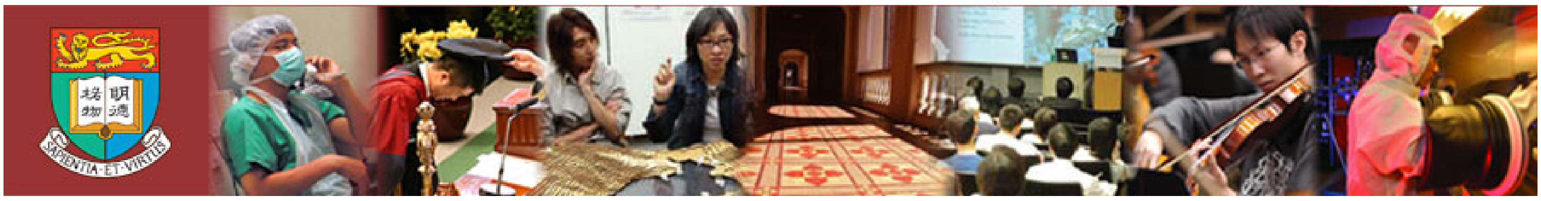

Minimization of Cloud Task Execution Length with Workload Prediction Errors

Sheng Di[1], Cho-Li Wang[2]
[1]INRIA, France
[2]The University of Hong Kong, Hong Kong
{sdi, clwang}@cs.hku.hk

***Abstract*—In cloud systems, it is non-trivial to optimize task's execution performance under user's affordable budget, especially with possible workload prediction errors. Based on an optimal algorithm that can minimize cloud task's execution length with predicted workload and budget, we theoretically derive the upper bound of the task execution length by taking into account the possible workload prediction errors. With such a state-of-the-art bound, the worst-case performance of a task execution with a certain workload prediction errors is predictable. On the other hand, we build a close-to-practice cloud prototype over a real cluster environment deployed with 56 virtual machines, and evaluate our solution with different resource contention degrees. Experiments show that task execution lengths under our solution with estimates of worst-case performance are close to their theoretical ideal values, in both non-competitive situation with adequate resources and the competitive situation with a certain limited available resources. We also observe a fair treatment on the resource allocation among all tasks.**

I. Introduction

Cloud computing has emerged as a compelling paradigm for the deployment of ease-of-use virtual environment on the Internet. One of typical features of Clouds is its pool of instantly accessible virtualized resources that can be dynamically customized, while optimizing the resource utilization.

Traditional task scheduling adopted in distributed systems like Grids assumes discrete resource usage model [1], [2], [3]. The processing ability assigned to a task cannot be customized by users elastically. Such an indivisible resource consumption model with discrete computation units results in a non-trivial problem like binary Integer programming problem. For example, given a CPU-intensive task that was assigned with a payment budget and a particular per-time-unit price for resource consumption, how many cores would lead to the minimization of the task execution length (or execution time) is an Integer Programming problem. That is, much of CPU rate may not be fully utilized in this case.

With virtual machine (VM) resource isolation technology [4], [5], [6], [7], [8], [9], the computational resources could be partitioned and reassembled to meet users' specific demands, significantly improving resource utilization. The key contribution of this paper is improving the algorithm to suit practical issues like the worst-case performance with possible workload prediction errors.

- In our work, the divisible-resource allocation problem is modeled as a convex optimization problem [10], such that we can find the optimal solution quickly from the perspective of resource allocation. We also take into account in our model the possible execution costs like the extra time in loading VM images. Based on the problem formulation, we can rigorously prove that our proposed algorithm is an optimal solution to minimize cloud task execution length with respect to resource allocation.
- We further study the upper bound of task execution length in the situation that task workloads are predicted with a margin of errors, which is more in the line with reality. Although the existing workload prediction method like *multi-variate polynomial regression* [11] have been very effective, it still suffers inevitable margin of prediction errors like 10%. The inevitable workload prediction errors may significantly affect task execution in reality. In this paper, we theoretically derive the performance bound of task length for the above proposed optimal algorithm in terms of a margin of prediction errors, as compared to the theoretically ideal task length with hypothetically precise workload information. This is fairly valuable/useful in that users are able to predict the worst performance in advance and one can tune the resource allocation to meet higher Quality of Service (QoS) based on our derived bound of task execution time.
- We implement and evaluate the algorithm over a real cluster environment deployed with 56 real VM instances. By leveraging Xen's credit scheduler [12] and Linux network traffic controller (TC) [13], we evaluate our algorithm based on divisible CPU rates, on-demand network and disk bandwidth. The cloud services in our experiment corresponds to different types of executions like computation incentive or I/O incentive applications. Experiments confirm that our solution by considering the workload prediction errors is able to effectively restrict task's execution length even in a competitive situation with a certain limited resource capacities.

The remainder of the paper is organized as follows. In Section II, we formulate the research as a convex-optimization problem that aims to minimize task execution length with divisible resource allocation subject to some constraints like user payment budgets. In Section III, we describe the optimal solution and rigourously prove its optimality in minimizing task execution length. In Section IV, we derive the upper bound of task execution length for the situation with erroneous workload prediction, as against to the ideal results with the hypothetically precise prediction. We present the experimental results generated over a real cluster environment in Section V. We discuss the related works in Section VI and conclude with a vision of the future work in Section VII.

II. Problem Formulation

Our architecture follows a popular data center model (or server/client model) to process user requests. The cloud server is responsible for collecting dynamic availability states of managed nodes and customizing virtual machines for users based on their elastic demands. The procedure in executing a task can be split into three steps: (1) Select a physical node upon task arrival. (2) Each task will be executed in an individual VM instance, whose resources are customized on demand by virtual machine monitor (VMM), a.k.a., hypervisor. (3) Send computational results to users.

Different task executions are likely of largely different patterns due to the fact that they need multiple types of resources (or execution dimensions). For example, one task execution may be split into multiple steps, each calling for a different CPU rate or I/O bandwidth on demand (e.g., CPU, IO, CPU, $\cdots$).

We assume there are n resource nodes (denoted by p_i, where $1 \leq i \leq n$). For any particular task requiring R execution dimensions (e.g., R execution steps each with different types of resources), we denote the complete set of execution dimensions by Π and denote p_i's capacity vector on the multiple dimensions by $\boldsymbol{c}(p_i)$=$(c_1(p_i), c_2(p_i), \cdots, c_R(p_i))^T$ (In the paper, we use bold-face to indicate a *vector*).

A task assigned to node p_i is denoted by t_{ij}, where $1 \leq j \leq m_i$, and m_i refers to the number of tasks assigned to p_i. The *workload vector* of the task t_{ij} to process in multiple dimensions is denoted by $\boldsymbol{l}(t_{ij})$=$(l_1(t_{ij}),l_2(t_{ij}),\cdots, l_R(t_{ij}))^T$. Hence, t_{ij} needs a resource vector to complete its workloads, and we denote such a vector as $\boldsymbol{r}(t_{ij})$=$(r_1(t_{ij}),r_2(t_{ij}),\cdots,r_R(t_{ij}))^T$, where $r_k(t_{ij})$ (k=1,2,$\cdots$,R) refers to the resource fraction split from t_{ij}'s assigned node p_i. Node p_i's resource availability vector on multiple dimensions (denoted $\boldsymbol{a}(p_i)$) is calculated by $\boldsymbol{c}(p_i) - \sum_{t_{ij}\ running\ on\ p_i} \boldsymbol{r}(t_{ij})$. The resource to be allocated to a task t' must conform to Inequality (4), where $\preceq$ means componentwise inequality between two vectors.

In our model, t_{ij}'s execution length (or execution time) is defined as $\boldsymbol{l}(t_{ij})^T \cdot \boldsymbol{r}(t_{ij})^{-1} + \triangle$ (Equation (2)), where $\boldsymbol{r}(t_{ij})^{-1}$=$(r_1^{-1}(t_{ij}),r_2^{-1}(t_{ij}),\cdots,r_R^{-1}(t_{ij}))^T$ and $\triangle$ implies a constant extra cost (such as VM-loading time). Such a definition of execution time specifies a broad set of applications, each of which needs a series of phases to process (or mixed non-overlapped executions on various types of resources). For example, computing the matrix formula $(A_{m\times n} \cdot A_{n\times m})^k \boldsymbol{x}$=$\boldsymbol{d}_{m\times 1}$ could be split into 5 phases - loading matrix from disk, computing the product $C_{m\times m}$=$A_{m\times n} \cdot A_{n\times m}$, computing matrix-power $C_{m\times m}^k$, solving $C_{m\times m}\boldsymbol{x}$=$\boldsymbol{d}_{m\times 1}$, and storing $\boldsymbol{x}$ onto disk.

The R types of resources (or R execution dimensions) are associated with a price vector denoted as $\boldsymbol{b}(p_i)$=$(b_1(p_i), b_2(p_i), \cdots, b_R(p_i))^T$. Let $b_k(p_i)$ $(1 \leq k \leq R)$ denote the per-time-unit price the consumers need to pay for the resource consumption of p_i at kth execution dimension. Then, t_{ij}'s total payment $\rho(t_{ij}, \Delta t)$ can be calculated via Equation (1), where Δt refers to t_{ij}'s execution period on p_i.

$$\rho(t_{ij}, \Delta t) = \Delta t \cdot \boldsymbol{b}(p_i)^T \cdot \boldsymbol{r}(t_{ij}) \tag{1}$$

We argue that it is non-trivial to precisely predict task execution length and quantify the resource consumption at each dimension, thus it is inviable for users to forecast their total payment in advance. Hence, we adopt the pay-by-reserve policy, in which the users could get the reserved resources for the task execution. The users feel happy as long as the per-time-unit rate is always within an acceptable budget (denoted $B(t_{ij})$), i.e., Formula (3). Such a payment policy is widely adopted by Cloud systems like OpenPEX [14], and also recommended by Amazon EC2 for cost modeling research [15].

In the following text, we might omit the symbols t_{ij} and p_i if thus would not cause ambiguity (especially when a task is already determined). For instance, $l_k(t_{ij})$, $\boldsymbol{r}(t_{ij})$, $b_k(p_i)$, $\boldsymbol{a}(p_i)$ and $B(t_{ij})$ may be substituted by l_k, $\boldsymbol{r}$, b_k, $\boldsymbol{a}$, and B respectively.

Finally, the resource allocation problem is modeled as a convex optimization problem: for a submitted task t' with its workload vector $\boldsymbol{l}(t')$, how to minimize its execution length (i.e., Equation (2)) subject to constraints (3) and (4), where p_e is selected from among all managed hosts and $\boldsymbol{r}(t', p_e)$ means executing t' on p_e).

$$f(\boldsymbol{r}(t', p_e)) = \boldsymbol{l}^T(t') \cdot \boldsymbol{r}^{-1}(t') + \triangle = \sum_{i=1}^{R} \frac{l_i}{r_i} + \triangle \tag{2}$$

$$\boldsymbol{b}(p_e)^T \cdot \boldsymbol{r}(t') \leq B(t'), \ where\ p_e\ is\ execution\ node \tag{3}$$

$$\boldsymbol{r}(t') \preceq \boldsymbol{a}(p_e) \tag{4}$$

III. Optimal Divisible-Resource Allocation (ODRA)

We outline the pseudo-code of the skeleton algorithm in Algorithm 1, which describes how to select nodes for the specific task t'. The input list contains two parts, resource information and task information. The former includes the candidate node set S=$\{p_1,p_2,\cdots,p_n\}$, price vector set P=$\{\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n\}$ and availability vector set A=$\{\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_n\}$; the latter includes t''s budget $B(t')$ and its predicted workload vector $\boldsymbol{l}(t')$.

Algorithm 1 SKELETON OF ODRA ALGORITHM

```
Input: $S$, $P$, $A$, $B(t')$, $\boldsymbol{l}(t')$;
Output: execution node $p_e$, $\boldsymbol{r}^*(t')$
1: $p_e$=$p_1$;
2: for (each node $p_i$ in $S$) do
3:     $\boldsymbol{r}^*(t',p_i)$ = LOAA($B(t')$,$\boldsymbol{l}(t')$,$\boldsymbol{a}_i$,$p_i$); /*Calculate the optimal resource allocation for task $t'$ running on node $p_i$.*/
4:     Estimate $f(p_i)$ based on $\boldsymbol{r}^*(t',p_i)$ /*Presume $t'$'s time*/
5:     if ($f(p_i)$<$f(p_e)$) then
6:         $p_e$=$p_i$;
7:         $\boldsymbol{r}^*(t')$=$\boldsymbol{r}^*(t',p_i)$;
8:     end if
9: end for
```

According to Algorithm 1, Line 3 aims to perform a Local Optimal divisible-resource Allocation Algorithm (*LOAA*) on each physical node with low time complexity (to be described in Algorithm 2). This is the most critical step in that the rest part (Line 4∼7) just selects the node which can achieve the shortest execution length.

The optimal solution to the local resource allocation could leverage convex optimization theory [10]. We denote the optimal resource fraction vector as $\boldsymbol{r}^*(t_{ij})$ or $\boldsymbol{r}^*$, i.e., the resource vector that minimizes the task execution length, subject to payment budget and resource availability. However, it is non-trivial to directly solve it as analyzed in our previous work [16]). To this end we devise an algorithm with only $O(R^2)$ to find the optimal solution. As follows, we first present the optimal resource fraction without the constraint (4) in Theorem 1, and then present the optimal algorithm (LOAA) that takes into account constraint (4).

Theorem 1: In order to minimize $f(\boldsymbol{r}(t_{ij}))$ subject to the constraint (3), t_{ij}'s optimal received resource vector $\boldsymbol{r}^{(*)}(t_{ij})$ is shown as Equation (5), where k=1, 2, $\cdots$, R. (Note that $\boldsymbol{r}^{(*)}(t_{ij})$ is not subject to Inequality (4), unlike the notation $\boldsymbol{r}^*(t_{ij})$ that relies on Inequality (4).)

$$r_k^{(*)}(t_{ij}) = \frac{\sqrt{l_k(t_{ij})/b_k(p_i)}}{\sum\limits_{k=1}^{R}\sqrt{l_k(t_{ij})b_k(p_i)}} \cdot B(t_{ij}) \qquad (5)$$

Proof: Since $\frac{\partial^2 f(\boldsymbol{r})}{\partial r_k^2}$=$2\frac{l_k}{r_k^3}$>0, $f(\boldsymbol{r})$ is convex with a minimum extreme point. Then, we can easily derive Equation (6) using the Lagrangian multiplier method.

$$r_1 : r_2 : \cdots : r_R = \sqrt{\frac{l_1}{b_1}} : \sqrt{\frac{l_2}{b_2}} : \cdots : \sqrt{\frac{l_R}{b_R}} \qquad (6)$$

In order to minimize $f(\boldsymbol{r})$, the optimal resource vector $\boldsymbol{r}^{(*)}$ should make $\boldsymbol{b}(p_i)^T \cdot \boldsymbol{r}(t_{ij})$ equal to $B(t_{ij})$. By combining this equation with Equation (6), we derive Equation (5). ■

Remark: By combining the constraint (4), $\boldsymbol{r}^{(*)}$ based on Equation (5) is right the optimal solution as long as $\boldsymbol{r}^{(*)} \preceq \boldsymbol{a}(p_i)$. However, if $\boldsymbol{r}^{(*)}$ does not fully satisfy the constraint (4) (i.e., $\exists$ k: $r_k^{(*)}$>$a_k(p_i)$), $\boldsymbol{r}^{(*)}$ should not be a viable solution. Hence, we propose an efficient algorithm (Algorithm 2) to find the optimal solution subject to the constraint (4) with the provable time complexity $O(R^2)$.

Definition 1: Given a specific budget C for task t_{ij}'s execution in a subset $\Gamma(\subseteq\Pi$, i.e., subset of the execution dimension set), CO-STEP(Γ, C) is defined as the procedure in computing the optimal solution of minimizing $f(\boldsymbol{r}_\Gamma(t_{ij}))$ subject to the constraint (7) by using convex optimization (similar to the proof of Theorem 1), where $\boldsymbol{r}_\Gamma(t_{ij})$ and $\boldsymbol{b}_\Gamma^T(p_i)$ denote the resource fractions gained by t_{ij} and the price vector assigned by p_i w.r.t. Γ respectively.

$$\boldsymbol{b}_\Gamma^T(p_i) \cdot \boldsymbol{r}_\Gamma(t_{ij}) \leq C, \quad where\ C\ is\ a\ constant. \qquad (7)$$

Algorithm 2 (namely *LOAA*) is devised for minimizing $f(\boldsymbol{r}(t_{ij}))$ subject to the constraints (3) and (4).

Algorithm 2 LOCAL OPTIMAL ALLOCATION ALGORITHM

```
function name: LOAA($\Pi$, $B(t_{ij})$, $\boldsymbol{l}(t_{ij})$, $\boldsymbol{b}(p_i)$, $\boldsymbol{a}(p_i)$);
Input:  $\Pi$: the execution dimension set (universe)
        $B(t_{ij})$: $t_{ij}$'s budget
        $\boldsymbol{l}(t_{ij})$: $t_{ij}$'s predicted workload vector
        $\boldsymbol{b}(p_i)$: execution node $p_i$'s price vector
        $\boldsymbol{a}(p_i)$: execution node $p_i$'s availability vector
Output: $\boldsymbol{r}^*(t_{ij})$: $t_{ij}$'s optimal resource allocation vector on $p_i$
1: $\Gamma$ = $\Pi$, $C$ = $B(t_{ij})$, $\boldsymbol{r}^*$ = $\varnothing$ (empty set);
2: repeat
3:     $\boldsymbol{r}_\Gamma^{(*)}$ = CO-STEP($\Gamma$,$C$); /*Compute optimal $r^{(*)}$ based on $\Gamma$ with unbounded capacity assumption*/
4:     $\Omega$={$d_k$|$d_k\in\Gamma$ & $r_k^{(*)}$>$a_k$};/*select elements violating constraint (4)*/
5:     $\Gamma$ = $\Gamma\backslash\Omega$; /*$\Gamma$ takes away $\Omega$*/
6:     $C$ = $C - \sum_{d_k\in\Omega}(b_k \cdot a_k)$; /*Update $C$*/
7:     $\boldsymbol{r}^*$ = $\boldsymbol{r}^*\cup${$r_k^*$ = $a_k$ | $d_k\in\Omega$ & $a_k$ is $d_k$'s upper bound};
8: until ($\Omega$ = $\varnothing$);
9: $\boldsymbol{r}^*$ = $\boldsymbol{r}^*$ $\cup$ $\boldsymbol{r}_\Gamma^{(*)}$;
```

In this algorithm, line 3 executes CO-STEP(Γ,C) in order to find the optimal $\boldsymbol{r}_\Gamma^{(*)}$, without considering the constraint (4). If $\boldsymbol{r}_\Gamma^{(*)}$ happens to completely satisfy the constraint (4) (i.e., Ω=$\varnothing$), then it is the final result. Otherwise, the resource fractions (r_k) that violate the constraint (4) will be set to the upper bounds (i.e., a_k) and the corresponding dimensions (i.e., Ω) will be taken away from Γ, then, $C = C - \sum_{d_k\in\Omega}(b_k \cdot a_k)$ for the remaining dimensions. The process will go on until all the remaining computed optimal resource fractions satisfy the constraint (4). Since the time complexity of CO-STEP(Γ,C) is $O(|\Gamma|)$, the number of computation steps of Algorithm 2 in the worst case is $\sum_{i=0}^{R-1}(R-i)$, thus the time complexity=$O(R^2)$.

Theorem 2: Given a submitted task t_{ij} with a workload vector $\boldsymbol{l}(t_{ij})$ and a budget $B(t_{ij})$ and a qualified node p_i with its resource price vector $\boldsymbol{b}(p_i)$, Algorithm 2's output $\boldsymbol{r}^*$ is optimal for minimizing t_{ij}'s execution length (i.e., $f(\boldsymbol{r}(t_{ij}))$), subject to the constraint (3) and constraint (4).

We can prove Algorithm 2's output must satisfy the sufficient and necessary conditions of optimal solution. We omit the detailed proof, which can be found in [16].

IV. Optimality Analysis with Workload Prediction Errors

A. Problem Description

Although Algorithm 2 is proved optimal, such optimality is subject to a strong assumption that the workload vector is accurately estimated. In reality, task's workloads may not be precisely predicted by ordinary users. In this section, we derive the upper bound of task execution length under Algorithm 2 with possible workload prediction errors.

Definition 2: Suppose a task t_{ij}'s estimated workload vector is $\boldsymbol{l}'(t_{ij})$ but the real workload vector is $\boldsymbol{l}(t_{ij})$. Then, $\boldsymbol{l}'(t_{ij})$ satisfies Inequality (8), where α and β are the lower bound and upper bound for the estimation ratio.

$$\alpha \le \frac{l'_k(t_{ij})}{l_k(t_{ij})} \le \beta,\ k = 1, 2, \cdots R \tag{8}$$

We use the following example to illustrate the above definition. Suppose the task t_{ij}'s real workload vector always ranges in [125, 1000] single-core-length, and the workload vector $\boldsymbol{l}'(t_{ij})$ used by Algorithm 2 is based on the history of the task's execution. Each element $l'_k(t_{ij})$ $(k = 1, 2, \cdots R)$ will be set to 250 if the corresponding true workload fluctuates in [125, 500] and set to 750 if the true workload ranges in (500, 1000]. Then, we could get Inequality (9) below, where α=$\frac{250}{500}$=0.5 and β=$\frac{250}{125}$=2.

$$0.5 \le \frac{l'_k(t_{ij})}{l_k(t_{ij})} \le 2,\ k = 1, 2, \cdots R \tag{9}$$

It is obvious that with the inaccurate prediction of task's workload, the output of Algorithm 2 will definitely be skewed from the result with accurate information. Hence, one question is how far the output produced by Algorithm 2 based on $\boldsymbol{l}'(t_{ij})$ would be away to the result based on $\boldsymbol{l}(t_{ij})$. That is, our objective is to derive an upper bound of task execution length for Algorithm 2 based on erroneous workload information (i.e., Inequality (8)).

B. Analysis with Erroneous Workload Prediction

We analyze the worst execution performance with erroneous workload prediction in Theorem 3 and Theorem 4.

For simplicity of description, we denote $\boldsymbol{r}^*_E$ (=(r^*_{E1}, r^*_{E2}, $\cdots$, $r^*_{ER})^T$) and f^*_E (=$\sum_{k=1}^{R}\frac{l_k}{r^*_{Ek}}$+$\triangle$) as the output of Algorithm 2 with the erroneous workload estimation and the corresponding execution length respectively, and E indicates "Estimation with error". Similarly, we denote $\boldsymbol{r}^*_I$ (=(r^*_{I1}, r^*_{I2}, $\cdots$, $r^*_{IR})^T$) and f^*_I (=$\sum_{k=1}^{R}\frac{l_k}{r^*_{Ik}}$ + $\triangle$) as the output with precise workload vector and the corresponding execution length, respectively, and I indicates "Ideal case". Hence, our objective is to determine the upper bound of $\frac{f^*_E}{f^*_I}$, a.k.a., *approximation ratio*.

Denote $\boldsymbol{r}^{(*)}_E$ the optimal resource allocation with unbounded resource capacities. The output of Algorithm 2 could be split into two situations:

- case 1: $\boldsymbol{r}^*_E(t_{ij}) = \boldsymbol{r}^{(*)}_E(t_{ij})$.
- case 2: $\boldsymbol{r}^*_E(t_{ij}) \neq \boldsymbol{r}^{(*)}_E(t_{ij})$.

The first situation indicates that in terms of the skewed workload prediction, all of the resource fractions calculated by the initial CO-STEP in Algorithm 2 happen to be no greater than the corresponding resource capacities. That is, the output of the first-round CO-STEP complies with the Inequality (10).

$$\boldsymbol{r}^{(*)}_E(t_{ij}) \preceq \boldsymbol{a}(p_i) \tag{10}$$

In contrast, the second situation means that the initial CO-STEP cannot fulfill the above condition, and the optimal allocation cannot be found until a few more adjustment steps (line 4 $\sim$ line 7 of Algorithm 2).

We first derive the upper bound of task t_{ij}'s execution length for the first case (i.e., Theorem 3), and then discuss the upper bound (i.e., Theorem 4) for the second one.

Theorem 3: **Given** a task t_{ij} with a budget $B(t_{ij})$, a node p_i whose resource price vector is $\boldsymbol{b}(p_i)$, and a inaccurate workload vector $\boldsymbol{l}'(t_{ij})$ subject to Inequality (8), **then** the tight upper bound of t_{ij}'s execution length under the resource allocation $\boldsymbol{r}^{(*)}_E$ conforms to Inequality (11).

$$\frac{f^{(*)}_E}{f^*_I} \le \sqrt{\frac{\beta}{\alpha}} \tag{11}$$

Main idea of proof: It is obvious that $\boldsymbol{r}^{(*)}_I$ must be no worse than $\boldsymbol{r}^*_I$ (i.e., $f^{(*)}_I \le f^*_I$ must always hold), because $\boldsymbol{r}^*_I$ is the result with more constraints. Thus, we could get the final conclusion as long as we could prove Inequality (12).

$$\frac{f^{(*)}_E}{f^{(*)}_I} \le \sqrt{\frac{\beta}{\alpha}} \tag{12}$$

Proof: As mentioned previously, we omit the notation t_{ij} for simplicity of expression. For example, B indicates $B(t_{ij})$. In terms of Theorem 1 and Equation (5), we could get the following deduction:

$$\begin{aligned}
f^{(*)}_E &= \sum_{k=1}^{R}\frac{l_k}{r^{(*)}_{Ek}} + \triangle = \sum_{k=1}^{R}\frac{l_k}{B\frac{\sqrt{l'_k/b_k}}{\sum_{i=1}^{R}\sqrt{l'_i b_i}}} + \triangle \\
&= \frac{1}{B}\sum_{k=1}^{R} l_k\Big(\frac{\sum_{i=1}^{R}\sqrt{l'_i b_i}}{\sqrt{l'_k/b_k}}\Big) + \triangle \\
&\le \frac{1}{B}\sum_{k=1}^{R} l_k\Big(\frac{\sum_{i=1}^{R}\sqrt{(\beta l_i) b_i}}{\sqrt{(\alpha l_k)/b_k}}\Big) + \triangle \quad \{due\ to\ Inequality\ (8)\} \\
&\le \sqrt{\frac{\beta}{\alpha}}\frac{1}{B}\sum_{k=1}^{R} l_k\Big(\frac{\sum_{i=1}^{R}\sqrt{l_i b_i}}{\sqrt{l_k/b_k}}\Big) + \sqrt{\frac{\beta}{\alpha}}\triangle \quad \{Note\ \sqrt{\frac{\beta}{\alpha}} \ge 1\} \\
&= \sqrt{\frac{\beta}{\alpha}} f^{(*)}_I \le \sqrt{\frac{\beta}{\alpha}} f^*_I
\end{aligned}$$

Thus, the Inequality (11) holds.

We use a case to illustrate that Inequality (11)'s upper bound is tight: Let $\boldsymbol{l}'$ always be equal to $2\boldsymbol{l}$, then, α=β=2, that is, $\sqrt{\frac{\beta}{\alpha}}$=1. Then, $f_E^{(*)} \leq f_I^*$, according to Inequality (12). On the other hand, suppose $\boldsymbol{r}_E^{(*)}$ satisfies Inequality (10), then $f_E^* = f_E^{(*)} = f_I^*$, which means that the upper bound is reached in this situation. ■

Note that if the Inequality (10) does not hold, the output of Algorithm (2) immediately becomes very non-trivial. This is because the optimal allocated resource fractions will not strictly conform to the Equation (6) any more, due to the fact that some fractions (say $r_k(t_{ij})$) have to be set equal to the corresponding available capacities (i.e., $a_k(p_i)$). However, with the in-depth analysis, we can still find an upper bound for such a situation, which will be proved in Theorem 4. Before proving this new theorem, it is necessary to introduce a new definition and Lemma 1, which will be used later.

Definition 3: Suppose Γ is a set of execution dimensions selected from the complete set Π (i.e., $\Gamma \subseteq \Pi$). Considering the dimensions $d_k \in \Pi \backslash \Gamma$, $r_{Ik}^{[*]}(\Gamma)$ denotes the convex-optimal resource-share calculated by aiming to minimize the target function $f_I(\Gamma)$ subject to the condition (13), where $\Pi \backslash \Gamma$ denotes Γ's complement.

$$\sum\nolimits_{d_k \in \Pi \backslash \Gamma} r_{Ik} b_k = B - \sum\nolimits_{d_i \in \Gamma} r_{Ii}^* b_i \tag{13}$$

Based on Definition 3 and Theorem 1, we could derive $r_{Ik}^{[*]}(\Gamma)$ as Equation (14), since $\sum_{d_i \in \Gamma} \frac{l_i}{r_{Ii}^*}$ and $\sum_{d_i \in \Gamma} r_{Ii}^* b_i$ could be considered two constants here.

$$r_{Ik}^{[*]}(\Gamma) = (B - \sum_{d_i \in \Gamma} r_{Ii}^* b_i) \frac{\sqrt{l_k / b_k}}{\sum\limits_{d_i \in \Pi \backslash \Gamma} \sqrt{l_i b_i}} \tag{14}$$

The key rule of the above definition is that the resource fractions outside Γ are optimized stationary points while those inside are not. For simplicity of description, we use $r_{Ik}^{[*]}$ to substitute $r_{Ik}^{[*]}(\Gamma)$ in the following text, as long as thus would not cause ambiguity. Further more, we denote $f_I^{[*]}(\Gamma)$ as the execution time in terms of the resource fraction set $\{r_{Ii}^* | d_i \in \Gamma\} \cup \{r_{Ik}^{[*]} | d_k \in \Pi \backslash \Gamma\}$. Hence, $f_I^{[*]}(\Gamma)$ could be denoted as Equation (15).

$$f_I^{[*]}(\Gamma) = \sum\nolimits_{d_i \in \Gamma} \frac{l_i}{r_{Ii}^*} + \sum\nolimits_{d_k \in \Pi \backslash \Gamma} \frac{l_k}{r_{Ik}^{[*]}} + \triangle \tag{15}$$

Lemma 1: $f_I^{[*]}(\Gamma)$ is no larger than f_I^* (i.e., Inequality (16) must hold), where f_I^* is the task execution length with r^*.

$$f_I^{[*]}(\Gamma) \leq f_I^* \tag{16}$$

Proof: Denote $\boldsymbol{r}'_I = (r'_{I1}, r'_{I2}, \cdots, r'_{IR})^T$, where r'_{Ii} is selected from $\{r_{Ii}^* | d_i \in \Gamma\} \cup \{r_{Ii}^{[*]} | d_i \in \Pi \backslash \Gamma\}$, conforming to the Equation (17).

$$r'_{Ii} = \begin{cases} r_{Ii}^* & d_i \in \Gamma \\ r_{Ii}^{[*]} & d_i \in \Pi \backslash \Gamma \end{cases} \tag{17}$$

Based on Definition 3, $r_{Ii}^{[*]}$ ($\forall d_i \in \Pi \backslash \Gamma$) are all stationary points such that $f_I(\Gamma)$ is minimized using convex optimization. In contrast, r_{Ii}^* ($\forall d_i \in \Gamma$) may not be stationary points based on the Definition 3. Note that all of r_{Ii}^* (i=1,2,$\cdots$,R) together also conforms to the Formula (13), thus $f_I^{[*]}(\Gamma)$ must be a better solution than f_I^*, which means $f_I^{[*]}(\Gamma) \leq f_I^*$. ■

Theorem 4: **Given** a task t_{ij} with a budget B, a resource node whose available resource vector and price vector are $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively, and an erroneous workload vector $\boldsymbol{l}'$ subject to Inequality (8), **then**, the tight upper bound of t_{ij}'s execution length with resource allocation $\boldsymbol{r}_E^*$ conforms to Inequality (18), where Ω refers to the set of dimensions constructed at Line 4 of Algorithm 2 performed with the inaccurate workload vector ($\boldsymbol{l}'$) and r_{Ii}^* is the optimal allocated resource fraction outputted when using the real workload vector (i.e., $\boldsymbol{l}$).

$$\frac{f_E^*}{f_I^*} \leq \theta \sqrt{\frac{\beta}{\alpha}}, \ \ where\ \theta = \frac{B - \sum_{di \in \Omega} r_{Ii}^* b_i}{B - \sum_{di \in \Omega} a_i b_i} \tag{18}$$

Proof: If Inequality (10) holds (i.e., $\boldsymbol{r}_E^{(*)} \preceq \boldsymbol{a}$), it is obvious that $\boldsymbol{r}_E^* = \boldsymbol{r}_E^{(*)}$. That is, $f_E^* = f_E^{(*)}$. Besides, since r_{Ii}^* (i=1,2,$\cdots$,R) indicates one solution (more specifically, it is actually the optimal resource fraction with accurate workload vector) outputted by Algorithm 2, we have $r_{Ii}^* \leq a_i$. Hence, Inequality (19) must hold.

$$\frac{B - \sum_{d_i \in \Omega} r_{Ii}^* b_i}{B - \sum_{d_i \in \Omega} a_i b_i} \geq 1 \tag{19}$$

Based on the Inequality (11) and Inequality (19), we can get Equation (18) when $\boldsymbol{r}_E^{(*)} \preceq \boldsymbol{a}$.

As follows, we will intensively discuss the situation with $\boldsymbol{r}_E^{(*)} \npreceq \boldsymbol{a}$ (i.e., there exists at least one resource dimension $d_j \in \Omega$ such that its convex-optimal resource fraction $r_j^{(*)}$ after the first-round CO-STEP in Algorithm 2 is larger than its available capacity a_j). In this situation, $\Omega \neq$ empty set ($\varnothing$) as the whole Algorithm 2 ends. Without loss of generality, let the index numbers of the dimensions in Ω be 1, 2, $\cdots$, $|\Omega|$.

As for $\boldsymbol{r}_E^*$, according to Algorithm 2, we know that $r_{E1}^* = a_1$, $r_{E2}^* = a_2$, $\cdots$, $r_{E|\Omega|}^* = a_{|\Omega|}$ and $r_{E(|\Omega|+1)}^*$, $\cdots$, r_{ER}^* are right the stationary points computed by convex optimization subject to the Equation (20).

$$\sum\nolimits_{i=|\Omega|+1}^{R} r_{Ei}^* b_i = B - \sum\nolimits_{i=1}^{|\Omega|} a_i b_i \tag{20}$$

Hence, we could get Equation (21) as follows.

$$r_{E(|\Omega|+1)}^* \cdots : r_{ER}^* = \sqrt{\frac{l_{|\Omega|+1}}{b_{|\Omega|+1}}} : \cdots : \sqrt{\frac{l_R}{b_R}} \tag{21}$$

Then, we could get the following derivation.

$$\begin{aligned}f_E^* - \textstyle\sum_{i=1}^{|\Omega|} \frac{l_i}{a_i} - \triangle &= \textstyle\sum_{k=|\Omega|+1}^{R} \frac{l_k}{r^*_{Ek}} \\ &= \sum_{k=|\Omega|+1}^{R} \left(\frac{l_k}{\frac{\sqrt{l'_k/b_k}}{\sum_{i=|\Omega|+1}^{R} \sqrt{l'_i b_i}} \cdot (B - \sum_{i=1}^{|\Omega|} a_i b_i)} \right) \\ &= \frac{1}{(B - \sum_{i=1}^{|\Omega|} a_i b_i)} \sum_{k=|\Omega|+1}^{R} \left(l_k \cdot \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l'_i b_i}}{\sqrt{l'_k/b_k}} \right)\end{aligned} \quad (22)$$

Based on Equation (22), we could get Equation (23):

$$(f_E^* - \sum_{i=1}^{|\Omega|} \frac{l_i}{a_i} - \triangle) \cdot (B - \sum_{i=1}^{|\Omega|} a_i b_i) = \sum_{k=|\Omega|+1}^{R} (l_k \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l'_i b_i}}{\sqrt{l'_k/b_k}}) \quad (23)$$

On the other hand, we should also study f_I^* and compare it to f_E^*. First of all, as for the optimal allocation vector $\boldsymbol{r}_E^*$, let us select the resource fractions whose dimensions belong to Ω. Please note that this Ω is the one outputted by Algorithm 2 using the "erroneous estimated workload vector" instead of the accurate workload vector. Then, we could easily get the identical equation (24).

$$f_I^* = \sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} + \sum_{i=|\Omega|+1}^{R} \frac{l_i}{r^*_{Ii}} \quad (24)$$

According to Lemma 1, i.e., Inequality (16), we have $f_I^{[*]} \leq f_I^*$. To this end, in order to prove Formula (18), we can switch the target to comparing f_E^* and $f_I^{[*]}$.

Based on the Definition 3 (let $\Gamma = \Omega$), we could derive the following equations about $f_I^{[*]}$:

$$\begin{aligned}f_I^{[*]} - \sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} - \triangle &= \sum_{k=|\Omega|+1}^{R} \frac{l_k}{r^{[*]}_{Ik}} \\ &= \sum_{k=|\Omega|+1}^{R} \frac{l_k}{(B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i) \frac{\sqrt{l_k/b_k}}{\sum_{i=|\Omega|+1}^{R} \sqrt{l_i b_i}}} \\ &= \frac{1}{(B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i)} \sum_{k=|\Omega|+1}^{R} \left(l_k \cdot \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l_i b_i}}{\sqrt{l_k/b_k}} \right)\end{aligned} \quad (25)$$

Then, we could get Equation (26).

$$(f_I^{[*]} - \sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} - \triangle) \cdot (B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i) = \sum_{k=|\Omega|+1}^{R} (l_k \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l_i b_i}}{\sqrt{l_k/b_k}}) \quad (26)$$

Further more, we could get the following inequality based on Inequality (8).

$$\sum_{k=|\Omega|+1}^{R} (l_k \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l'_i b_i}}{\sqrt{l'_k/b_k}}) \leq \sqrt{\frac{\beta}{\alpha}} \sum_{k=|\Omega|+1}^{R} (l_k \frac{\sum_{i=|\Omega|+1}^{R} \sqrt{l_i b_i}}{\sqrt{l_k/b_k}}) \quad (27)$$

By combining Equation (23), Equation (26) and Inequality (27), we could further get the Inequality (28).

$$\begin{aligned}&(f_E^* - \sum_{i=1}^{|\Omega|} \frac{l_i}{a_i} - \triangle) \cdot (B - \sum_{i=1}^{|\Omega|} a_i b_i) \\ &\leq \sqrt{\frac{\beta}{\alpha}} \cdot (f_I^{[*]} - \sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} - \triangle) \cdot (B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i)\end{aligned} \quad (28)$$

Consequently, we could get the following deductions:

$$\begin{aligned}f_E^* &\leq \sqrt{\frac{\beta}{\alpha}} \cdot \frac{B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i}{B - \sum_{i=1}^{|\Omega|} a_i b_i} \cdot (f_I^{[*]} - \sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} - \triangle) + \sum_{i=1}^{|\Omega|} \frac{l_i}{a_i} + \triangle \\ &= \sqrt{\frac{\beta}{\alpha}} \cdot \frac{B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i}{B - \sum_{i=1}^{|\Omega|} a_i b_i} \cdot f_I^{[*]} + \sum_{i=1}^{|\Omega|} \frac{l_i}{a_i} + \triangle - \sqrt{\frac{\beta}{\alpha}} \frac{B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i}{B - \sum_{i=1}^{|\Omega|} a_i b_i} (\sum_{i=1}^{|\Omega|} \frac{l_i}{r^*_{Ii}} + \triangle) \\ &\leq \sqrt{\frac{\beta}{\alpha}} \cdot \frac{B - \sum_{i=1}^{|\Omega|} r^*_{Ii} b_i}{B - \sum_{i=1}^{|\Omega|} a_i b_i} \cdot f_I^{[*]}\end{aligned}$$

Note that the last step in the above derivation is due to the fact that $\alpha \leq \beta$ and $\forall d_i \in \Omega$, $r^*_{Ii} \leq a_i$. By further taking Inequality (16) into account, we get the conclusion that $\frac{f_E^*}{f_I^*} \leq \theta \sqrt{\frac{\beta}{\alpha}}$, where $\theta = \frac{B - \sum_{di \in \Omega} r^*_{Ii} b_i}{B - \sum_{di \in \Omega} a_i b_i}$.

As follows, we will use a use-case to argue that the upper bound is tight. Suppose the number of dimensions is 2, i.e., $|\Pi|$=2, and a task's load vector is $(l_1, l_2)^T$. We consider the following two cases, each of which makes $f_E^* = f_I^*$:

- If Ω is empty set and $a_1 b_1 + a_2 b_2 \leq B$, then the optimal solution will always be $\boldsymbol{a} = (a_1, a_2)^T$, regardless of the values of α and β, thus $f_E^* = f_I^*$.
- If Ω is non-empty set and $|\Omega|$=1 and let $d_1 \in \Omega$ without loss of generality, we take into account such a case that $\frac{l'_1}{l_1} = \frac{l'_2}{l_2}$, then we could get that $\alpha = \beta$ and $r^{(*)}_{Ei} = r^{(*)}_{Ii}$ where i=1,2. As such, we could easily deduce that $r^*_{Ei} = r^*_{Ii}$, which also implies $f_E^* = f_I^*$.

■

V. Performance Evaluation

A. Experimental Setting

We evaluate our Optimal Divisible-Resource Allocation (ODRA) algorithm in a real cluster environment, called Gideon-II [17], which is the most powerful super computer at Hong Kong. We are assigned 8 physical nodes connected with 10Gbps high-speed intra-network. Each node has 8 2.45MHz-cores and 16GB of memory size. We deployed XEN 4.0 [18] on each node. Since there must be one core reserved for XEN hypervisor, we created 56(=8×7) VM instances (centos 5.2) on the 8 physical nodes. Three types of resource attributes (CPU rate, network bandwidth, and I/O disk bandwidth) will be split on demand according to our ODRA algorithm. Specifically, we make use of XEN's credit scheduler [12] to dynamically allocate various CPU rates (or capabilities) to the VM-instances. The network bandwidth and the disk reading/writing rate allocated to each user are both reshaped by linux network traffic controller (TC) [13] on demand at run-time.

In our experiment, each user task is constructed by multiple subtasks, each corresponding to various web services with heterogeneous workloads to process. The subtasks could also be data transmission via network or data read/write through disk. So, each subtask could be regarded as an execution dimension with a particular workload to process (such as number of float points, data to transmit) and a resource capacity to allocate (such as CPU rate and network bandwidth).

By leveraging *ParallelColt* [19] (a library of matrix-computation), we implement 10 matrix computation programs in the form of web services (listed in Table I), which can be further combined to construct more complex matrix problems (i.e., user tasks). The number of subtasks per task is randomly set in [5, 15], and each subtask is a matrix computation selected in Table I. The matrices in our experiments are randomly generated with the scales from 100×100 to 2500×2500, and their data sizes range from 192KB ($\{100\times100\}$) to 115MB ($\{2500\times2500\}$).

The tasks emulated in our experiment represent different execution patterns, e.g., CPU bound, memory bound or others. First, ten different matrix computations have various workloads. In Table I, we show their heterogeneous single-core execution lengths w.r.t. square matrices, where M refers to matrix size and $m(\in [10, 20])$ is the exponent in the matrix-power computation. Second, each task execution involves three types of resources (CPU rate, network bandwidth and I/O disk bandwidth) and possibly multiple execution nodes. For a particular task, its first subtask is reading one or more matrix data from disk drive. The second subtask could be some matrix computation like matrix product, decomposition, or others. As the entire matrix computation is finished, its output (a new matrix) will be transmitted to another VM instance through the network. The data transmission is also a kind of subtask whose workload and capacity is data size to transmit and the network bandwidth controlled on demand. The last subtask of one task is storing the final matrix result into the disk drive. The CPU rate assigned to VMs is controlled by XEN's credit scheduler [12]. Network bandwidth and disk I/O rate are both controlled within [10,300] Mb/s over NFS through Linux network traffic controller (TC). The CPU capacity of each multi-threaded program (e.g., matrix product, matrix power) is set to 8-cores (i.e., the maximum processing ability of one physical node), while that of any single-threaded program (such as matrix decomposition) is set to 1-core in that more resources cannot get further speedup on it.

We predict the workload of each matrix computation based on history for simplicity, where α and β are set to 0.7 and 2 respectively based on our characterization about prediction errors. In practice, one could use more accurate methods like multi-variate polynomial regression method [11], whose margin of prediction errors is 7-10%.

Each task is associated with a budget, which is a key factor impacting task's resource allocation. The prices of the 10 matrix operations are set to $1,2,\cdots,10$ with the decrease of their workloads. We evaluate our algorithm with different budgets assigned, which are simulated based on Formula (29), where $b(F_i)$ and θ (=0.5∼3) refer to the price of the function F_i and a coefficient to tune users' various economic strengths respectively.

$$\tau' s\ budget = \theta \sum\nolimits_{i=1}^{K} b(F_i) \tag{29}$$

Upon receiving a task made up of multiple consecutive matrix computations, our designed algorithm is triggered to compute the optimal resource vector for it and perform resource isolation (e.g., notifying corresponding VMM to customize the CPU rates by credit scheduler).

We compare the experimental results under our algorithm to two types of theoretically optimal results. The first one is the ideal execution length (i.e., $f_I^{(*)}$) based on the ideal convex-optimal resource vector calculated by Theorem 1, with the assumption that the resource capacities are unbounded. We call it *ideal optimal time (IOT)*. The second one is the execution length (i.e., f_I^*) based on the practical optimal resource vector under the limited capacities in reality. We call it *practical optimal time (POT)*.

B. Experimental Results

There are two key evaluation metrics. The first one is called *execution stretch* (ES), aiming to evaluate task's execution performance. A task's ES is defined as its real execution length (with possibly erroneous workloads predicted) divided by its theoretically optimal execution length calculated based on its real workloads recorded after its execution. Smaller ES implies higher execution efficiency and ES being equal to 1 implies that the task's practical execution length reaches its theoretically optimal result. The other one is called *performance-payment ratio* (PPR), which is used to evaluate the effectiveness of user's payment. A task τ's PPR is defined in Formula (30), where τ's payment level is equal to $\frac{\tau' s\ final\ payment}{\tau' s\ budget}$. The smaller PPR, the higher performance with lower payment meanwhile, indicating higher satisfactory level.

$$PPR(\tau) = ES(\tau) \times (\tau' s\ payment\ level) \tag{30}$$

We first evaluate the impact of different budgets assigned to a single task to its execution performance and user's final payment. The task is computing $||(A^2_{2000\times2000}\times((A_{2000\times2000}\times A_{2000\times1000})\times\boldsymbol{v}_{1000}))\times\boldsymbol{v}_{1000}||_2$, where A and $\boldsymbol{v}$ means a matrix and a vector respectively. It is made up of 6 different matrix operations, including M-M-Multi., M-V-Multi., V-V-Multi., and so on. In Figure 1 (a), we observe that the task's ES compared to its IOT increases linearly with the increase of the budgets assigned. This can be explained by Formula (3) or Equation (5). That is, higher budgets assigned will result in larger theoretically optimal resource amounts allocated, leading to a shorter IOT. However, the task has to be run atop the resources with limited capacities in practice, so the practical optimal time (POT) is actually also degraded correspondingly. We compare task's real execution length to its theoretically practical optical value by taking capacity into account.

Table I
WORKLOADS OF 10 MATRIX OPERATIONS (SINGLE-CORE EXECUTION LENGTH, MEASUREMENT UNIT: SECOND)

Matrix Scale	**M-M-Multi.**	**QR-Decom.**	**Matrix-Power**		**M-V-Multi.**	**Frob.-Norm**	**Rank**	**Solve**	**Solve-Tran.**	**V-V-Multi.**	**Two-Norm**
500	0.7	2.6	m=10	2.1	0.001	0.010	1.6	0.175	0.94	0.014	1.7
1000	11	12.7	m=20	55	0.003	0.011	8.9	1.25	7.25	0.021	9.55
1500	38	35.7	m=20	193.3	0.005	0.03	29.9	4.43	24.6	0.047	29.4
2000	99.3	78.8	m=10	396	0.006	0.043	67.8	10.2	57.2	0.097	68.2
2500	201	99.5	m=20	1015	0.017	0.111	132.6	18.7	109	0.141	136.6

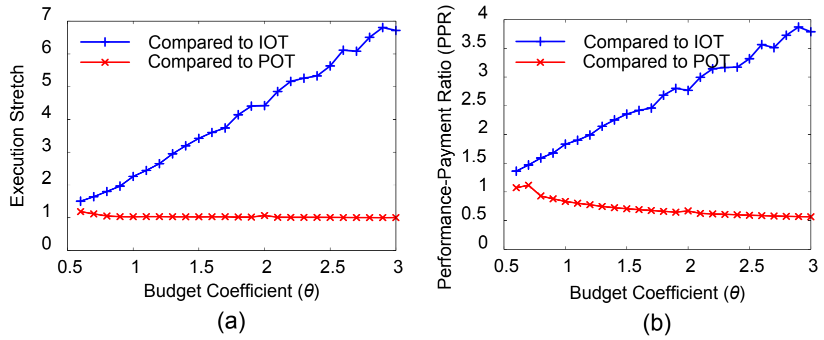

Figure 1. Execution Performance under Different Payments. (a) Execution Stretch. (b) Performance-Payment Ratio

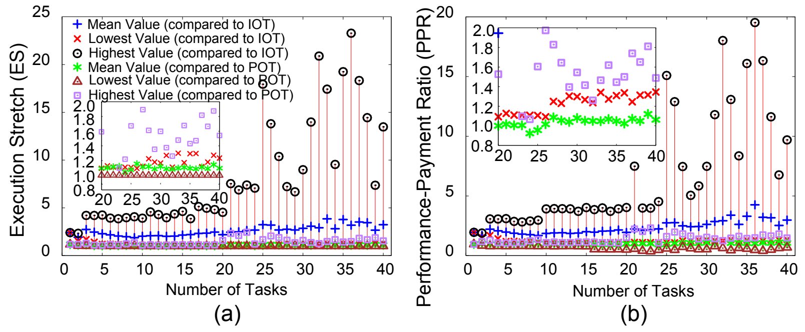

Figure 2. Execution Performance with Different Number of Tasks. (a) Execution Stretch. (b) Performance-Payment Ratio

Figure 1 (a) shows that the average ES is always less than 1.2 in this situation, which confirms that our solution is indeed able to optimize task's performance. In addition, the similar observation goes to the PPR metric, as shown in Figure 1 (b), confirming the payment should also be satisfied by users.

We also compare the results in the situation with adequate resources and the short-supply situation with relatively inadequate resources respectively. Figure 2 (a) shows the mean/lowest/highest values of the ES compared to IOT and POT respectively. When only a few tasks (e.g. $\leq$10) are submitted, the mean ES in both situations is always below 1.1, while the highest value of ES (worst situation) is up to 4. This is reasonable based on the following explanation. Note that the computation workloads among some subtasks (i.e., basic matrix operations) are largely different (e.g., between M-M-Multi. and V-V-Multi.), thus the resource amounts derived based on convex-optimization could be quite different. This will make the ideal optimal resource fractions of heavily-loaded subtasks be much bigger than the resource capacities (8-cores for the multi-threaded programs or 1-core for the single-threaded programs), such that subtasks cannot run in its ideal optimal states.

With further increasing number of tasks (from 10 to 40), the ES compared to IOT would decrease notably, yet the ES compared to the POT still keeps pretty close to 1 under this situation (as observed in Figure 2 (a)). This is attributed to more and more tasks cannot be assigned with the ideal optimal resource vectors (i.e., $\boldsymbol{r}^{(*)}$) but only the practical optimal ones (i.e., $\boldsymbol{r}^{*}$) by considering the limitation of the resource capacities. In fact, at such a situation with relatively short resource supply, task's practical optimal performance would also be degraded correspondingly, and ES$\approx$POT means that the tasks under our resource allocation run as efficiently as the practical optimal state with the capacity limitation. The similar observation goes with the PPR metric, as shown in Figure 2 (b). When comparing to IOT, the PPR enhances with increasing number of tasks to process, yet it increases more slowly than that of ES shown in Figure 2 (a), because users' payments are correspondingly reduced with the smaller resource amounts allocated.

We finally analyze the fairness of task's resource allocation based on Jain's fairness index [20] (Equation (31), where x_i is either ES or PPR.

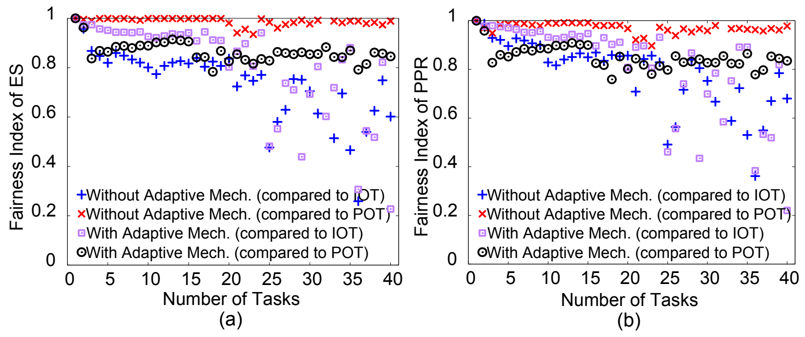

Figure 3. Fairness of Task's Execution Performance. (a) Fairness Index of ES. (b) Fairness Index of PPR.

$$F(\boldsymbol{x}) = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2} \tag{31}$$

The fairness of ES (PPR) which is compared to POT is much higher than the one that is compared to IOT, as shown in Figure 3 (a). This is because the basic matrix operations in our experiment are with largely different workloads (Table I), which could easily make the resource fractions assigned at different dimensions are quite uneven. That is, the resource amounts expected at some dimensions may be extremely huge, finally succeeding the corresponding resource capacities to different degrees. Then, the degradation of the practical execution compared to IOT could be very arbitrary. However, we could still observe quite stable and highly fair treatment on task's resource allocation w.r.t. POT, i.e., when comparing to the execution length to the practical optimal state considering the capacity limitation.

VI. Related Work

Cloud resource allocation problem has been extensively studied for years, however, most of the existing work overlooks the practical issue with possible erroneous workload predictions. Usiao et al. [21] proposed a distributed load rebalancing method for distributed file systems in Clouds. Unlike the file system where data size is relatively easy to predict precisely, we have to deal with erroneous prediction issue in our computational cloud platform with multiple execution dimensions. PACORA [22] is a performance-aware convex optimization model for resource allocation problem, assuming workload information could be known precisely in advance. Goudarzi et al. [23] proposed a multi-dimensional SLA-based resource allocation for multi-tier applications, with the assumption that the average of user request and resource power are pre-known exactly. Jalaparti et al. [24] aims to optimize the resource allocation utilities between any two clients or between client and provider. Their solutions have a strong assumption that the resource capacities are always large enough, while in our model, limited resource capacity is a key constraint, leading to a huge challenge especially in the bound analysis with prediction errors. Meng et al. [25] explicitly endeavored to maximize resource utilization by analyzing VM-pairs' compatibility in terms of the forecasted workload and estimated VM sizes. Their solution is able to approximately identify the compatibility of any pair of two VMs, but cannot resolve the situation with more than two VMs on the same machine. Wei et al. [26] formulated the Cloud resource allocation to be a binary Integer programming problem and solved it using an evolutionary method. A strong assumption in their work is the precise prediction of task's workload vector on multiple execution dimensions.

In order to provide guaranteed service-level agreement (SLA), it is crucial to analyze the possible situation with inaccurately predicted information. A few works (not many) also analyzed this issue for their approaches in the context of Cloud platforms, but differ a lot from this paper. Mao's auto-scaling method [27] and Di's approach [28] took into account load prediction issue in Cloud systems, whereas they both handled a different objective that aims to minimize user payment with guaranteed task deadlines. Thus, the problem formulation is fairly distinct, so is the following solution. Wood et al. [29] adopted black-box and gray-box strategies for virtual machine migration, in order to alleviate hot spots based on statistical analysis. However, statistical analysis cannot be used to derive the bound of task execution performance at the worst case. In [30], AuYoung et al. evaluated the impact of inaccurate prediction for various utility-based scheduling approaches. They make use of simulation to analyze the working efficiency of First-Come-First-Serve (FCFS) scheduler and backfilling scheduler [31] with possible skewed estimate of application utility function and resource ability state. Although simulation work could confirm the fault tolerance ability to a certain extent, it cannot prove its effectiveness fundamentally.

VII. Conclusion and Future Work

In this paper, a novel algorithm (namely ODRA) which aims to minimize task execution length under a budget with possible prediction errors is proposed and analyzed in depth. We carefully derived the upper bound of task execution length for a practical situation with erroneous prediction of task workloads. The state-of-the-art derived bound of task execution length is very concise, such that it can be easily applied in practice. To the best of our knowledge, this is the first paper to optimize the divisible-resource allocation with in-depth analysis on upper bound of task execution length with prediction errors. We evaluate the performance using a real cluster environment with composite web services. These services are of different execution patterns on multiple types of resources. Experiments show that compared to the idea execution length, task's average execution stretch is less than 1.1 when submitting small number of tasks, and less than 1.2 when over-many tasks are submitted simultaneously. That is, task execution lengths with our solution are fairly close to their theoretically optimal results. In the future, we plan to implement more practical web services for users under our optimal algorithm.

Acknowledgments

This research is supported by ANR project Clouds@home (ANR-09-JCJC-0056-01), ANR RESCUE 5323, and also in part by a Hong Kong RGC Grant HKU716712E.

References

[1] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journ.al of the ACM (JACM)*, vol. 24, pp. 280–289, April 1977.

[2] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," *Proc. third IEEE International Conference on Cloud Computing (Cloud'10)*, pp. 418–425, 2010.

[3] C. Jiang, C. Wang, X. Liu, and Y. Zhao, "A survey of job scheduling in grids," *Proc. of the joint nineth Asia-Pacific web and eighth int'l conf. on web-age information management conf. on Advances in data and web management (APWeb/WAIM'07)*, pp. 419–427, 2007

[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. seventh ACM/IFIP/USENIX Int'l Conf. on 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. ACM workshop on Experimental computer science (ExpCS'07)*, 2007.

[7] S. Chinni and R. Hiremane, "Virtual machine device queues," Virtualization Technology White Paper, Tech. Rep., 2007.

[8] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi, "Providing performance guarantees to virtual machines using real-time scheduling," *Proc. fifth aCM Workshop on Virtualization in High-Performance Cloud Computing (VHPC'10)*, 2010.

[9] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds : Managing performance interference effects for qos-aware clouds," *Proc. ACM European Conf. on Comp. Sys. (EuroSys'10)*, pp. 237–250, 2010.

[10] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2009.

[11] 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 Annual Conf. on Neural Information Processing Systems (NIPS'10)*, pp. 1–9, 2010.

[12] Xen-credit-scheduler: on line at http://wiki.xensource.com/xen wiki/creditscheduler.

[13] W. Almesberger, "Linux network traffic control – implementation overview," 1999. [Online]. Available: http://diffserv.sourceforge.net/

[14] S. Venugopal, J. Broberg, and R. Buyya, "OpenPEX: An Open Provisioning and EXecution System for Virtual Machines," Tech. Rep., CLOUDS-TR-2009-8, CLOUDS Laboratory, The University of Melbourne, Australia, Aug. 25, 2009.

[15] AWS economics: online at http://aws.amazon.com/economics.

[16] S. Di and C.-L. Wang, "Dynamic optimization of multi-attribute resource allocation in self-organizing clouds," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 24, no. 3, pp. 467–478, 2013.

[17] Gideon-II Cluster: http://i.cs.hku.hk/∼clwang/Gideon-II.

[18] 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. nineteenth ACM symp. on Operating systems principles (SOSP'03)*, pp. 164–177, 2003.

[19] P. Wendykier and J. G. Nagy, "Parallel colt: A high-performance java library for scientific computing and image processing," *ACM Trans. Math. Softw.*, vol. 37, pp. 31:1–31:22, September 2010.

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

[21] H. Hsiao, H. Su, H. Shen and Y. Chao, "Load Rebalancing for Distributed File Systems in Clouds," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2012.

[22] L. S. Bird and J. B. Smith, "Pacora: Performance aware convex optimization for resource allocation," *Proc. ACM third USENIX Workshop on Hot Topics in Parallelisation (HotPar'11) (Poster)*, 2011.

[23] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for cloud computing systems," *Proc. first Int'l workshop on data center performance (DCPerf'11), held in conjunction with ICDCS'2011*, 2011.

[24] V. Jalaparti, G. Nguyen, G. Indranil, and C. Matthew, "Cloud resource allocation games," University of Illinois, Tech. Rep., 2010, Available: http://hdl.handle.net/2142/17427.

[25] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," *Proc. of 7th IEEE International conf. on Autonomic computing (ICAC'10)*, pp. 11–20, 2010.

[26] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *J. Supercomputing*, vol. 54, no. 2, pp. 252–269, 2009.

[27] M. Mao and M. Humphrey, "Auto-Scaling to Minimize Cost and Meet ApplicationDeadlines in Cloud Workflows," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 49:1–49:12

[28] S. Di and C.-L. Wang, "Error-tolerant Resource Allocation and Payment Minimization for Cloud System," *Trans. on Parallel and Distributed Systems (TPDS)*, 2012.

[29] T. Wood, P.J. Shenoy, A. Venkataramani, and M.S. Yousif, "Blackbox and Gray-box strategies for virtual machine migration," in *Proc. of the 4th USENIX conference on Networked systems design and implementation (NSDI'07)*, pp. 17–31, 2007.

[30] A. AuYoung, A. Vahdat, and A. C. Snoeren, "Evaluating the impact of inaccurate information in utility-based scheduling," *Proc. of ACM Conf. on High Performance Computing Networking, Storage and Analysis (SC'09)*, pp. 38:1–38:12, 2009.

[31] D. A. Lifka, "The anl/ibm sp scheduling system," *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 295–303, 1995.