}^C / F_\delta^C\}$. Data transfers are 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 takes values as follows: $\{50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$. 100 instances are randomly generated for each value and thus testing a total of 1100 instances.

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

5.2 The competing algorithms

There are no other existing workflow scheduling algorithms for the problem studied in this paper. The IC-PCP proposed by Abrishami et al. [1] and the MRH in Cai et al. [26] were developed for traditional workflows. They are similar to the task-batch based worklows. For the sake of comparison, both IC-PCP and MRH are adapted for the considered applications without taking into account task-batches. In addition, the IC-PCP is modified to be aware of the operating system loading time and the software setup time as done in [26]. For a workflow instance w, the Relative Decreased Percentage (RDP) of the rental cost obtained by an algorithm A compared with IC-PCP is calculated by

$$\text{RDP}_w^A = (C_w^I - C_w^A)/C_w^I \times 100\% \quad (8)$$

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

There are 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 are used, we adopted the same parameter values of a, b and c to those calibrated in [26], i.e., a=100, b=10, c=1. The weight d of MPUF is tested taking values from {0,1,10,100}. If d=0, it means that the MPUF rule is not used. In addition, there is two variants for the right shifting operation, *true* or *false*. Based on levels of parameters or variants of these components, we construct 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 are analyzed by the multifactor analysis of variance (ANOVA) method [33]. The three main hypotheses (normality, homoskedasticity and independence of the residuals) are checked and accepted. After the analysis, the instance type and the deadline factor have the largest $F-$ratios which indicate that they have a statistically significant impact on the response variable RDP. Figure 7 shows the means plots of RDP (with 95% Turkey Honest Significant Difference (HSD) confidence intervals) with different deadline factors.

In Figure 7, 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 to MRH on Epigenomics workflows. The reason for the better performance obtained by URH on CyberShake, Montage, LIGO and SIPHT workflows is that these types of workflows have 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 can handle these unbalanced structures much better.

For the Epigenomics workflows, URH_{00} and MRH got similar results, which is because most tasks of Epigenomics workflows need more than one hour (one pricing interval) and there is no space for the unit-aware deadline division method to improve, i.e., the proposed URH_{00} is more suitable for tasks with smaller execution times than the pricing interval length. If the rental interval is far smaller than the execution time, consolidation of tasks to improve the utilization of rented intervals is no longer important. However, the task-unit based deadline division still gives more chances to reuse software and saved the VM loading time. Figure 8 shows 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 are similar, which indicates that the two factors (software setup time and the bandwidth) have little effect on the performance of both algorithms. As a result, these two factors are not discussed in the rest of this paper.

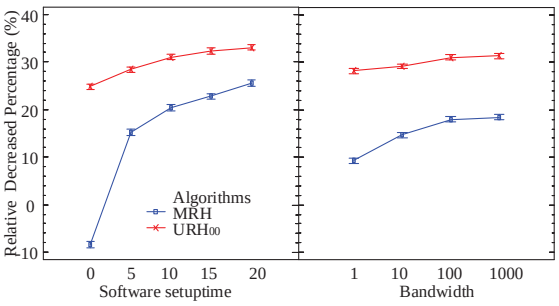


Fig. 8. The means plot of RDP with 95.0% Tukey HSD confidence Intervals of the MRH and URH_{00} as a function of software setup times and bandwidths.

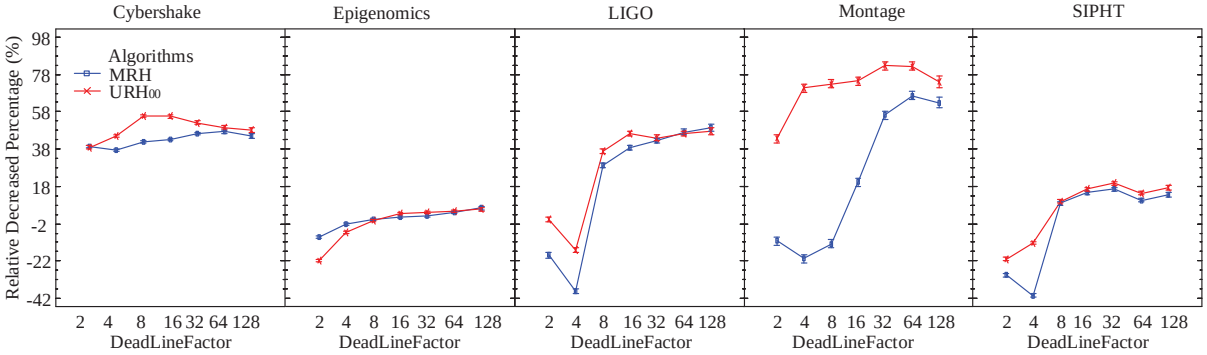


Fig. 7. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH and URH as a function of the DeadLinefactor and the workflow type.

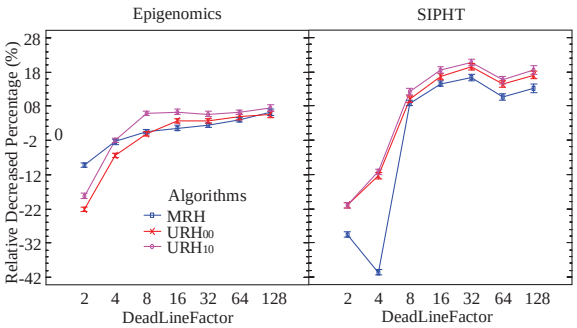


Fig. 9. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH, URH_{00} and URH_{10} as a function of the DeadLineFactor and the workflow type.

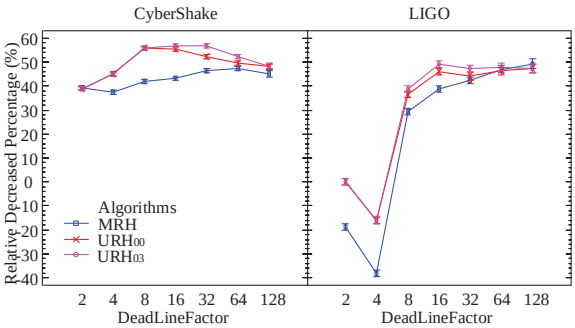


Fig. 10. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of algorithms as a function of the DeadLineFactor for the CyberShake and LIGO workflows.

5.3.2 *Effectiveness of task right shifting*

Experimental results show that the task right shifting works better for workflows with longer tasks. This is because longer tasks give more chances to apply the task right shifting. In the five types of tested workflows, right shifting only works better on Epigenomics and SIPHT workflows. Figure 9 shows the means plot of MRH, URH_{00} and URH_{10} (with 95% Tukey HSD confidence intervals) on Epigenomics and SIPHT workflows, which shows that URH_{10} gets larger RDP values, i.e., task right shifting helps in decreasing the rental cost.

5.3.3 *Effectiveness of the MPUF rule*

Experimental results show that the MPUF rule worked better on workflows with deadlines which are longer than one pricing interval and especially for those with many shorter tasks such as CyberShake, LIGO and Montage workflows. Figure 10 shows the means plot of RDP obtained by URH_{00} and URH_{03} on CyberShake and LIGO workflows. It shows that the MPUF rule works when the scheduling horizon is at least longer than one pricing interval (Deadlinefactor is larger than 8 for CyberShake and LIGO workflows). On the contrary, when the scheduling horizon is too loose, there is little chance for MPUF rule to improve performance. This is because loose deadlines usually allow all compared algorithms to fully consolidate tasks to improve the utilization of rented intervals.

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

5.4 Combined results

Experimental results indicate that there are interactions among task right shifting and the MPUF rule, i.e., the joint use of them can decrease the resource rental cost even further. Figure 12 shows the means plot of RDP of Epigenomics workflows, which shows that URH_{10} and URH_{03} are similar or a bit worse than the MRH when the deadline factor is 4. However, URH_{13} with the joint use of task right shifting and the MPUF rule is better than MRH for those instances. The means plot of all types of workflows in Figure 13 shows that URH_{13}

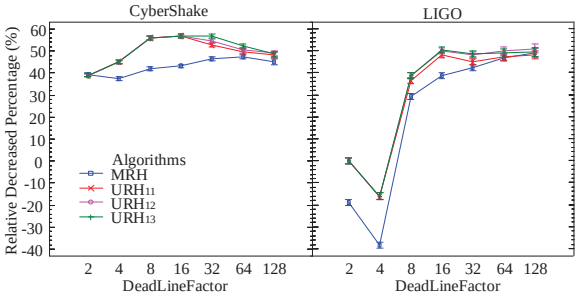


Fig. 11. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of URH with different predicting weights as a function of the DeadLineFactor for the CyberShake and LIGO workflows.

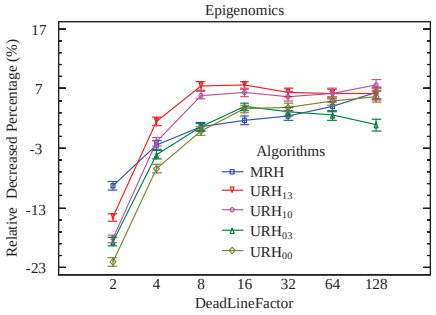


Fig. 12. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of algorithms as a function of the DeadLineFactor for the Epigenomics workflows.

gets a statistically significant higher RDP than MRH in almost all cases, which also indicates that the combined use of unit-aware deadline division, task right shifting and MPUF can 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 has been proposed which takes advantage of the task-unit based structures of realistic workflows to divide the workflow deadline. Experimental results have showed that the unit-aware deadline division method is significantly better than traditional ones. In addition, a rule-based task scheduling method which consists of task right shifting and waste prediction has been developed to improve the performance even further. Experimental results indicate that task right shifting works better on workflows with longer tasks while the MPUF rule can 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 the proposed URH heuristics against the existing MRH, it can be concluded that deadline division has a statistically significant impact on the results. Therefore, developing appropriate deadline division methods according to the application characteristics is a promising line for future research.

7 Acknowledgments

This work has been supported by the National Natural Science Foundation of China (61272377). 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] 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.

[5] V. K. Vavilapalli, A. C. Murthy, C. Douglas, and et al., "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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*, vol. 25, no. 6, pp. 1403–1421.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

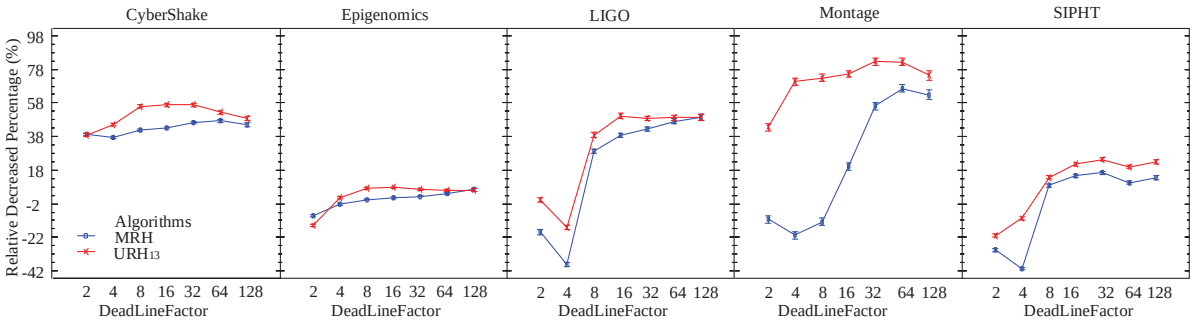


Fig. 13. Means plot of RDP with 95.0% Tukey HSD confidence Intervals of MRH and URH_{13} as a function of the DeadLineFactor for all workflows.

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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*.

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

[28] 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, Eighth*. IEEE, 1999, pp. 99–111.

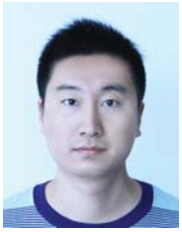
[29] J. Yu, R. Buyya, and C. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *First International Conference on e-Science and Grid Computing*. IEEE, 2005, pp. 8–pp.

[30] 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.

[31] 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.

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

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



Zhicheng Cai received his B.Sc. and Ph.D. degrees in Computer Science and Engineering from Southeast University, Nanjing, China, in 2009 and 2015 respectively. He is currently an assistant professor at the School of Computer Science and Engineering of Nanjing University of Science and Technology. His research interests focus on Load Prediction, Dynamic Capacity Management, Task Scheduling in clouds. He is the author of several publications in international journals such as *IEEE Transactions on Services Computing*, *IEEE Transactions on Automation Science and Engineering* and at conferences such as *ICSOC*, *ICPADS*, *SMC* and *CASE*.

Xiaoping Li (M'09-SM'12) received his B.Sc. and M.Sc. degrees in Applied Computer Science from the Harbin University of Science and Technology, Harbin, China, in 1993 and 1999 respectively. He obtained his Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology, Harbin, China, in 2002. He joined Southeast University, Nanjing, China, in 2005, and is currently a full professor at the School of Computer Science and Engineering. He is the author or co-author over more than 100 academic papers, some of which have been published in international journals such as *IEEE Transactions on Automation Science and Engineering*, *IEEE Transactions on Services Computing*, *Omega*, *European Journal of Operational Research*, *Information Sciences*, *Expert Systems with Applications*.

Rubén Ruiz is full professor of Statistics and Operations Research at the Polytechnic University of Valencia, Spain. He is co-author of more than 50 papers in International Journals and has participated in presentations of more than a hundred papers in national and international conferences. He is editor of the Elseviers journal *Operations Research Perspectives (ORP)* and co-editor of the JCR-listed journal *European Journal of Industrial Engineering (EJIE)*. He is also associate editor of other important journals like *TOP* or *Applied Mathematics and Computation* as well as member of the editorial boards of several journals most notably *European Journal of Operational Research* and *Computers and Operations Research*. He is the director of the Applied Optimization Systems Group (SOA, http://soa.iti.es) at the Instituto Tecnolgico de Informtica (ITI, http://www.iti.es) where he has been principal investigator of several public research projects as well as privately funded projects with industrial companies. His research interests include scheduling and routing in real life scenarios.