Error-Tolerant Resource Allocation and Payment Minimization for Cloud System

Sheng Di, *Member*, *IEEE*, and Cho-Li Wang, *Member*, *IEEE*

Abstract—With virtual machine (VM) technology being increasingly mature, compute resources in cloud systems can be partitioned in fine granularity and allocated on demand. We make three contributions in this paper: 1) We formulate a deadline-driven resource allocation problem based on the cloud environment facilitated with VM resource isolation technology, and also propose a novel solution with polynomial time, which could minimize users' payment in terms of their expected deadlines. 2) By analyzing the upper bound of task execution length based on the possibly inaccurate workload prediction, we further propose an error-tolerant method to guarantee task's completion within its deadline. 3) We validate its effectiveness over a real VM-facilitated cluster environment under different levels of competition. In our experiment, by tuning algorithmic input deadline based on our derived bound, task execution length can always be limited within its deadline in the sufficient-supply situation; the mean execution length still keeps 70 percent as high as user-specified deadline under the severe competition. Under the original-deadline-based solution, about 52.5 percent of tasks are completed within 0.95-1.0 as high as their deadlines, which still conforms to the deadline-guaranteed requirement. Only 20 percent of tasks violate deadlines, yet most (17.5 percent) are still finished within 1.05 times of deadlines.

Index Terms—VM multiplexing, resource allocation, convex optimization, prediction error tolerance, payment minimization

1 Introduction

Cloud computing [1], [2] has emerged as a compelling paradigm for the deployment of ease-of-use virtual environment on the Internet. One typical feature of clouds is its pool of easily accessible virtualized resources (such as hardware, platform or services) that can be dynamically reconfigured to adjust to a variable load (scale). All the resources provisioned by cloud system are supposed to be under a payment model [2], in order to avoid users' overdemand of their resources against their true needs.

Each task's workload is likely of multiple dimensions. First, the compute resources in need may be multiattribute (such as CPU, disk-reading speed, network bandwidth, etc.), resulting in multidimensional execution in nature. Second, even though a task just depends on one resource type like CPU, it may also be split to multiple sequential execution phases, each calling for a different computing ability and various price on demand, also leading to a potentially high-dimensional execution scenario.

The resource allocation in cloud computing is much more complex than in other distributed systems like Grid computing platform. In a Grid system [3], it is improper to share the compute resources among the multiple applications simultaneously running atop it due to the inevitable mutual performance interference among them. Whereas, cloud systems usually do not provision physical hosts directly to users, but leverage virtual resources isolated by VM technology [4], [5], [6]. Not only can such an elastic resource usage way adapt to user's specific demand, but it can also maximize resource utilization in fine granularity and isolate the abnormal environments for safety purpose. Some successful platforms or cloud management tools leveraging VM resource isolation technology include Amazon EC2 [7] and OpenNebula [8]. On the other hand, with fast development of scientific research, users may propose quite complicated demands. For example, users may wish to minimize their payments when guaranteeing their service level such that their tasks can be finished before deadlines. Such a deadline-guaranteed resource allocation with minimized payment is rarely studied in literatures. Moreover, inevitable errors in predicting task workloads will definitely make the problem harder.

Based on the elastic resource usage model, we aim to design a resource allocation algorithm with high prediction-error tolerance ability, also minimizing users' payments subject to their expected deadlines.

Since the idle physical resources can be arbitrarily partitioned and allocated to new tasks, the VM-based divisible resource allocation could be very flexible. This implies the feasibility of finding the optimal solution through convex optimization strategies [9], unlike the traditional Grid model [10] that relies on the indivisible resources like the number of physical cores. However, we found that it is inviable to directly solve the necessary and sufficient condition to find the optimal solution, a.k.a., Karush-Kuhn-Tucker (KKT) conditions [9]. Our first contribution is devising a novel approach (with only $O(n \cdot R^2)$ time complexity) to solve the problem, where R denotes the

- S. Di is with the MESCAL Group, INRIA, Room 213, Laboratoire LIG, ENSIMAG antenne de Montbonnot ZIRST, 51, avenue Jean Kuntzmann, 38330 Monbonnot Saint Martin, France. E-mail: disheng222@gmail.com.
- C.-L. Wang is with the Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong. E-mail: clwang@cs.hku.hk.

Manuscript received 29 Feb. 2012; revised 27 Aug. 2012; accepted 30 Sept. 2012; published online 19 Oct. 2012.
Recommended for acceptance by V.B. Misic, R. Buyya, D. Milojicic, and Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-02-0179.
Digital Object Identifier no. 10.1109/TPDS.2012.309.

1045-9219/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

number of execution dimensions and n is the system scale (the number of compute nodes).

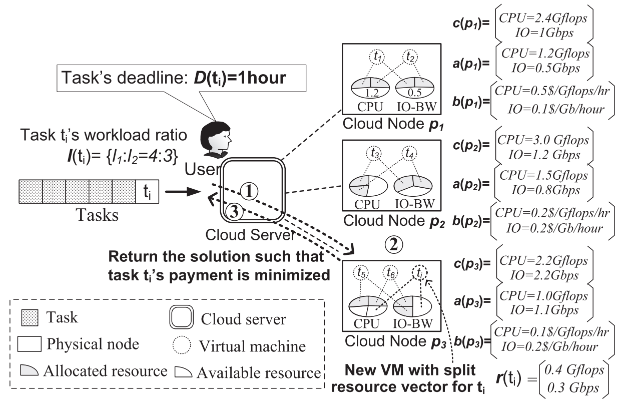
In literatures, traditional optimization problems are often subject to the precise prediction of task's characteristic (or execution property), which is nontrivial to realize in practice. Accordingly, as the state of the art, we further analyze our algorithm's optimality approximation ratio given the possibly wrong predictions of tasks' execution properties. In particular, we will try to answer such a question: when application's characteristic is predicted with certain levels of errors, will the application's final execution length (a.k.a., execution time) violate (or surpass) its deadline? If yes, what is the ratio of the final execution time to its deadline? These theoretical results will be significantly valuable to the guarantee of user's service level in practice. In fact, by setting a relatively stricter deadline properly based on our derived approximation ratio, each task can be guaranteed to be finished within its original deadline even though task properties cannot be predicted accurately.

In addition to the above theoretical contribution, we further confirm the effectiveness of our solutions by implementing a set of advanced web services that are based on complex matrix-operations, over a real cluster environment with 60 virtual machines. All the theoretical conclusions are confirmed with our experiments. Specifically, in the situation with relatively sufficient resources, the worst case tasks under the *stricter deadline-based allocation* only take as about 0.75 times as their deadlines to complete, as compared to the 1.2 times of the deadlines under the original *user-predefined deadline based allocation*. We also observe that in the competitive environment, the latter algorithm performs much more stable than the former instead, which means that the latter tolerates the resource competition better. We also confirm the effectiveness of our solution via the distribution of the number of tasks with respect to execution times and user payments: in the competitive situation, majority of tasks can be guaranteed to be completed within deadlines.

The rest of the paper is organized as follows: In Section 2, we formulate our problem based on the cloud scenario which supports elastic divisible resource customization. In Section 3, we first discuss the complexity of the modeled problem in brief, and then formally describe a novel algorithm, which can minimize user's payment based on task's preset execution deadline. In Section 4, we intensively derive the lower bound and upper bound of execution time for the situation with the possibly skewed predictions on tasks' properties as compared to the deadlines. We rigorously implement our algorithm and analyze experimental results on a real-cluster setting in Section 5. We discuss the related works in Section 6 and conclude with future work in Section 7.

2 Problem Formulation

In cloud systems, the cloud proxy (a.k.a., server) continually receives and responds to user requests (or tasks) with customized requirements (or virtual machines). All tasks will be handled based on their priorities (like Google task scheduler [11]) or in terms of First-Come-First-Serve (FCFS) policy when the tasks are of the same priorities (like [12]). Each task's execution may involve multidimensional resources, such as CPU and disk I/O. A data mining task, for example, usually needs to load a large set of data from disk before or in the middle of its computation. Eventually, such a task may store its computation results onto the local disk or a public server through network. Fig. 1 illustrates the procedure in processing such a task (denoted t_i). Suppose the task's execution times cost on computation and disk processing are predicted as 4 and 3 hours, respectively. Upon receiving the request, the scheduler checks the precollected availability states of all candidate nodes, and estimates the minimal payment of running the task within its deadline on each of them (i.e., Step 1 in the figure). The host (Node p_3 shown in Fig. 1) that requires the lowest payment will run the task via a customized VM instance with isolated resources (Step 2 in Fig. 1). Specifically, the VM will be customized with such a CPU rate (e.g., 0.4 Gflops) and disk I/O rate (e.g., 0.3 Gbps) that the task can be finished within its deadline ($D(t_i) = 1$ hour in the example) and its user payment can also be minimized meanwhile. Finally (Step 3), its computation results (or feedbacks) will be returned to users.



Task's deadline: D(t_i)=1hour
Task t_i's workload ratio
l(t_i)= {l_1:l_2=4:3}
User
t_i
Tasks
①
③
②
Cloud Server
Return the solution such that task t_i's payment is minimized
t_1
t_2
1.2
0.5
CPU
IO-BW
Cloud Node p_1
t_3
t_4
CPU
IO-BW
Cloud Node p_2
t_5
t_6
t_i
CPU
IO-BW
Cloud Node p_3
c(p_1)= CPU=2.4Gflops IO=1Gbps
a(p_1)= CPU=1.2Gflops IO=0.5Gbps
b(p_1)= CPU=0.5$/Gflops/hr IO=0.1$/Gb/hour
c(p_2)= CPU=3.0 Gflops IO=1.2 Gbps
a(p_2)= CPU=1.5Gflops IO=0.8Gbps
b(p_2)= CPU=0.2$/Gflops/hr IO=0.2$/Gb/hour
c(p_3)= CPU=2.2Gflops IO=2.2Gbps
a(p_3)= CPU=1.0Gflops IO=1.1Gbps
b(p_3)= CPU=0.1$/Gflops/hr IO=0.2$/Gb/hour
New VM with split resource vector for t_i
r(t_i) = 0.4 Gflops 0.3 Gbps
Task
Cloud server
Physical node
Virtual machine
Allocated resource
Available resource

Fig. 1. Resource allocation in cloud system.

Suppose there are n compute nodes (denoted by p_i, where $1 \le i \le n$). Since all the resources are managed centrally, the availability state of each resource within any recent or later period can be predicted prior, for executing any given task with multiple execution dimensions. For any particular task with R execution dimensions, we use Π to denote the whole set of dimensions and $\boldsymbol{c}(p_i) = (c_1(p_i), c_2(p_i), \ldots, c_R(p_i))^T$ as node p_i's capacity vector on these dimensions (In the paper, we use bold type to indicate a vector). In Fig. 1, for example, node p_1's physical capacity vector is $\boldsymbol{c}(p_1) = \{\text{CPU} = 2.4\text{Gflops}, \text{disk\_IO} = 1\text{Gbps}\}$.

Any user's task is denoted as t_i, where $1 \le i \le m$, and m refers to the total number of submitted tasks. Each task has a multidimensional *workload vector*, denoted by $\boldsymbol{l}(t_i) = (l_1(t_i), l_2(t_i), \ldots, l_R(t_i))^T$, which needs to be finished before the task's deadline. We denote the resource vector allocated to t_i as $\boldsymbol{r}(t_i) = (r_1(t_i), r_2(t_i), \ldots, r_R(t_i))^T$, where $r_k(t_i) (k = 1, 2, \ldots, R)$ refers to the resource amount on kth execution dimension isolated by hypervisor/virtual machine monitor (VMM) for the task's execution. Node p_i's availability vector (denoted $\boldsymbol{a}(p_i)$) along the multiple dimensions is calculated by $\boldsymbol{c}(p_j) - \sum_{t_i \, running \, on \, p_j} \boldsymbol{r}(t_i)$. For example, node

p_1 in Fig. 1 is running two VMs that are allocated with half of the total physical resources, so its availability vector $\boldsymbol{a}(p_1) = \{\text{CPU} = 1.2 \text{ Gflops}, \text{disk\_IO} = 0.5 \text{ Gbps}\}$. If there are no workloads being executed simultaneously for a particular task, its total execution time will be the sum of the individual processing times on different dimensions. If the execution of the workloads overlap, however, the task's completion time would be shorter. Accordingly, t_i's final execution time (denoted as $T(t_i)$) is definitely confined within such a range $[\max(\frac{l_k}{r_k}), \sum_{k=1}^{R} \frac{l_k}{r_k}]$. For simplicity, we denote task t_i's execution time as (1) (affine transformation of $\sum_{i=1}^{R} \frac{l_k}{r_k}$), where θ denotes a constant coefficient. Such a definition specifies a defacto broad set of applications each with multiple execution dimensions. The typical example is a single job with multiple sequentially interdependent tasks or some program with distinct execution phases each relying on independent compute resources (where $\theta = 1$).

$$T(t_i) = \theta \sum_{k=1}^{R} \frac{l_k(t_i)}{r_k(t_i)},\ where\ \theta \in \left[\frac{\max(\frac{l_k}{r_k})}{\sum_{k=1}^{R} \frac{l_k}{r_k}}, 1 \right]. \tag{1}$$

For any cloud system, the resources provisioned are usually set with a price vector denoted as $\boldsymbol{b}(p_i) = (b_1(p_i), b_2(p_i), \ldots, b_R(p_i))^T$ along R dimensions. $b_k(p_i)$ $(1 \leq k \leq R)$ denotes the per-time-unit price that the consumers need to pay for the consumption of the kth dimension on p_i. Each task t_i is set with a deadline (denoted $D(t_i)$) for its execution and the payment is expected to be minimized under our algorithm.

In our cloud model, any task will be executed on one or more virtual machines with *user-reserved* resources and the payment is calculated based on the customized resource (a.k.a., pay-by-reserve policy). Adopting such a pricing policy is driven by three reasons. First, the efficiencies of many applications usually rely on multiple resources but it is nontrivial to precisely evaluate the exact amount of their consumption separately on individual resources. Second, quite a few users prefer to reserving resources for tolerating usage burst and guaranteeing their service levels. Lastly, the alternative pricing policy, pay-as-you-consume, is rather simple because its payment is always fixed $(= \theta \sum_{k=1}^{R} (b_k(p_s) \cdot r_k(t_i) \frac{l_k(t_i)}{r_k(t_i)}) = \theta \sum_{k=1}^{R} b_k(p_s) \cdot l_k(t_i))$ regardless of the resource allocation.

Based on the pay-by-reserve policy, task t_i's total payment will be calculated via (2), where p_s refers to t_i's execution node. The mean price (i.e., $\frac{1}{R}\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}(t_i)$) will be used as the pricing unit over time, for computing user's payment. Such a design can be consistent with our pay-by-reserve model, and also prevent users from feeling too costly when their applications' execution cannot overlap at different dimensions

$$P(\boldsymbol{r}(t_i)) = \frac{1}{R}\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}(t_i) \cdot T(t_i). \tag{2}$$

In this paper, we might omit the notations t_i and p_i if thus would not cause ambiguity. For instance, $l_k(t_i)$, $\boldsymbol{r}(t_i), b_k(p_i), \boldsymbol{a}(p_i)$, and $D(t_i)$ may be substituted by $l_k, \boldsymbol{r}, b_k, \boldsymbol{a}$, and D respectively, in the following text.

Our research could be briefly summarized as the following convex optimization format: for any task t_i with its workload vector $\boldsymbol{l}(t_i)$, given a set of candidate execution nodes $(p_s, s = 1, 2, \ldots, n)$, how to select p_s and split resources such that t_i's payment (i.e., (2)) is minimized, subject to

$$Min\ \ P(\boldsymbol{r}(t_i))$$
$$s.t.$$

$$T(t_i) \leq D(t_i) \tag{3}$$

$$\boldsymbol{r}(t_i) \preceq \boldsymbol{a}(p_s). \tag{4}$$

3 Optimal Resource Allocation

In this section, we will first analyze the problem mentioned above, and then propose our optimal solution.

By combining (1) and (2), it is easy to verify that $\forall r_k$, $\frac{\partial^2 P(\boldsymbol{r}(t_i))}{\partial r_k^2} = \frac{\theta}{R}(-\frac{2b_k l_k}{r_k^2} + \frac{2l_k}{r_k^3}\sum_{i=1}^{R} b_i r_i) > 0$; thus, the target function $P(\boldsymbol{r}(t_i))$ is convex, which means that there must exist a minimal extreme point.

Based on the convex optimization theory [9], the Lagrangian function of the problem could be formulated as (5), where λ and $\mu_1, \mu_2, \ldots, \mu_R$ are corresponding Lagrangian multipliers. Note that θ is a constant defined in (1) and r is the abbreviation of $\boldsymbol{r}(t_i)$ as stated above

$$\begin{aligned} F_1(\boldsymbol{r}) = &\frac{1}{R}\left(\sum_{k=1}^{R} b_k r_k\right)\left(\theta \sum_{k=1}^{R} \frac{l_k}{r_k}\right) + \lambda\left(\theta \sum_{k=1}^{R} \frac{l_k}{r_k} - D\right) \\ &+ \sum_{k=1}^{R} \mu_k (r_k - a_k). \end{aligned} \tag{5}$$

Accordingly, we could get the Karush-Kuhn-Tucker conditions [9] (i.e., the necessary and sufficient condition of the optimization) as below:

$$\begin{cases} \lambda \geq 0, \mu_k \geq 0, k = 1, 2 \ldots, R \\ \displaystyle\sum_{i=1}^{R} \theta \frac{l_i}{r_i} \leq D \\ \lambda\left(\theta \displaystyle\sum_{i=1}^{R} \frac{l_i}{r_i} - D\right) = 0 \\ r_k \leq a_k(p_s), k = 1, 2, \ldots, R; s = 1, 2, \ldots, n \\ \mu_k(r_k - a_k(p_s)) = 0, k = 1, 2, \ldots, R; s = 1, 2, \ldots, n \\ \dfrac{\partial F_1}{\partial r_k} = \dfrac{1}{R}\left(\left(\displaystyle\sum_{i=1}^{R} b_i r_i\right) \cdot \dfrac{-l_k}{r_k^2} + b_k \cdot \displaystyle\sum_{i=1}^{R} \frac{l_i}{r_i} + \dfrac{-\lambda l_k}{r_k^2} + \mu_k\right) = 0. \\ \quad k = 1, 2, \ldots, R \end{cases} \tag{6}$$

In other words, as long as we can find such an allocation case $(\boldsymbol{r} = (r_1, r_2, \ldots, r_R)^T)$ to satisfy the above conditions simultaneously, we can set it as the optimal solution of the deadline-driven payment-minimized problem. However, it is nontrivial to do that, because the last condition $(\frac{\partial F_1}{\partial r_k} = 0)$ cannot be directly solved. Whereas, we exploit a novel algorithm with polynomial time complexity $(n \cdot R^2)$ to allocate resource, which can be proved to satisfy the KKT condition listed above.

Our algorithm is designed based on such a discovery: if we do not consider the limit of resource capacities (i.e., condition (4)), the problem can be directly solved using Lagrangian multiplier method. As follows, we will first derive the optimal solution to the problem with unbounded capacities (i.e., without the condition (4)) in Theorem 1. And then, we will describe our algorithm by recursively using Theorem 1 to search the resource allocation case that satisfies the whole KKT condition (6), in polynomial time.

Theorem 1. *For a specific task t_i, in order to minimize $P(\boldsymbol{r}(t_i))$ subject to the constraint (3), the optimal resource vector $\boldsymbol{r}^{(*)}(t_i)$ is (7), where $k = 1, 2, \ldots, R$. (Note that $\boldsymbol{r}^{(*)}(t_i)$ is not subject to Inequality (4), unlike the notation $\boldsymbol{r}^{*}(t_i)$ that takes into account this inequality.)*

$$lr_k^{(*)}(t_i) = \left(\frac{\theta}{D}\sum_{j=1}^{R}\sqrt{l_j b_j}\right)\sqrt{\frac{l_k}{b_k}}. \tag{7}$$

Proof. As mentioned previously, the target function is convex; thus, there must exist the minimal extreme point. In order to simplify the target function (i.e., (2)), we fix the task's execution time to be $T(\leq D)$, which also satisfies the problem's conditions. Then, the target function could be converted to

$$P(\boldsymbol{r}) = \frac{T}{R}\cdot\sum_{k=1}^{R} b_k r_k, \text{ where } T \leq D. \tag{8}$$

The corresponding Lagrangian function is shown below:

$$F_2(\boldsymbol{r}) = \frac{T}{R}\cdot\sum_{k=1}^{R} b_k r_k + \lambda\left(\theta\sum_{k=1}^{R}\frac{l_k}{r_k} - D\right). \tag{9}$$

Based on the Lagrangian multiplier method, $\frac{\partial F_2}{\partial r_k} = 0$ (where $k = 1, 2, \ldots, R$) constructs a set of necessary conditions for getting the optimal solution (i.e., (10) must hold, where λ is a constant)

$$\lambda\theta R/T = b_k r_k^2/l_k. \tag{10}$$

According to (10), we can easily get (11), $\forall j, k(1 \leq j \neq k \leq R)$

$$r_k^2 b_k/l_k = r_j^2 b_j/l_j. \tag{11}$$

That is, (12) is the sufficient and necessary condition of the optimal solution, s.t. a given deadline

$$r_1 : r_2 : \cdots r_R = \sqrt{l_1/b_1} : \sqrt{l_2/b_2} : \cdots : \sqrt{l_R/b_R}. \tag{12}$$

In order to save the resource utilized by the current task as much as possible, the optimal allocation should make $\sum_{i=1}^{R}\frac{l_i}{r_i}$ equal to D. In fact, for any resource allocation $\boldsymbol{r}(t_i)$ meeting (12) while $\sum_{i=1}^{R}\frac{l_i}{r_i} < D$, there must exist another solution with lower resource allocation $\boldsymbol{r}(t_i)'$ (i.e., $\boldsymbol{r}'(t_i) \preceq \boldsymbol{r}(t_i)$) such that it also satisfies (12). Hence, the task t_i's optimal resource allocation should make $\sum_{k=1}^{R}\frac{l_k}{r_k} = D$, then, by combining this equation, we can calculate the optimal resource vector to be allocated as (7). □

Remark. With unbounded resource availabilities, there will be no any constraint to the problem of minimizing the target function $P(\boldsymbol{r})$. Based on the above analysis, there are infinite number of optimal stationary points, whose sufficient and necessary conditions are (12). For a vivid illustration, we show the graph of a simple case in Fig. 2, where $\boldsymbol{b} = (1, 1)^T$ and $\boldsymbol{l} = (1, 1)^T$. From this figure, we can observe that there exist the minimal extreme points and their number is infinitive, along the line $\{r_1 = r_2$ and $P(\boldsymbol{r}) = 4\}$. This result is consistent with (12).

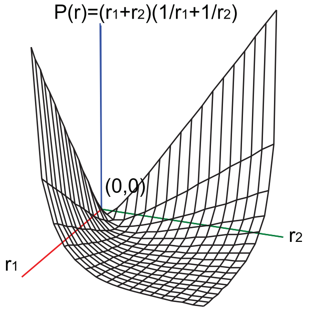


Fig. 2. The function graph of a simple case.

Formula (7) presents the resource share vector $\boldsymbol{r}^{(*)}$ gained by t_i such that its payment and the resource utilization can be both minimized within its execution deadline (i.e., (3)). Considering the constraint (4), $\boldsymbol{r}^{(*)}$ is right the optimal solution as long as $\boldsymbol{r}^{(*)} \preceq \boldsymbol{a}(p_s)$. However, if $\boldsymbol{r}^{(*)}$ does not fully satisfy the constraint (4) (i.e., $\exists k$: $r_k^{(*)} > a_k(p_s)$), $\boldsymbol{r}^{(*)}$ should not be a feasible solution. As one contribution, we propose an efficient algorithm (Algorithm 1) to determine the optimal solution subject to the constraint (4) with the provable time complexity $O(n \cdot R^2)$. □

Definition 1. *For any task t_i, based on a subset $\Gamma(\subseteq \Pi)$, CO-STEP(Γ, C) is defined as the procedure of computing the optimal solution of minimizing $P(\boldsymbol{r}_\Gamma(t_i))$ subject to the constraint (13) by using convex optimization (similar to the proof of Theorem 1), where C denotes a deadline and $\boldsymbol{r}_\Gamma(t_i)(= (r_1, r_2, \ldots, r_R)^T)$ denotes the resource shares gained by t_i on the execution dimension set Γ.*

$$\theta\sum_{i=1}^{R}\frac{l_i}{r_i} \leq C. \tag{13}$$

We devise Algorithm 1 for minimizing $P(\boldsymbol{r}(t_i))$ subject to the constraints (3) and (4), as shown below.

Algorithm 1. OPTIMAL ALLOCATION ALGORITHM

```
Input: $D(t_i)$; Output: execution node $p_s, \boldsymbol{r}^*(t_i)$
for (each candidate node $p_s$) do
    $\Gamma = \Pi, C = D(t_i), \boldsymbol{r}^* = \Phi$ (empty set);
    repeat
        $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s) = $ CO-STEP$(\Gamma, C)$; /* Compute optimal $r$ on $\Gamma$*/
        $\Omega = d_k | d_k \in \Gamma$ & $r_k^{(*)}(t_i, p_s) > a_k(p_s)\}$;
        /* select elements violating constraint (4)*/
        $\Gamma = \Gamma \backslash \Omega$; /*$\Gamma$ takes away $\Omega$*/
        $C = C - \theta\sum_{d_k\in\Omega}\frac{l_k}{a_k}$; /*Update $C$*/
        $\boldsymbol{r}^*(t_i, p_s) = \boldsymbol{r}^*(t_i, p_s) \cup \{r_k^* = a_k(p_s) | d_k \in \Omega \& a_k(p_s)$ is $d_k$'s upper bound$\}$;
    until $(\Omega = \Phi)$;
    $\boldsymbol{r}^*(t_i, p_s) = \boldsymbol{r}^*(t_i, p_s) \cup \boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$;
end for
Select the smallest $P(t_i)$ by traversing the candidate solution set;
Output the selected node $p_s$ and resource allocation $\boldsymbol{r}^*(t_i, p_s)$;
```

In this algorithm, line 4 executes CO-STEP(Γ, C) in order to find the optimal $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$, under the assumption without constraint (4). If $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$ completely satisfies the constraint (4) (i.e., $\Omega = \Phi$), then $\boldsymbol{r}_\Gamma^{(*)}(t_i, p_s)$ is the local optimal resource allocation for t_i to be run on p_s; otherwise, let the resource shares ($r_k(t_i)$, where $k = 1, 2, \ldots, R$) that violate the constraint (4) equal to its upper bound (i.e., $a_k(p_s)$) and take the corresponding execution dimensions (i.e., Ω) away from Γ, then, $C = C - \theta \sum_{d_k \in \Omega} \frac{l_k}{a_k}$ for the remaining dimensions. The process will go on until the computed optimal resource shares on the remaining dimensions satisfy the constraint (4). Since the time complexity of CO-STEP(Γ, C) is $O(|\Gamma|)$, the number of computation steps of line 2-10 in Algorithm 1 in the worst case is $\sum_{i=0}^{R-1}(R-i)$, thus the total time complexity of Algorithm $1 = O(n \cdot R^2)$.

Based on the Algorithm 1, it is obvious that the local optimal resource allocation for t_i to be executed on a specified node p_s is the most crucial part. In fact, the final outputted resource allocation solution of the whole algorithm will be globally optimal around the whole system as long as each local process on a specified node (line 2-10) can be proved as optimal resource allocation. Consequently, we will intensively discuss the local divisible-resource allocation by specifying a particular execution node, in the following text.

Theorem 2. ***Given*** *a submitted task t_i with its load vector $\boldsymbol{l}(t_i)$ and a deadline $D(t_i)$ and a particular node p_s with its resource price vector $\boldsymbol{b}(p_s)$,* ***then*** *the output after running the line 2-10 of Algorithm 1 (i.e., $\boldsymbol{r}^*(t_i, p_s)$ is optimal for minimizing t_i's payment (i.e., $P(\boldsymbol{r}(t_i))$), subject to the constraints (3) and (4).*

Main idea. We will prove that the $\boldsymbol{r}^*(t_i, p_s)$ satisfies KKT conditions (i.e., (6)).

Proof. At the beginning, the algorithm executes the CO-STEP$(\Pi, D(t_i))$ and the output is denoted $\boldsymbol{r}_\Pi^{(*)}$. Since $\boldsymbol{r}_\Pi^{(*)}$ is derived from Definition 1 and Theorem 1, $\boldsymbol{r}_\Pi^{(*)}$ must satisfy (12) and $\theta \sum_{i=1}^R \frac{l_i}{r_i} = D$, then if we let $\mu_k = 0$ for any k, there must exist an assignment such that all the conditions in (6) hold except for $r_k^{(*)} \leq a_k$. Accordingly, $\boldsymbol{r}^* = \boldsymbol{r}_\Pi^{(*)}$ as long as $r_k^{(*)} \leq a_k$ for all $r_k^{(*)}$s in $\boldsymbol{r}_\Pi^{(*)}$.

If $\boldsymbol{r}_\Pi^{(*)}$ cannot satisfy all the R inequalities ($r_k^* \leq a_k$, where $k = 1, 2, \ldots, R$), we need to further adjust the solution $\boldsymbol{r}_\Pi^{(*)}$ to find the one completely satisfying the condition (6). In Algorithm 1, at this moment, all the $r_k^{(*)}$s such that $r_k^{(*)} > a_k$ will be selected and set to a_k. Without loss of generality, assuming there are h_1 such resource shares and they are denoted as $r_1, r_2, \ldots, r_{h_1}$. Obviously, each selected r_k must satisfy $\mu_k \cdot (r_k - a_k) = 0$ because $r_k = a_k$. On the other hand, Algorithm 1 will continue to execute CO-STEP(Γ, C) on the rest $R - h_1$ dimensions, where $C = D(t_i) - \theta \sum_{k=1}^{h_1} \frac{l_k}{r_k}$. Likewise, all the $R - h_1$ new resource shares (each denoted by $r_k, k = h_1{+}1, \ldots, R$) must also satisfy

$$r_{h_1+1} : r_{h_1+2} : \cdots : r_R = \sqrt{\frac{l_{h_1+1}}{b_{h_1+1}}} : \sqrt{\frac{l_{h_1+2}}{b_{h_1+2}}} : \cdots : \sqrt{\frac{l_R}{b_R}},$$

and $\sum_{i=1}^R \frac{l_i}{r_i} = D$, thus if each of them meets the condition $r_k \leq a_k$, the $R - h_1$ new resource shares and the previously selected h_1 will together compose the solution satisfying the condition (6). If there are still $h_2 (0 < h_2 \leq R - h_1)$ new resource shares violating $r_k \leq a_k$ in this round, Algorithm 1 will continue the adjustment until the Hth round such that either all the $R - \sum_{i=1}^H h_i$ remaining resource shares can satisfy $r_k \leq a_k$ or there are no remaining resource dimensions in Γ. In the former case, we can easily verify that all the R resource shares satisfy the condition (6) simultaneously, composing an optimal solution; for the latter case, we could conclude that $\theta \sum_{i=1}^R \frac{l_i}{a_i} \geq D$, then there does not exist a feasible resource allocation to run the task within the specified deadline. In this situation, $\boldsymbol{r}^* = \boldsymbol{a} = (a_1, a_2, \ldots, a_R)^T$ will get the execution time closest to the deadline, and it will serve as the final solution. □

Although Algorithm 1 is proved optimal for minimizing the payment cost within user-defined deadline for his/her task, the deadline still may not be guaranteed due to two factors, either bounded available resources or inaccurate workload vector information about the task. We propose the following lemma, which provides a necessary and sufficient condition of guaranteeing the task's deadline given accurate prediction and relatively sufficient resources. In next section, we will discuss how to guarantee task's deadline when performing the Algorithm 1 with even inaccurate workload vector.

Lemma 1. *Given a task t_i's workload vector $\boldsymbol{l}(t_i) = (l_1, l_2, \ldots, l_R)^T$ and its deadline $D(t_i)$, and a candidate execution node p_s, then t_i can be executed within $D(t_i)$ if and only if (i.e., $\Leftrightarrow$) Inequality (14) holds:*

$$\sum_{j=1}^R \frac{l_j(t_i)}{a_j(p_s)} \leq D(t_i). \tag{14}$$

Proof. To prove $\Leftarrow$: If Inequality (14) holds, it is obvious there must exist a viable resource allocation $\boldsymbol{r}(t_i) (\preceq \boldsymbol{a}(p_s))$, such that $\sum_{j=1}^R \frac{l_j(t_i)}{a_j(p_s)} = D(t_i)$. Hence, t_i can be executed within $D(t_i)$.

To prove $\Rightarrow$: If t_i can be executed within $D(t_i)$, there must exist a viable resource allocation $\boldsymbol{r}(t_i)$ such that $\sum_{j=1}^R \frac{l_j}{r_j} \leq D(t_i)$ and $\boldsymbol{r}(t_i) \preceq \boldsymbol{a}(p_s)$. Assuming Inequality (14) does not hold at the moment, i.e., $\sum_{j=1}^R \frac{l_j(t_i)}{a_j(p_s)} > D(t_i)$, then, we could derive

$$\sum_{j=1}^R \frac{l_j(r_i)}{r_j(p_s)} < \sum_{j=1}^R \frac{l_j(t_i)}{a_j(p_s)}. \tag{15}$$

Accordingly, we can derive that there must exist a dimension, for example, d_k such that $r_k(t_i) \geq a(p_s)$, which contradicts to the previous assumption that $r(t_i)$ is a viable solution ($\boldsymbol{r}(t_i) \preceq \boldsymbol{a}(p_s)$). □

4 Optimality Analysis with Inaccurate Information

In this section, we focus on such a question: what is the final upper bound of task execution length as compared to its predefined deadline D, when running it using the resource vector allocated under Algorithm 1 with *inaccurately* predicted workload information?

4.1 Problem Description

Although Algorithm 1's output is proved optimal, such a result relies on a strong condition, i.e., accurate task's workload vector. That is, each user needs to precisely predict the execution property (i.e., workload ratio) for his/her task, before constructing the resource allocation with minimized payment for its execution under a user-specified deadline. In some cases, the execution property could be easily estimated accurately. For instance, we can decide the workload ratio between the data to be read/written from/to disk and those to be downloaded/uploaded via network by comparing their data sizes. In many other cases, however, the execution property cannot be accurately estimated, such as computation-intensive applications whose execution times highly depends on the CPU cycles to consume.

Definition 2. *Suppose a task t_i's real workload vector is $\boldsymbol{l}(t_i)$, while its workload vector used by our algorithm is $\boldsymbol{l}'(t_i)$ subject to the Inequality (16), where α and β are the lower bound and upper bound for the estimation ratio specified by user based on experiences or particular workload prediction methods such as [13], [14], [15]:*

$$\alpha \leq \frac{l'_k(t_i)}{l_k(t_i)} \leq \beta, k = 1, 2, \ldots R. \tag{16}$$

To illustrate the above definition, an example is given. Assuming the task t_i's real workload ratios range in [0.125, 1], and the workload vector $\boldsymbol{l}'(t_i)$ used by Algorithm 1 will be set based on the task's historical execution records. Suppose each element $l'_k(t_i)(k = 1, 2, \ldots R)$ will be set to 0.25 if the corresponding true workload fluctuates in [0.125, 0.5] and set to 0.75 if the true workload ranges within (0.5, 1]. Then, we could get Inequality (17) below, where $\alpha = \frac{0.125}{0.25} = 0.5$ and $\beta = \frac{0.5}{0.25} = 2$:

$$0.5 \leq \frac{l'_k(t_i)}{l_k(t_i)} \leq 2, k = 1, 2, \ldots R. \tag{17}$$

Using the inaccurate prediction $\boldsymbol{l}'(t_i)$ to perform the Algorithm 1, it is obvious that t_i's real execution time may surpass the expected execution deadline $D(t_i)$. Hence, one question is what the worst performance will get when using $\boldsymbol{l}'(t_i)$ instead of $\boldsymbol{l}(t_i)$, compared to the expected deadline $D(t_i)$.

4.2 Deadline Extension Ratio (DER) with Skewed Estimation of Execution Property

For simplicity of description, we denote $\boldsymbol{r}^*_E(= (r^*_{E1}, r^*_{E2}, \ldots, r^*_{ER})^T)$ and $T^*_E(= \theta\sum_{k=1}^{R} \frac{l_k}{r^*_{Ek}})$ as the output of Algorithm 1 with the skewed workload prediction and the corresponding execution time, respectively (E here implies "Estimation with error"). Similarly, we denote $\boldsymbol{r}^*_I(= (r^*_{I1}, r^*_{I2}, \ldots, r^*_{IR})^T)$ and $T^*_I(= D = \theta\sum_{k=1}^{R} \frac{l_k}{r^*_{Ik}})$ as the output with real workload vector and the corresponding execution time, respectively (I here indicates "Ideal case"). Hence, our objective is to determine the upper bound of $\frac{T^*_E}{T^*_I}$, a.k.a., deadline extension ratio.

We partition the situation that Algorithm 1 would face to two categories, where $\boldsymbol{r}^{(*)}_E$ refers to the optimal resource allocation with the constraint (4) (unlike the notation $\boldsymbol{r}^*_E$):

- $\boldsymbol{r}^*_E(t_i) = \boldsymbol{r}^{(*)}_E(t_i)$.
- $\boldsymbol{r}^*_E(t_i) \neq \boldsymbol{r}^{(*)}_E(t_i)$.

The first situation indicates that in terms of the skewed estimation of workload ratios, all the resource shares calculated by the initial CO-STEP in Algorithm 1 are always no greater than the corresponding capacities. That is, it is equal to the situation with the assumption that Inequality (18) holds:

$$\boldsymbol{r}^{(*)}_E(t_i) \preceq \boldsymbol{a}(p_s). \tag{18}$$

In contrast, the second one means that the initial CO-STEP cannot fulfill the above condition, and the optimal allocation cannot be found unless a few more adjustment steps (line 5-8 of Algorithm 1).

As follows, we will first derive task t_i's execution time upper bound for the first category (i.e., Theorem 3), and then discuss the upper bound (i.e., Theorem 4) for the more generic case including the second category.

Theorem 3. ***Given** a submitted task t_i with a predefined deadline $D(t_i)$, a candidate execution node p_s with unbounded resource capacity and a resource price vector (denoted $\boldsymbol{b}(p_s)$), and a skewed workload vector $\boldsymbol{l}'(t_i)$ subject to Inequality (16), **then** the bound of execution time must satisfy Inequality (19), under the resource allocation $\boldsymbol{r}^{(*)}_E$:*

$$\frac{1}{\beta} \cdot D(t_i) \leq T^{(*)}_E(t_i) \leq \frac{1}{\alpha} \cdot D(t_i). \tag{19}$$

Proof.

$$T^{(*)}_E = \theta \sum_{k=1}^{R} \frac{l_k}{r^{(*)}_{Ek}} = \theta \sum_{k=1}^{R} \frac{l_k}{(\frac{\theta}{D}\sum_{i=1}^{R}\sqrt{l'_i b_i})\sqrt{\frac{l'_k}{b_k}}}$$

$$= \left(\frac{D}{\sum_{i=1}^{R}\sqrt{l'_i b_i}}\right) \sum_{k=1}^{R} \frac{l_k\sqrt{b_k}}{\sqrt{l'_k}} \leq \frac{D}{\sqrt{\alpha}} \cdot \frac{1}{\sqrt{\alpha}} \cdot \frac{\sum_{k=1}^{R}\sqrt{l_k b_k}}{\sum_{i=1}^{R}\sqrt{l_i b_i}} = \frac{D}{\alpha}.$$

□

The key of the above proof is based on the Inequality (16). Similarly, According to Inequality (16), we can also derive $T^{(*)}_E \geq \frac{D}{\beta}$.

Accordingly, Inequality (19) holds. It is easy to see that Inequality (19)'s bound is tight. Considering such a case: $\forall k, l'_k(t_i) = \alpha l_k(t_i)$, then $T^{(*)}_E$ will be equal to $\frac{D}{\alpha}$

Theorem 4. ***Given** a submitted task t_i with a predefined deadline $D(t_i)$, a candidate execution node p_s with a limited available resource vector ($\boldsymbol{a}(p_s)$) and price vector $\boldsymbol{b}(p_s)$, and a skewed workload vector $\boldsymbol{l}'(t_i)$ subject to Inequality (16), if Inequality (14) holds, **then** under the resource allocation $\boldsymbol{r}^*_E$, the bound of execution time must conform to*

$$\frac{1}{\beta} \cdot D(t_i) \leq T^*_E(t_i) \leq \frac{1}{\alpha} \cdot D(t_i). \tag{20}$$

Proof. Without loss of generality, we denote Ω to be the set of resource dimensions accumulated by Line 5 of Algorithm 1, and the corresponding dimensions' indexes are 1, 2,$\ldots, |\Omega|$. That is, $r^*_1 = a_1, r^*_2 = a_2, \ldots, r^*_{|\Omega|} = a_{|\Omega|}$, while $r_{|\Omega|+1} < a_{|\Omega|+1}, \ldots, r_R < a_R$. Hence, we can get the following equation:

$$T_E^* = \theta\left(\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \sum_{i=|\Omega|+1}^{R}\frac{l_i}{r_{Ei}^*}\right). \tag{21}$$

We could further prove the Inequality (20) as follows:

$$\begin{aligned}
T_E^* &= \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \theta\sum_{k=|\Omega|+1}^{R}\frac{l_k}{r_{Ek}^*}\\
&= \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \theta\sum_{k=|\Omega|+1}^{R}\frac{l_k\sqrt{\frac{b_k}{l'_k}}}{\frac{\theta}{D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_i}{a_i}}\sum_{i=|\Omega|+1}^{R}\sqrt{l'_i b_i}}\\
&= \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \sum_{k=|\Omega|+1}^{R}\frac{\left(D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_i}{a_i}\right)\cdot l_k\sqrt{\frac{b_k}{l'_k}}}{\sum_{i=|\Omega|+1}^{R}\sqrt{l'_i b_i}}\\
&\le \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \left(D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_i}{a_i}\right)\sum_{k=|\Omega|+1}^{R}\frac{l_k\sqrt{\frac{b_k}{\alpha l_k}}}{\sum_{i=|\Omega|+1}^{R}\sqrt{\alpha l_i b_i}}\\
&= \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} + \frac{1}{\alpha}\left(D-\theta\sum_{i=1}^{|\Omega|}\frac{l'_i}{a_i}\right)\\
&\le \frac{D}{\alpha} + \theta\sum_{i=1}^{|\Omega|}\frac{l_i}{a_i} - \frac{\theta}{\alpha}\sum_{i=1}^{|\Omega|}\frac{\alpha l_i}{a_i} = \frac{D}{\alpha}.
\end{aligned}$$

□

The key of the above proof is based on the Inequality (16). Similarly, According to Inequality (16), we can also derive $T_E^* \ge \frac{D}{\beta}$. Hence, Inequality (20) holds.

When Ω is empty, the lower bound and upper bound of Inequality (20) can be reached as the upper bound and lower bound of Inequality (16) are met, respectively.

Remark. Let us review the Theorem 4 and discuss its significance. Inequality (20) implies that task t_i's execution time based on the optimal resource allocation of Algorithm 1 under inaccurate workload ratios has an upper bound, which is only determined by the lower bound of the inaccurate ratio α. In principle, by leveraging this theoretical result, we can always provide the strict guarantee for user-preset deadline even with the wrong prediction of task's property, as long as there are relatively sufficient resources. In fact, what we need to do is just setting a stricter deadline D' according to (22) and preforming the Algorithm 1 based on D' instead of D. Then, the user task's deadline will be strictly limited under its expected value D even though the workload ratio information is inaccurate (s.t. Inequality (16)). On the other hand, existing workload prediction work can be used to support how to determine the value of α. For example, the polynomial regression method [16] can bound the prediction error in 10 percent (i.e., $\alpha = 0.1$):

$$D' = \alpha \cdot D. \tag{22}$$

□

5 Performance Evaluation

5.1 Experimental Setting

We implement a web service-based prototype that can compute a set of combined matrix operations. Each matrix operation is called by some user task through a web service API and each task is executed in a VM container. Our algorithm is evaluated on such a real cluster environment. There are 10 physical nodes in the cluster, each owning 2 quad-core Xeon CPU E5540 (i.e., eight processors per node) and 16 G of memory. There are 60 VM-images (centos 5.2) kept by Network File System (NFS), so 60 VM-instances will be created at the bootstrap before our experiment. XEN 3.1 [17] serves as the hypervisor/VMM on each node and dynamically allocates various CPU speeds (or capabilities) to the VM-instances at runtime using credit scheduler.

TABLE 1
Workload of Typical Matrix Operations (Seconds/Core)

Matrix-Matrix-Multiply				QR-Decom. Solving				Matrix-Power		
M	N	P	load	M	N	P	load	M	m	load
500	500	500	1.08	500	500	500	1.39	500	10	3.65
1000	1000	1000	13.7	1000	1000	1000	6.1	500	20	4.2
1500	1500	1000	31	1500	1500	1000	10.5	1000	10	55
2000	1500	1000	40.2	2000	1500	1000	14.1	1000	20	67.2
2000	2000	2000	118	2000	2000	2000	27.9	2000	10	457.7
2500	2500	2500	242	2500	2500	2500	51	2500	20	1206

Users can submit their computation request by editing their mathematical formulas. In our experiment, we make use of *ParallelColt* [18] to perform math computations, each consisting of a partially ordered set of operations. ParallelColt [18] is such a library that can effectively calculate complex matrix-operations like matrix matrix multiply, in parallel via multiple threads. Here is an example computation request, which is submitted as *Solve* $((A_{m\times n}\cdot A_{n\times m})^k, B_{m\times m})$. Such a computation task can be split into three steps (or subtasks) of different matrix operations: 1) matrix multiplication: $C_{m\times m} = A_{m\times n}\cdot A_{n\times m}$; 2) matrix-power: $D_{m\times m} = C^k_{m\times m}$; 3) Least squares solution of $D\cdot X = B$ based on QR-Decomposition: *Solve*$(D_{m\times m}, B_{m\times m})$. In our benchmark, we simulate a large number of user requests, each of which is composed of 3-15 subtasks. Each subtask is constructed of three typical matrix operations (i.e., matrix multiply, matrix power, and QR-matrix solving(least square)) with various parameters assigned. That is, each request contains many subtasks that are randomly selected from the above three types. We evaluate our algorithm under different competitive situation with different number (1-40) of tasks submitted simultaneously; thus, there are 40 cases for each experiment which has 820 submitted tasks in total as observed.

In our system, each matrix-operation's workload is estimated based on the historical tracing records. The workload prediction formula is shown in (23), where j denotes the number of processors and $T(op_i, j)$ indicates the execution time of running the matrix operation (denoted op_i) on j cores:

$$l_i = \frac{1}{8}\sum_{j=1}^{8}(j\cdot T(op_i, j)). \tag{23}$$

Each user request (denoted as task t_i) is assigned with a deadline, which is a random value in $[\frac{1}{8}\cdot T_1(t_i), T_1(t_i)]$, where $T_1(t_i)$ means the estimated execution time when running the task t_i on a particular core. Based on our experiment, the three matrix operations on one core will cost from 1 second to 1,206 seconds as shown in Table 1, which implies a quite heterogenous nature. In Table 1,

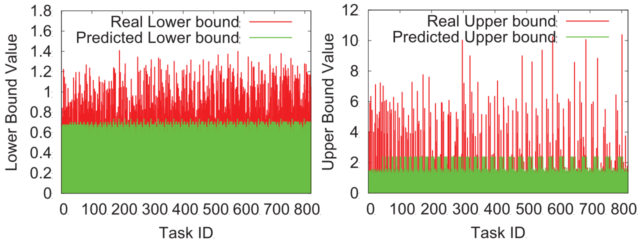


Fig. 3. Workload prediction. (a) Lower bound. (b) Upper bound.

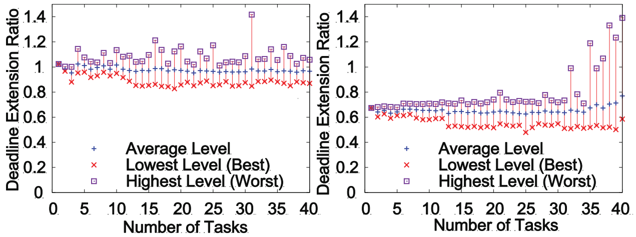


Fig. 4. Deadline extension ratio. (a) $D' = D$. (b) $D' = \alpha \cdot D$.

M, N, and P refers to the matrix scale in the matrix-matrix-multiply and QR-Decomposition Solving, and m indicates the value of exponent in the matrix-power computation. Users' prices of running the three individual matrix-operations are set to 1, 2, and 3, respectively.

5.2 Experimental Results

We first present the prediction effect over the historical records of the three matrix operations (as shown in Fig. 3), in that the approximation ratio of our optimal algorithm is based on the inaccuracy of the workload predicted, according to the analysis in Section 4. From this figure, we can clearly observe that the prediction method we used can make sure that the lower bound of the workload predicted (i.e., α's value will be set close to 0.7, where α is defined in Definition 2 and used in Theorem 3 and 4) is always lower than the real workload that is calculated after its execution.

We evaluate our designed algorithm with and without the prediction-error-tolerant support. That is, the system will test the Algorithm 1 with the tuned stricter deadline (D') or the original one (D). We use *Deadline Extension Ratio* (defined as the ratio of task's final execution time to its deadline) to evaluate the statistical task execution lengths compared to their expected deadlines. We run 40 separate cases each with different number (1-40) of tasks, and show the lowest/average/highest level of DER for each case.

We first show the experimental result by using the original deadline D (i.e., $D' = D$) in the algorithm. From Fig. 4a, we see that the tasks' execution times cannot be always guaranteed to be executed within their deadlines in the worst case, no matter how many tasks (1-40) are submitted. Specifically, even though the system availability is relatively high (e.g., there are only several tasks submitted), the average value of deadline extension ratio is nearly to 1 and its highest value in the worst case is up to 1.2. This is mainly due to the inaccurate workload prediction with about 30 percent margin of errors as shown in Fig. 3. In comparison, Fig. 4b shows the deadline extension ratio when the deadline D' is set to a stricter deadline ($\alpha \cdot D$). When the number (denoted by m) of tasks submitted scales up to 30, all tasks' execution times can be kept nearly to about only 0.7 times as high as their preset deadlines (D) at the worst situation (i.e., the highest level shown in the figure). With further increasing number of the submitted tasks, tasks' execution times cannot be always guaranteed because of the limited resource capacities (or the higher level of competitions on resources), but the mean level is still kept remarkably lower than 1, which means that most of the tasks can still meet the QoS (i.e., large majority can be finished before deadlines). Note that there are only 10 physical machines in our experiment but much more than 10 tasks can be processed with guaranteed deadlines, which indicates a remarkably high level on service consolidation. This also implies a great potential in improving resource utilization by taking advantage of VM-multiplexing feature.

Fig. 5 presents the distribution of the deadline extension ratio, in a competitive situation where there are 40 tasks submitted. We observe that the stricter deadline-based algorithm can more effectively limit the majority tasks' execution times to about 0.7 times as high as the user-specified deadlines (i.e., the original ones), but it may suffer from higher DER at the worst case. In comparison, the majority of tasks (about 52.5 percent) under the original-deadline-based algorithm are completed within 0.95- 1.0 times of their deadlines, which still conforms to the deadline-guaranteed requirement; there are about 20 percent of tasks that would violate deadlines, most of which (17.5 percent) are still finished within 1.05 times of deadlines.

Finally, we evaluate the fairness of task processing in the two cases, confirming the stability. Based on Jain's work [19], fairness index (higher value means higher fairness) is defined as (24) whose value ranges in [0, 1], where x_i refers to the DER of task t_i:

$$F(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}. \tag{24}$$

We present the experimental results about the fairness index of the DER in Fig. 6. As observed, the fairness index is always kept over 0.99 for both cases under the relatively uncompetitive situation (e.g., $m \leq 30$), and still kept about 0.95 in the case with higher competition (i.e., when $m > 30$). Recall that there are only 10 physical machines used for resource provisioning in our experiment, which implies our solution's allocation effect is confirmed to be quite stable to any task's execution under such a dense server consolidation. In addition, the main reason for the degradation of the fairness of DER in the competitive situation is that the tasks

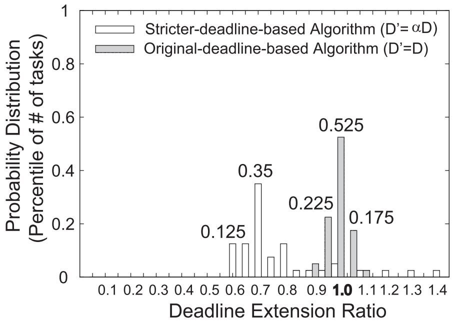


Fig. 5. Distribution of DER (the number of tasks).

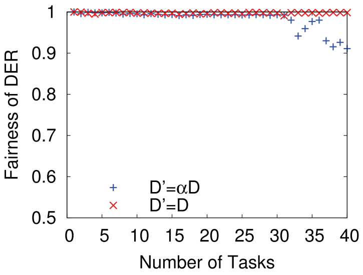


Fig. 6. Fairness index of DER.

with higher priorities or the ones arriving earlier would be treated with higher service level in our experiment, which would definitely impact other lower priority tasks' execution in the short-supply situation. In fact, guaranteeing the higher priority tasks's QoS by sacrificing lower priority tasks' benefit may also be considered a fairer treatment in many scenarios. Hence, for different applications, we can easily maximize the fairness level among all tasks by assigning the adaptive values for α, which will be further studied in our future work.

6 Related Work

Traditional job scheduling [20] is often formulated as a kind of combinatorial optimization problem (or queue-based multiprocessor scheduling problem [21], [22], [12]), due to the nonguaranteed performance isolation for multiple tasks running on the same machines. That is, most of the existing deadline-driven task scheduling solutions (from single cluster environment confined in LAN [23], [24] to the Grid computing environment suitable for WAN [25], [26]) are also strictly subject to the queuing model under which a single machine's multiple resources cannot be further split to smaller fractions at will. This will eventually cause the raw-grained resource allocation, relatively low resource utilization and suboptimal task execution efficiency.

With the VM resource isolation technology being mature recently, it is viable to design more efficient resource allocation due to the fledged performance isolation among VMs running on the same machines. Meng et al. [27] proposed a VM multiplexing-based resource allocation approach, which can successfully analyze the compatibility of any two different VMs (each with an application running atop it) on the same physical machines, and reschedule the combination of the VMs to improve the overall performance. However, it cannot guarantee high compatibility among more than two VMs on the same machine. Q-Clouds [28] is another well-known system which can realize high consolidation of multiple VM-hosted applications, focusing on how to prevent inevitable performance interference among VMs from degrading user's QoS or enhancing corresponding users' payment unexpectedly.

Compared to the above existing works about VM-multiplexing resource allocation, our work aims to not only confine tasks' execution to be within their deadlines, but also minimize the payments for their users. This work will definitely benefit and motivate many cloud users or service providers, who wish to minimize the infrastructure cost with the guaranteed QoS, actually already endeavored by many researchers. Wu et al. [29], for example, proposed a SLA-based resource allocation method, which is compatible to the heterogeneity of infrastructure and adaptable to the dynamic change of customer requests. It can maximize the profit of SaaS providers by minimizing the humber of SLA violations and the cost by reusing VMs. Chaisiri et al. [30] also aim to minimize the provisioning cost incurred to users by taking into account stochastic programming, robust optimization, and sample-average approximation together. Mao et al. [31], [32] present a cloud autoscaling mechanism to automatically scale computing instances based on workload information and performance desire, also aiming to guarantee task's deadline with less payment. In comparison, our approach can be fundamentally proved optimal via the convex optimization theory, which we believe is a huge step forward especially from the perspective of theoretical analysis.

Most of the existing theoretical research on cloud computing [33], [34], [35] mainly focused on the relatively ideal scenarios by assuming tasks' workloads can be accurately predicted, simplifying the resource allocation problem. For example, Weinman [33] analyzed the penalty functions working in the workload aggregation and relative statistical effects, given a set of fixed task workloads to be used, while Petrucci et al. [34] proposed an optimization VM-based model to minimize the power and management cost by assuming that application's consumption can be predicted precisely by monitoring system. Unlike these works, we theoretically analyze the upper bound of task's execution time compared to its deadline and that of user-specified payment to the precise-prediction-based result. By taking advantage of the derived bounds and approximation ratio, we can more effectively guarantee user tasks' QoS in terms of their demands. To the best of our knowledge, this is the first attempt to study how to minimize the payment cost in the cloud system, which can also tolerate the prediction errors of tasks' properties.

7 Conclusion and Future Work

In this paper, we propose a novel resource allocation algorithm for cloud system that supports VM-multiplexing technology, aiming to minimize user's payment on his/her task and also endeavor to guarantee its execution deadline meanwhile. We can prove that the output of our algorithm is optimal based on the KKT condition, which means any other solutions would definitely cause larger payment cost. In addition, we analyze the approximation ratio for the expanded execution time generated by our algorithm to the user-expected deadline, under the possibly inaccurate task property prediction. When the resources provisioned are relatively sufficient, we can guarantee task's execution time always within its deadline even under the wrong prediction about task's workload characteristic. In the future, we plan to integrate our algorithms with stricter/original deadlines into some excellent management tools like OpenNebula, for maximizing the system-wide performance. Some queuing policies like earliest deadline first (EDF) will be studied to further reduce user payment especially in the short supply situation. More complex scheduling constraints like the compatibility and security issue will also be taken into account.

ACKNOWLEDGMENTS

This research was supported by a Hong Kong RGC grant HKU 7179/09E and a Hong Kong UGC Special Equipment Grant (SEG HKU09).

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the Clouds: A Berkeley View of Cloud Computing,” Technical Report UCB/EECS-2009-28, EECS Dept., Univ. California, Berkeley, Feb. 2009.

[2] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A Break in the Clouds: Towards a Cloud Definition,” *SIGCOMM Computer Comm. Rev.,* vol. 39, no. 1, pp. 50-55, 2009.

[3] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, Nov. 2003.

[4] J.E. Smith and R. Nair, *Virtual Machines: Versatile Platforms For Systems and Processes.* Morgan Kaufmann, 2005.

[5] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing Performance Isolation across Virtual Machines in Xen,” *Proc. ACM/IFIP/USENIX Int'l Conf. Middleware (Middleware '06),* pp. 342-362, 2006.

[6] J.N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, “Quantifying the Performance Isolation Properties of Virtualization Systems,” *Proc. Workshop Experimental Computer Science (ExpCS '07),* 2007.

[7] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, 2012.

[8] D. Milojicic, I.M. Llorente, and R.S. Montero, “Opennebula: A Cloud Management Tool,” *IEEE Internet Computing,* vol. 15, no. 2, pp. 11-14, Mar./Apr. 2011.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge Univ. Press, 2009.

[10] E. Imamagic, B. Radic, and D. Dobrenic, “An Approach to Grid Scheduling by Using Condor-G Matchmaking Mechanism,” *Proc. 28th Int'l Conf. Information Technology Interfaces,* pp. 625-632, 2006.

[11] B. Sharma, V. Chudnovsky, J.L. Hellerstein, R. Rifaat, and C.R. Das, “Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters,” *Proc. Second ACM Symp. Cloud Computing (SOCC '11),* pp. 3:1-3:14, 2011.

[12] H. Khazaei, J.V. Misic, and V.B. Misic, “Modelling of Cloud Computing Centers Using m/g/m Queues,” *Proc. Int'l Conf. Distributed Computing Systems Workshops (ICDCS),* pp. 87-92, 2011.

[13] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, “Adaptive Workload Prediction of Grid Performance in Confidence Windows,” *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 7, pp. 925-938, July 2010.

[14] S. Di, D. Kondo, and W. Cirne, “Characterization and Comparison of Cloud versus Grid Workloads,” *Proc. 14th Int'l Conf. Cluster Computing,* pp. 230-238, 2012.

[15] Q. Zhang, J.L. Hellerstein, and R. Boutaba, “Characterizing Task Usage Shapes in Google's Compute Clusters,” *Proc. Large-Scale Distributed Systems and Middleware Workshop (LADIS '11),* 2011.

[16] L. Huang, J. Jia, B. Yu, B.G. Chun, P. Maniatis, and M. Naik, “Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression,” *Proc. 24th Conf. Neural Information Processing Systems (NIPS '10),* pp. 1-9, 2010.

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization,” *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* pp. 164-177, 2003.

[18] P. Wendykier and J.G. Nagy, “Parallel Colt: A High-Performance Java Library for Scientific Computing and Image Processing,” *ACM Trans. Math. Software,* vol. 37, pp. 31:1-31:22, Sept. 2010.

[19] R.K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling.* John Wiley & Sons, Apr. 1991.

[20] C. Jiang, C. Wang, X. Liu, and Y. Zhao, “A Survey of Job Scheduling in Grids,” *Proc. Joint Ninth Asia-Pacific Web and Eighth Int'l Conf. Web-Age Information Management Conf. Advances in Data and Web Management (APWeb/WAIM '07),* pp. 419-427, 2007.

[21] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems,* ftp://ftp.nada.kth.se/Theory/Viggo-Kann/compendium.pdf, 2012.

[22] O. Sinnen, *Task Scheduling for Parallel Systems.* Wiley-Interscience, May 2007.

[23] K. Ramamritham, J.A. Stankovic, and W. Zhao, “Distributed Scheduling of Tasks with Deadlines and Resource Requirements,” *IEEE Trans. Computers,* vol. 38, no. 8, pp. 1110-1123, Aug. 1989.

[24] M.C. McElvany and P.D. Stotts, “Guaranteed Task Deadlines for Fault-Tolerant Workloads with Conditional Branches,” *Real-Time Systems,* vol. 3, no. 3, pp. 275-305, 1991.

[25] L. Zhao, Y. Ren, and K. Sakurai, “A Resource Minimizing Scheduling Algorithm with Ensuring the Deadline and Reliability in Heterogeneous Systems,” *Proc. 25th IEEE Int'l Conf. Advanced Information Networking and Applications (AINA '11),* pp. 275-282, 2011.

[26] W. Chen, A. Fekete, and Y.C. Lee, “Exploiting Deadline Flexibility in Grid Workflow Rescheduling,” *Proc. 11th IEEE/ACM Int'l Conf. Grid Computing (Grid '10),* pp. 105-112, 2010.

[27] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, “Efficient Resource Provisioning in Compute Clouds via VM Multiplexing,” *Proc. Seventh Int'l Conf. Autonomic Computing (ICAC '10),* pp. 11-20, 2010.

[28] R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-Clouds : Managing Performance Interference Effects for Qos-Aware Clouds,” *Proc. European Conf. Computer Systems (EuroSys '10),* pp. 237-250, 2010.

[29] L. Wu, S.K. Garg, and R. Buyya, “SLA-Based Resource Allocation for Software as a Service Provider (SAAS) in Cloud Computing Environments,” *Proc. 11th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGRID '11),* pp. 195-204, 2011.

[30] S. Chaisiri, R. Kaewpuang, B.-S. Lee, and D. Niyato, “Cost Minimization for Provisioning Virtual Servers in Amazon Elastic Compute Cloud,” *Proc. 19th Ann. IEEE/ACM Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems (MASCOTS '11),* pp. 85-95, 2011.

[31] M. Mao, J. Li, and M. Humphrey, “Cloud Auto-Scaling with Deadline and Budget Constraints,” *Proc. 11th IEEE/ACM Int'l Conf. Grid Computing (Grid '10),* pp. 41-48, 2010.

[32] M. Mao and M. Humphrey, “Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows,” *Proc. Int'l Conf. High Performance Computing, Networking, Storage & Analysis (SC '11),* pp. 49:1-49:12, 2011.

[33] J. Weinman, “Smooth Operator: The Value of Demand Aggregation,” http://joeweinman.com/Resources/Joe_Weinman_Smooth_Operator _Demand_Aggregation.pdf, 2011.

[34] V. Petrucci, O. Loques, and D. Mossé, “A Dynamic Optimization Model for Power and Performance Management of Virtualized Clusters,” *Proc. First Int'l Conf. Energy-Efficient Computing and Networking (e-Energy '10),* pp. 225-233, 2010.

[35] F. Chang, J. Ren, and R. Viswanathan, “Optimal Resource Allocation in Clouds,” *Proc. IEEE Int'l Conf. Cloud Computing,* pp. 418-425, 2010.

Sheng Di received the MPhil degree from Huazhong University of Science and Technology in 2007 and the PhD degree from The University of Hong Kong in 2011. He is currently a postdoctoral researcher at INRIA. His research interest involves optimization of distributed resource allocation especially in P2P systems and large-scale cloud computing platforms. His background is mainly on the fundamental theoretical analysis and practical system implementation. He is a member of the IEEE.

Cho-Li Wang received the PhD degree from the University of Southern California in 1995. His research interests include multicore computing, software systems for Cluster and Grid computing, and virtualization techniques for cloud computing. He serves on the editorial boards of several international journals, including *IEEE Transactions on Computers*(2006-2010), *Journal of Information Science and Engineering*, and *Multiagent and Grid Systems*. He is the regional coordinator (Hong Kong) of IEEE Technical Committee on Scalable Computing (TCSC). He is a member of the IEEE.