Dynamic Optimization of Multiattribute Resource Allocation in Self-Organizing Clouds

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

Abstract—By leveraging virtual machine (VM) technology which provides performance and fault isolation, cloud resources can be provisioned on demand in a fine grained, multiplexed manner rather than in monolithic pieces. By integrating volunteer computing into cloud architectures, we envision a gigantic self-organizing cloud (SOC) being formed to reap the huge potential of untapped commodity computing power over the Internet. Toward this new architecture where each participant may autonomously act as both resource consumer and provider, we propose a fully distributed, VM-multiplexing resource allocation scheme to manage decentralized resources. Our approach not only achieves maximized resource utilization using the proportional share model (PSM), but also delivers provably and adaptively optimal execution efficiency. We also design a novel multiattribute range query protocol for locating qualified nodes. Contrary to existing solutions which often generate bulky messages per request, our protocol produces only one lightweight query message per task on the Content Addressable Network (CAN). It works effectively to find for each task its qualified resources under a randomized policy that mitigates the contention among requesters. We show the SOC with our optimized algorithms can make an improvement by 15-60 percent in system throughput than a P2P Grid model. Our solution also exhibits fairly high adaptability in a dynamic node-churning environment.

Index Terms—Cloud computing, VM-multiplexing resource allocation, convex optimization, P2P multiattribute range query

1 Introduction

Cloud computing has emerged as a compelling paradigm for deploying distributed services. Resource allocation problem in cloud systems emphasizes how to harness the multiattribute resources by multiplexing operating systems. With virtual machine (VM) technology [1], we are able to multiplex several operating systems on the same hardware and allow task execution over its VM substrates without performance interference. Fine-grained resource sharing can be achieved as each VM substrate can be configured with proper shares of resources (such as CPU, memory, storage, network bandwidth) dynamically.

In recent years, various enhancements on resource isolation techniques [2], [13], [8] have been proposed to achieve fine-grained dynamic resource provisioning. A proportional share scheduler can be implemented based on Xen's *credit scheduler* [14] to multiplex CPU resource among virtual machines in a fair manner. The *balloon driver* [15], *difference engine* [10], *joint-VM* [8], and *virtual putty* [9], can dynamically adjust the memory resource among colocated virtual machines. The *dm-ioband* [16] can dynamically control the usage of disk I/O bandwidth among colocated virtual machines. These advanced techniques enable computing resources to be dynamically partitioned or reassembled to meet the elastic needs of end users. Such solutions create an unprecedented opportunity to maximize resource utilization, which were not possibly applied in most Grid systems [34], [36], [19], [37], [38] that usually treat the underlying resources as indivisible ones and prevent simultaneous access to them.

Today's cloud architectures are not without problems. Most cloud services built on top of a centralized architecture may suffer denial-of-service (DoS) attacks [3], unexpected outages, and limited pooling of computational resources. On the contrary, volunteer computing systems (or Desktop Grids) can easily aggregate huge potential computing power to tackle grand challenge science problems [4]. In view of this, we propose a novel cloud architecture, namely *self-organizing cloud* (SOC), which can connect a large number of desktop computers on the Internet by a P2P network. In SOC, each participating computer acts as both a resource provider and a resource consumer. They operate autonomously for locating nodes with more abundant resource or unique services in the network to offload some of their tasks, meanwhile they could construct multiple VM instances for executing tasks submitted from others whenever they have idle resources.

We focus on two key issues in the design of SOC: 1) the multiattribute range query problem in a fully decentralized environment for locating a qualified node to satisfy a user task's resource demand with bounded delay and 2) how to optimize a task's execution time by determining the optimal shares of the multiattribute resources to allocate to the tasks with various QoS constraints, such as the expected execution time and limited budget.

As a fundamental difference to existing approaches, we formulate such a resource allocation problem to be a convex optimization problem [23]. Given a task with its resource requirements and a budget, we first prove that the optimal resource allocation on a qualified node that can minimize a

- *S. Di is with the MESCAL Group, INRIA, Grenoble, Room 213, Laboratoire LIG, ENSIMAG, antenne de Montbonnot, Zirst, 51, avenue Jean Kuntzmann, 38330 Montbonnot 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 24 Sept. 2010; revised 2 Apr. 2012; accepted 11 Apr. 2012; published online 14 May 2012.
Recommended for acceptance by L.E. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-09-0722.
Digital Object Identifier no. 10.1109/TPDS.2012.144.

task's execution time does exist. We further show that it is nontrivial to solve such a convex optimization problem directly via a brute-force strategy and the interior point method [23]. By relaxing the problem definition, we propose an algorithm to optimize the task execution time on a qualified resource node, given its preset budget and tolerable quality of service (QoS). The proposed algorithm involves only $O(R^2)$ adjustment steps, where R denotes the number of resource attributes (or dimensions). We further propose a *dynamic optimal proportional-share* (DOPS) resource allocation algorithm with $O(R^3)$ complexity, by incorporating the *proportional-share model* (PSM) [12]. The key idea is to dynamically scale the amount of resources at each dimension among running tasks proportional to their demand, such that these tasks could use up the maximum capacity of each resource type at a node.

To locate qualified nodes in the SOC environment, we design a fully decentralized range query protocol, namely *pointer-gossiping CAN* (PG-CAN), tailored for DOPS. Existing P2P desktop Grids favor CAN-based [17] or Chord-based [18] resource discovery protocols [19], [20]. Every joining node registers its static resource attributes (e.g., CPU architecture, OS version) or maximum capacity on the CAN/Chord overlay, so that other users could find the most matched node within a logarithmic (or sublinear) number of routing steps. Such a design is feasible for a P2P desktop Grid because the resources of a selected node can only be used exclusively by a single task. However, due to dynamic resource provisioning technologies used in cloud, the frequent resource repartitioning and reallocation (e.g., upon task arrival or completion) make it a challenging problem to locate a node containing a combination of available resources along all the R resource attributes that would satisfy the requirements of a submitted task.

The proposed PG-CAN range query protocol in this work aims to find the qualified resources with minimized contention among requesters based on task's demand. It is unique in that for each task, there is only one query message propagated in the network during the entire course of discovery. This is different from most existing multiattribute range query solutions that require to propagate multiple subqueries along multiple dimensions in parallel. To mitigate the contention problem due to analogous queries in CAN, our range query protocol proactively diffuses resource indexes over the network and randomly route query messages among nodes to locate qualified ones that satisfy tasks' minimal demands. To avoid possibly uneven load distribution and abrupt resource overutilization caused by uncoordinated node selection process from autonomous participants, we investigate three node selection policies, namely double-check policy [21], queuing policy [22], and extra-virtual-dimension (VD) policy [19].

The rest of the paper is organized as follows: We formulate the resource allocation problem in a VM-multiplexing environment in Section 2. In Section 3, we prove that optimal resource allocation does exist and show that our solution is optimal. In Section 4, we present our DOPS resource allocation scheme. Section 5 details the proposed range query protocol. In Section 6, we show the simulation results. We discuss related work in Section 7 and conclude with an outline of future work in Section 8.

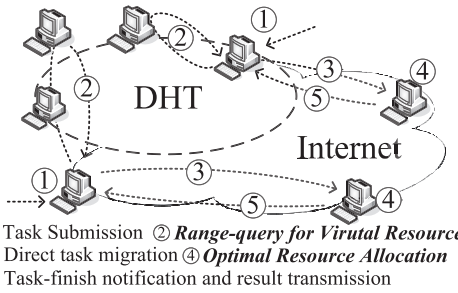


Fig. 1. The entire task execution procedure.

2 Problem Formulation

Fig. 1 shows the entire journey of a task from its submission to completion over the SOC system. In this work, we only focus on the multiattribute range query problem (Step 2) and the resource allocation problem for determining the amount of resources of a qualified node to the submitted task (Step 4).

Suppose there are n nodes in SOC, each is denoted as p_i, where $1 \leq i \leq n$. Each node owns R different resources (or resource attributes)[1] managed by a Virtual Machine Monitor (VMM). We denote Π to be the set of resource attributes owned by node p_i and $\boldsymbol{c}(p_i) = (c_1(p_i), c_2(p_i), \ldots, c_R(p_i))^T$ as p_i's capacity vector. For example, if a computer owns a 2.4 Gflops single-core CPU, 1 GB memory, and a 10 Mbps network bandwidth, its capacity vector is $(2.4, 1, 10)^T$.

Let m_i denote the total number of tasks submitted to p_i. A task submitted to node p_i is denoted as t_{ij}, where $1 \leq j \leq m_i$. Each task is associated with an *expected resource vector* $\boldsymbol{e}(t_{ij}) = (e_1(t_{ij}), e_2(t_{ij}), \ldots, e_R(t_{ij}))^T$. The user-specified *expected resource vector* is a rough estimation of the needed amount of resources with respect to the R resource attributes for a submitted task to be completed within a tolerable execution time. After t_{ij} gets scheduled, we denote its actual allocated resource as $\boldsymbol{r}(t_{ij}) = (r_1(t_{ij}), r_2(t_{ij}), \ldots, r_R(t_{ij}))^T$, where $\boldsymbol{r}(t_{ij}) \succeq \boldsymbol{e}(t_{ij})$. Here, $\succeq$ means the componentwise inequality between two vectors. For short, we denote $\boldsymbol{r}(t_{ij})$ and $r_k(t_{ij})$ as $\boldsymbol{r}$ and r_k, respectively, in the case without causing ambiguity.

Each task has a *load vector*, denoted as $\boldsymbol{l}(t_{ij}) = (l_1(t_{ij}), l_2(t_{ij}), \ldots, l_R(t_{ij}))^T$, indicating the amount of workload on each of the R resource attributes for completing the task. For simplicity, we assume the execution of a task cannot be done concurrently among different resources at the same node. Hence, if a task t_{ij} is executed at p_i, its execution time is equal to $\boldsymbol{l}(t_{ij})^T \cdot \boldsymbol{r}(t_{ij})^{-1}$, where $\boldsymbol{r}(t_{ij})^{-1} = (r_1^{-1}(t_{ij}), r_2^{-1}(t_{ij}), \ldots, r_R^{-1}(t_{ij}))^T$.

We define a *preferential weight vector* $\boldsymbol{w}(t_{ij}) = (w_1(t_{ij}), w_2(t_{ij}), \ldots, w_R(t_{ij}))^T$, satisfying $w_1(t_{ij}) : w_2(t_{ij}) : \cdots : w_R(t_{ij}) = l_1(t_{ij}) : l_2(t_{ij}) : \cdots : l_R(t_{ij})$, which indicates the relative importance of a resource that might affect the execution time of a task according to its property (e.g., CPU bound or IO bound). In essence, $\boldsymbol{w}(t_{ij})$ acts as a more relaxed requirement in using our model as we assume the user does not know the exact load vector, but only needs to specify the preferential weight vector.

1. The R resource attributes can also be viewed as R dimensions, so we use attributes and dimensions interchangeably in the following text.

TABLE 1
Key Notations Used in SOC Model

Notation	Description
n	number of nodes
p_i	a node, where i = 1,2,$\cdots$,n
$\boldsymbol{c}(p_i)$	capacity vector of node p_i
$\boldsymbol{l}(p_i)$	total workload vector of the tasks running on p_i
$\boldsymbol{a}(p_i)$	the availability vector of p_i =$(a_1(p_d), a_2(p_d), \cdots, a_R(p_d))^T$
$\boldsymbol{b}(p_i)$	the price vector of p_i's resources on multiple dimensions
t_{ij}	the jth task submitted to p_i
$\boldsymbol{l}(t_{ij})$	the workload vector of task t_{ij}
$\boldsymbol{e}(t_{ij})$	expected resource vector of t_{ij}
$\boldsymbol{r}(t_{ij})$	the resource share vector allocated to t_{ij}
$\boldsymbol{r}^*(t_{ij})$	t_{ij}'s optimal resource share vector with availability constraint
$\boldsymbol{r}^{(*)}(t_{ij})$	t_{ij}'s optimal resource share vector without availability constraint
$\boldsymbol{r}^{-1}(t_{ij})$	=$(r_1^{-1}(t_{ij}), r_2^{-1}(t_{ij}), \cdots, r_R^{-1}(t_{ij}))^T$
$\boldsymbol{w}(t_{ij})$	the weight vector of task t_{ij}, defined in Formula (4)
$B(t_{ij})$	budget of t_{ij}'s user (evaluated by per-time-unit)
$\rho(t_{ij}, \Delta t)$	user's payment in executing t_{ij}, where Δt is execution time

To mimic the pricing scheme in a real-world cloud, we follow a simple monetary model to analyze the economic implications between consumers and providers. For any node p_i, its *resource price vector* is denoted as $\boldsymbol{b}(p_i) = (b_1(p_i), b_2(p_i), \ldots, b_R(p_i))^T$, which is designated by the resource provider. Let $b_k(p_i)(1 \le k \le R)$ represent the per-time-unit price for using the kth resource attribute on p_i. Thus, to run a task t_{ij} on node p_s, the total payment is calculated as (1), where Δt is the execution time of t_{ij} on p_s

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

In our model, the user is "satisfactory" if the per-time-unit rate, i.e., the total of per-time-unit cost for using the R resource attributes at a selected node (denoted as $B(t_{ij})$), is always in accordance with Inequality (3). We argue that users can hardly predict their tasks' exact execution times in practice. However, the users would still feel worthy as long as the per-time-unit rate is still within their budget.

Given a submitted task t_{ij} with designated $\boldsymbol{e}(t_{ij})$ and $\boldsymbol{w}(t_{ij})$, this work investigates two key issues:

1. How to efficiently locate a qualified node in a large-scale peer-to-peer network for executing the submitted task with controllable message delivery overhead.
2. How to optimize t_{ij}'s execution time (i.e., (2)) by determining the optimal amount of resources (i.e., $\boldsymbol{r}^{(*)}$) to allocate to t_{ij}, subject to the constraints (3), (4), and (5), where node p_s's *available resource vector* $\boldsymbol{a}(p_s) = \boldsymbol{c}(p_s) - \sum_{t_{ij}\ on\ p_s} \boldsymbol{r}(t_{ij})$

$$Min \quad f(\boldsymbol{r}(t_{ij})) = \boldsymbol{l}^T(t_{ij}) \cdot \boldsymbol{r}^{-1}(t_{ij}) = \sum_{k=1}^{R} \frac{l_k}{r_k}, \tag{2}$$

s.t.

$$\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}(t_{ij}) \le B(t_{ij}), \tag{3}$$

$$w_1(t_{ij}) : \cdots : w_R(t_{ij}) = l_1(t_{ij}) : \cdots : l_R(t_{ij}), \tag{4}$$

$$\boldsymbol{e}(t_{ij}) \preceq \boldsymbol{r}(t_{ij}) \preceq \boldsymbol{a}(p_s). \tag{5}$$

We summarize the key notations in Table 1. In the following text, we might omit the symbols t_{ij} and p_s for simplicity if thus would not lead to the ambiguity. For example, the notations $l_k(t_{ij}), e_k(t_{ij}), \boldsymbol{r}(t_{ij}), b_k(p_s), B(t_{ij})$, and $\boldsymbol{a}(p_s)$ may be substituted by $l_k, e_k, \boldsymbol{r}, b_k, B$, and $\boldsymbol{a}$, respectively, in some following expressions or formulas.

3 Optimal Resource Allocation

Given a task t_{ij} with its weight vector $\boldsymbol{w}(t_{ij})$ and a budget $B(t_{ij})$, we first prove that the optimal resource allocation on a qualified node p_s with its price vector $\boldsymbol{b}(p_s)$ does exist.

Lemma 1. *The optimal allocation (denoted $\boldsymbol{r}^*(t_{ij})$) exists iff (i.e., $\Longleftrightarrow$) Inequalities (6) and (7) are met*

$$\boldsymbol{b}(p_s)^T \cdot \boldsymbol{e}(t_{ij}) \le B(t_{ij}), \tag{6}$$

$$\boldsymbol{e}(t_{ij}) \preceq \boldsymbol{a}(p_s). \tag{7}$$

Proof. To prove $\Rightarrow$: (transport property of inequalities)
If $r^*(t_{ij})$ exists, it must satisfy Inequalities (3) and (5), thus the Inequalities (6) and (7) should hold.

To prove $\Leftarrow$: (to satisfy *Slater*'s condition [23])
If $\boldsymbol{b}(p_s)^T \cdot \mathbf{e}(\mathbf{t_{ij}}) = \mathbf{B}(\mathbf{t_{ij}})$ or $\mathbf{e}(\mathbf{t_{ij}}) = \boldsymbol{a}(p_s)$, then $\mathbf{e}(\mathbf{t_{ij}})$ is a unique solution, which can be regarded as an optimal one. If $\boldsymbol{b}(p_s)^T \cdot \mathbf{e}(\mathbf{t_{ij}}) < \mathbf{B}(\mathbf{t_{ij}})$ and $\mathbf{e}(\mathbf{t_{ij}}) \prec \boldsymbol{a}(p_s)$, other than $\boldsymbol{e}(t_{ij})$, there must exist another better solution (denoted $\boldsymbol{r}'(t_{ij})$) such that $\boldsymbol{b}(p_s)^T \cdot \boldsymbol{r}'(t_{ij}) < B(t_{ij})$ and $\mathbf{e}(\mathbf{t_{ij}}) \prec \boldsymbol{r}'(t_{ij}) \prec \boldsymbol{a}(p_s)$, thus $\boldsymbol{r}^*(t_{ij})$ must exist according to *Slater*'s condition [23]. Similarly, if $\boldsymbol{b}(p_s)^T \cdot \mathbf{e}(\mathbf{t_{ij}}) < \mathbf{B}(\mathbf{t_{ij}})$ and $\mathbf{e}(\mathbf{t_{ij}}) \preceq \mathbf{a}(\mathbf{p_s})$, *Slater*'s condition can also hold by excluding the equations from the constraints (7). □

We assume the qualified node p_s that satisfies Inequalities (6) and (7) can be found based on t_{ij}'s expected resource vector $\mathbf{e}(\mathbf{t_{ij}})$ by a resource discovery protocol (to be discussed in Section 5). Thus, we could rewrite the constraint (5) to be Inequality (8) and construct a Lagrangian function $F_1(\boldsymbol{r}(t_{ij}))$ (i.e., (9)), where λ and $\nu_1, \nu_2, \ldots, \nu_R$ are the Lagrangian multipliers

$$(r_k - e_k)(r_k - a_k) \le 0, k = 1, 2, \ldots, R, \tag{8}$$

$$F_1(\boldsymbol{r}) = \sum_{k=1}^{R} \frac{l_k}{r_k} + \lambda \left(\sum_{k=1}^{R} b_k r_k - B \right) + \sum_{k=1}^{R} \nu_k (r_k - e_k)(r_k - a_k). \tag{9}$$

As analyzed previously, the optimal resource allocation ($\boldsymbol{r}^*$) does exist. Therefore, the optimal solution must satisfy Karush-Kuhn-Tucker (KKT) conditions [23], listed below

$$\begin{cases} \boldsymbol{b}^T \cdot \boldsymbol{r}^* \le B \\ \left(r_k^* - e_k\right)\left(r_k^* - a_k\right) \le 0 & k = 1, 2, \ldots, R \\ \lambda \ge 0, \nu \succeq 0 \\ \lambda \cdot (\boldsymbol{b}^T \cdot \boldsymbol{r}^* - B) = 0 \\ \nu_k \cdot \left(r_k^* - e_k\right)\left(r_k^* - a_k\right) = 0 & k = 1, 2, \ldots, R \\ -\dfrac{l_k}{r_k^{*2}} + \lambda \cdot b_k + \nu_k \left(2r_k^* - e_k - a_k\right) = 0 & \\ & k = 1, 2, \ldots, R. \end{cases} \tag{10}$$

That is, the optimal resource vector $\boldsymbol{r}^*$ could be found as long as we could satisfy the above conditions simultaneously. In order to solve the above simultaneous equations

and inequalities, there are two traditional candidate strategies: 1) the brute-force method and 2) the interior point method. Based on the brute-force method, we can first focus on the fifth formula, which involves R equations $(\nu_k \cdot (r_k^* - e_k)(r_k^* - a_k) = 0$, where $k = 1, 2, \ldots, R)$ and $2R$ variables $(r_1^*, r_2^*, \ldots, r_R^*, \nu_1, \nu_2, \ldots, \nu_R)$. Each equation (e.g., $\nu_k \cdot (r_k^* - e_k)(r_k^* - a_k) = 0$) holds, if and only if one of the three conditions is valid, i.e., $\nu_k = 0, r_k = e_k$, or $r_k = a_k$. For those combinations with their Lagrangian multipliers being equal to zero (e.g., $\nu_k = 0$), we still need to calculate their resource assignments (i.e., r_k) based on the fourth formula and the sixth formula. For all of the R equations $(\nu_k \cdot (r_k^* - e_k)(r_k^* - a_k) = 0$, where $k = 1, 2, \ldots, R)$, there are totally 3^R combinations that could make them simultaneously hold, so the overall time complexity is $O(3^R)$, which is intolerable with a large number of dimensions. Based on the interior point method (or *Newton*'s method), we need to guess a set of initial values for the resource vector $\mathbf{r}$ and try to guarantee the method is converged with them, which is complex especially because of a large number $(2R + 1)$ of variables, and its computation result will only be an approximate value. Consequently, this problem seems unsolvable based on the two traditional methods.

By revisiting Constraint (5), it is clear that $\boldsymbol{a}(t_{ij})$ is a "firm" bound while $\boldsymbol{e}(t_{ij})$ is a "soft" bound. That is, $\mathbf{r}(\mathbf{t_{ij}})$ cannot be greater than $\boldsymbol{a}(t_{ij})$ anyhow due to the limited resource capacity. But it can be lower than $\mathbf{e}(\mathbf{t_{ij}})$ because $\mathbf{e}(\mathbf{t_{ij}})$ was not a strict limitation but just estimated by users. If we replace Constraint (5) with Constraint (11), we could find an optimal solution through a few convex optimization steps. That is, via such a constraint relaxation, we could optimize the resource allocation for task t_{ij} on node p_s without exhausting all 3^R possible combinations like the brute-force method or worrying about the convergence problem in the interior point method

$$\boldsymbol{r}(t_{ij}) \preceq \boldsymbol{a}(p_s). \tag{11}$$

In the following text, we discuss the situation without Constraint (11) via convex optimization analysis, and then derive the optimal algorithm for the case with the constraint.

Theorem 1. *In order to minimize $f(\boldsymbol{r}(t_{ij}))$ subject to Constraints (3) and (4), t_{ij}'s optimal resource vector $\mathbf{r}^{(*)}(\mathbf{t_{ij}})$ is shown as (12), where $k = 1, 2, \ldots, R$. Note that $\boldsymbol{r}^{(*)}(t_{ij})$ is not subject to Inequality (5) or (11), unlike the notation $\boldsymbol{r}^*(t_{ij})$*

$$r_k^{(*)}(t_{ij}) = \frac{\sqrt{w_k(t_{ij})/b_k(p_s)}}{\sum_{k=1}^{R} \sqrt{w_k(t_{ij}) b_k(p_s)}} \cdot B(t_{ij}). \tag{12}$$

Proof. We first prove that the target function $f(\boldsymbol{r})$ is convex, then find the optimal $\boldsymbol{r}^{(*)}$ via convex optimization.

Since $\frac{\partial^2 f(r)}{\partial r_k} = 2\frac{l_k}{r_k^3} > 0$, $f(\boldsymbol{r})$ is convex with a minimum extreme point. Then, the target *Lagrangian* function can be defined as (13) and λ is the Lagrangian multiplier

$$F_2(\boldsymbol{r}) = \frac{l_1}{r_1} + \cdots + \frac{l_R}{r_R} + \lambda(b_1 r_1 + \cdots + b_R r_R - B). \tag{13}$$

Let $\frac{\partial F_2(r)}{\partial r_k} = 0$, then we could get (14), where $k = 1, 2, \ldots, R$. Accordingly, we can get (15)

$$l_k / r_k^2 = \lambda b_k, \tag{14}$$

$$r_1 : r_2 : \cdots : r_R = \sqrt{\frac{l_1}{b_1}} : \sqrt{\frac{l_2}{b_2}} : \cdots : \sqrt{\frac{l_R}{b_R}}. \tag{15}$$

In order to minimize $f(\boldsymbol{r})$, the optimal resource vector $\boldsymbol{r}^{(*)}$ should satisfy $\boldsymbol{b}^T \cdot \boldsymbol{r} = B$. By combining this equation with (4) and (15), we could get (12). □

Remark. 1) Based on (15), the relative amount of resource (r_k) to be allocated to a task is determined by $\sqrt{\frac{l_k}{b_k}}$. As long as the users and resource providers express their workload (l_k) and price (b_k) based on the same units of measurement (e.g., Gflops or Gflops/$), the ratio remains unchanged. 2) Formula (12) can be used to derive the resource vector $\boldsymbol{r}^{(*)}$ for t_{ij} such that its execution time can be minimized within a budget limit (i.e., (3)). Based on (12) and the proof of Theorem 1, $\boldsymbol{r}^{(*)}$ can be easily computed in O(R) time. 3) As mentioned, $\boldsymbol{r}^{(*)}(t_{ij})$ is not subject to Inequality (5) or (11), but can be regarded as an optimal resource allocation as long as it satisfies Constraint (11), i.e., $\forall k$: $r_k^{(*)} \le a_k(p_s)$. However, if $\boldsymbol{r}^{(*)}$ does not fully satisfy Constraint (11) (i.e., $\exists k$: $r_k^{(*)} > a_k(p_s)$), $\boldsymbol{r}^{(*)}$ is not a feasible solution. Therefore, we propose Algorithm 1 with a provable time complexity $O(R^2)$ to enforce Constraint (11), while minimizing $f(\boldsymbol{r}(t_{ij}))$ under the two additional constraints (3) and (4).

Definition 1 (CO-STEP (Γ, C)). *Let Γ denote a subset of resource attributes Π (i.e., $\Gamma \subseteq \Pi$), while $\boldsymbol{r}_\Gamma(t_{ij})$ and $\boldsymbol{b}_\Gamma^T(p_s)$ denote the resource vector assigned to t_{ij} and the price vector specified by p_s w.r.t. Γ, respectively. Given a budget C, CO-STEP (Γ, C) is a procedure for computing the optimal resource vector for t_{ij} w.r.t. Γ, which minimizes $f(\boldsymbol{r}_\Gamma(t_{ij}))$, subject to Constraints (4) and (16) but excluding Constraint (11) by using convex optimization.*

$$\boldsymbol{b}_\Gamma^T(p_s) \cdot \boldsymbol{r}_\Gamma(t_{ij}) \le C, \ \textit{where } C \textit{ is a constant}. \tag{16}$$

We devise Algorithm 1 for minimizing $f(\boldsymbol{r}(t_{ij}))$ subject to Constraints (3), (4), and (11), as shown below.

Algorithm 1. COMPUTE OPTIMAL RESOURCE VECTOR $\boldsymbol{r}^*$
Input: $\Gamma, B(t_{ij}), \boldsymbol{b}(p_s), \boldsymbol{a}(p_s)$; **Output**: $\boldsymbol{r}^*$
1: $\Gamma = \Pi, C = B(t_{ij}), \boldsymbol{r}^* = \Phi$ (empty set);
2: **repeat**
3: $\quad \boldsymbol{r}_\Gamma^{(*)} =$ CO-STEP(Γ,C); /* Compute optimal $\boldsymbol{r}$ based on Γ*/
4: $\quad \Omega = \{d_k | d_k \in \Gamma$ & $r_k^{(*)} > a_k\}$; /*Select elements violating Constraint (11)*/
5: $\quad \Gamma = \Gamma \backslash \Omega$; /*Remove Ω from Γ*/
6: $\quad C = C - \sum_{d_k \in \Omega} (b_k \cdot a_k)$; /*Update C*/
7: $\quad \boldsymbol{r}^* = \boldsymbol{r}^* \cup \{r_k^* = a_k | d_k \in \Omega$ & a_k is d_k's upper bound$\}$;
8: **until** $(\Omega = \Phi)$;
9: $\boldsymbol{r}^* = \boldsymbol{r}^* \cup \boldsymbol{r}_\Gamma^{(*)}$;

In this algorithm, Line 3 executes CO-STEP(Γ, C) in order to find the $\boldsymbol{r}_\Gamma^{(*)}$ without considering Constraint (11). If $\boldsymbol{r}_\Gamma^{(*)}$ completely satisfies Constraint (11) (i.e., $\Omega = \Phi$), the final result is found. Since CO-STEP(Γ, C) excludes Constraint (11), there might be some cases which are assigned with resource amount larger than the availability (i.e., $r_k^{(*)} > a_k(p_s)$). For

these cases, we will assign the maximum available resources to them (i.e., $r_k^{(*)} = a_k$) and remove the corresponding resource attributes/types (i.e., Ω) away from Γ. The remaining budget (i.e., $C = C - \sum_{d_k \in \Omega}(b_k \cdot a_k)$) is also updated accordingly. The whole process will continue until Ω becomes empty. Since the time complexity of CO-STEP(Γ,C) is $O(|\Gamma|)$, the number of computation steps of Algorithm 1 in the worst case is $\sum_{i=0}^{R-1}(R-i)$. The time complexity of Algorithm 1 is $O(R^2)$.

Theorem 2. *Given a submitted task t_{ij} with its weight vector $\boldsymbol{w}(t_{ij})$ and a budget $B(t_{ij})$, and a qualified node p_s with its resource price vector $\boldsymbol{b}(p_s)$, Algorithm 1's output $\boldsymbol{r}^*$ is optimal for minimizing t_{ij}'s execution time (i.e., $f(\boldsymbol{r}(t_{ij}))$), subject to Constraints (3), (4), and (11).*

Proof. With Constraints (3), (4), and (11), this is a typical convex optimization problem with the Lagrangian function formulated as

$$F_3(\boldsymbol{r}) = \sum_{k=1}^{R} \frac{l_k}{r_k} + \lambda\left(\sum_{k=1}^{R} b_k r_k - B\right) + \sum_{k=1}^{R} \nu_k (r_k - a_k). \quad (17)$$

The main idea is to prove the output of Algorithm 1 must satisfy KKT conditions (i.e., the necessary and sufficient condition of the optimization), which are listed below

$$\begin{cases} \boldsymbol{b}^T \cdot \boldsymbol{r}^* \le B \\ r_k^* - a_k \le 0 & k = 1, 2, \ldots, R \\ \lambda \ge 0, \nu \succeq 0 \\ \lambda \cdot \left(\boldsymbol{b}^T \cdot \boldsymbol{r}^* - B\right) = 0 \\ \nu_k \cdot \left(r_k^* - a_k\right) = 0 & k = 1, 2, \ldots, R \\ -\dfrac{l_k}{r_k^{*2}} + \lambda \cdot b_k + \nu_k = 0 & k = 1, 2, \ldots, R. \end{cases} \quad (18)$$

Algorithm 1 starts with the execution of CO-STEP($\Pi, B(t_{ij})$), which returns a result of $\boldsymbol{r}_\Pi^{(*)}$. According to Definition 1 and Theorem 1, $\boldsymbol{r}_\Pi^{(*)}$ is deduced from Constraints (4) and (16), so $\boldsymbol{r}_\Pi^{(*)}$ must satisfy $-\frac{l_k}{r_k^2}\lambda \cdot b_k = 0$ and $\boldsymbol{b}^T \cdot \boldsymbol{r} = B$. Then, if we let $\nu_k = 0$ for any k, there must exist an assignment such that all conditions in (18) hold except for the second condition $r_k^{(*)} - a_k \le 0$. Accordingly, $\boldsymbol{r}^* = \boldsymbol{r}_\Pi^{(*)}$ as long as $r_k^{(*)} - a_k \le 0, \forall\ k$ in Π (i.e., $1 \le k \le R$).

If $\boldsymbol{r}_\Pi^{(*)}$ cannot satisfy all the R Inequalities (i.e., $\exists k$ in $\Pi, r_k^{(*)} - a_k > 0$), we need to further adjust the solution $\boldsymbol{r}_\Pi^{(*)}$ to find an assignment that satisfies all KKT conditions in the (18). Based on Algorithm 1, for those cases with $r_k^{(*)} > a_k, r_k^{(*)}$ will be set to a_k. Assuming there are h_1 such cases and they are denoted as $r_1, r_2, \ldots, r_{h_1}$. Obviously, each selected r_k must satisfy $\nu_k \cdot (r_k - a_k) = 0$ because $r_k = a_k$. On the other hand, Algorithm 1 will continue to execute CO-STEP(Γ, C) on the remaining $R - h_1$ undecided cases (denoted as Γ), with $C = B(t_{ij}) - \sum_{k=1}^{h_1} r_k$. Likewise, all the $R - h_1$ cases (each denoted by $r_k, k = h_1 + 1, \ldots, R$) must also satisfy $-\frac{l_k}{r_k^2} + \lambda \cdot b_k = 0$ and $\boldsymbol{b}^T \cdot \boldsymbol{r} = B$. Thus, if each of them meets the condition $r_k - a_k \le 0$, then the $R - h_1$ cases in Γ and the previously selected h_1 cases will together compose the solution satisfying all conditions in (18). If there are still h_2 $(0 < h_2 \le R - h_1)$ new cases violating $r_k - a_k \le 0$ in this round, Algorithm 1 will continue the adjustment until the Hth round that either all the $R - \sum_{i=1}^{H} h_i$ cases can satisfy $r_k - a_k \le 0$ or Γ becomes empty. In the former case, we can easily verify that resource allocation among all the R attributes satisfy all KKT conditions in (18) simultaneously, composing an optimal solution. For the latter, we could conclude $\sum_{k=1}^{R} b_k \cdot a_k \le B(t_{ij})$, then the optimal resource allocation is $\boldsymbol{r}^* = \boldsymbol{a}$. □

4 Optimal Proportional-Share Allocation

In this section, we discuss the design of our *dynamic optimal proportional-share* resource allocation method, which leverages the proportional share model. The key idea to redistribute available resources among running tasks dynamically, such that these tasks could use up the maximum capacity of each resource in a node (i.e., up to $\mathbf{c}(\mathbf{p_s})$), while each task's execution time can be further minimized in a fair way.

DOPS consists of two main procedures: 1) *Slice handler*: It is activated periodically to equally scale the amount of resources allocated to tasks, such that each running task can acquire additional resources proportional to their demand along each resource dimension. 2) *Event handler*: It is responsible for resource redistribution upon the events of task arrival and completion. The pseudocodes are shown in Algorithm 2 and Algorithm 3. The slice handler (Algorithm 2) is periodically performed by p_s's VMM, while the event handler (Algorithm 3) is only invoked upon task arrival or completion.

Algorithm 2. SLICE HANDLER (PSM)
This program is activated periodically.
1: **for** $(k = 1 \rightarrow R)$ **do**
2: $sum\_allocation = \sum_{p=1}^{M} r_{(p)k}^*$;
3: **for** (each $t_{(j)}, j = 1, 2, \ldots, M$) **do**
4: $r_{(j)k}^{**} = (r_{(j)k}^* r_{(j)k}^* / sum\_allocation) \cdot c_k$;
5: Assign $r_{(j)k}^{**}$ to $t_{(j)}$;
6: **end for**
7: **end for**
8: Notify VMM to readjust resource allocation among all running tasks;

Algorithm 3. EVENT HANDLER
This program is invoked as an event is generated.
1: **if** (The event is the arrival of a scheduled task $t_{(x)}$) **then**
2: $\boldsymbol{a}_{(x)} = \boldsymbol{c}(p_s) - \sum_{j=1}^{x-1} r_{(j)}^*$;
3: Conduct Algorithm 1 for $t_{(x)}$;
4: **end if**
5: **if** (The event is the completion of a task $t_{(y)}$) **then**
6: $\boldsymbol{a}(p_s) = \boldsymbol{a}(p_s) + \boldsymbol{r}_{(y)}^*$; /*Release $t_{(y)}$'s resource $\boldsymbol{r}_{(y)}^*$*/
7: **for** (each $t_{(i)}$ still running on p_s) **do**
8: **if** ($\exists\ k$ such that $r_{(i)k}^* < r_{(i)k}^{(*)}, k = 1, 2, \ldots, R$) **then**
9: Conduct Algorithm 1 for $t_{(i)}$;
10: **end if**
11: **end for**
12: **end if**

Suppose there are M tasks running on a particular node p_s, denoted as $t_{(1)}, t_{(2)} \ldots, t_{(M)}$ based on their arrival order. Accordingly, $\boldsymbol{w}_{(i)}, B_{(i)}$, and $\boldsymbol{r}_{(i)}$ denote $t_{(i)}$'s preferential

weight vector, budget, and resource vector, respectively. Assume that when $t_{(i)}$ is to be scheduled onto p_s, the available resource vector is denoted as $\boldsymbol{a}_{(i)}$, which can be componentwise calculated as $\mathbf{c}(\mathbf{p_s}) - \sum_{\mathbf{j=1}}^{\mathbf{i-1}} \mathbf{r}^*_{\mathbf{(j)}}$. Note that resource node p_s found by our discovery protocol (to be described in Section 5) is "qualified", i.e., for $t_{(i)}(\mathrm{i} = 1, 2, \ldots, M)\boldsymbol{b}^T(p_s) \cdot \boldsymbol{e}_{(i)} \leq B_{(i)}$ and $\boldsymbol{e}_{(i)} \preceq \boldsymbol{c}(p_s) - \sum_{j=1}^{i-1} \boldsymbol{r}^*_{(j)}$. Thus, according to *Lemma* 1, the optimal resource vector $\boldsymbol{r}^*_{(i)}$ must exist and $\boldsymbol{r}^*_{(i)} \geq \boldsymbol{e}_i$.

Based on the above analysis, it is possible that the resource along certain dimensions (say the kth) may not be fully used, provided that $r^*_{(M)k}$ is lower than $a_{(M)k}$. Consequently, we could improve the resource utilization by redistributing the remaining resource at the kth dimension. Algorithm 2 shows how to determine the amount of resources allocated to the M running tasks so as to make full use of the underlying resources along every dimension (i.e., up to $\boldsymbol{c}(p_s)$). By leveraging PSM, each running task can acquire its resources proportional to their computed optimal shares (i.e., $\boldsymbol{r}^*$) along every dimension (Line 4-5). Since $\boldsymbol{r}^{**}_{(i)} \succeq r^*_{(i)}$, task $t_{(i)}$ will be executed faster with the augmented resource, while the payment will still be calculated based on $\boldsymbol{r}^*$ instead of $\boldsymbol{r}^{**}$. This means the user will not be charged more even with any extra resource allocation. It is easy to prove that Algorithm 2's time complexity is $O(R \cdot M)$.

Another issue is how to determine the optimal resource allocation (i.e., $\boldsymbol{r}^*$) upon task arrival and completion. As shown in Algorithm 3, when a new task is scheduled onto the node p_s, it will get the optimal shares of resource based on the availability of resources at p_s (Line 1-4). Note that the calculation is based on notations of Algorithm 1, instead of the scaled resource (i.e., $\boldsymbol{r}^{**}$ derived from Algorithm 2) as the slice handler will be activated afterwards based on the DOPS design. When a task is finished, it is possible for other running tasks to share the newly released resources (Line 5-12). The time complexity is $M \cdot R^3$ according the execution steps described in Algorithm 3.

There remain two issues concerning the sharing of the newly released resources: 1) Can the execution time of a running task with $\boldsymbol{r}^* \neq \boldsymbol{r}^{(*)}$ be further reduced by allocating additional resource? 2) If the answer to the above question is yes, would there occur the *resource contention problem*, i.e., there exist two running tasks which compete for the newly released resource at the same resource attributes (or dimensions). Theorems 3 and 4 answer the two questions.

Theorem 3. *$\forall t_{(i)}$ such that $r^*_{(i)k} < r^{(*)}_{(i)k}$, if another task was just completed and its released resource shares are denoted $\triangle \boldsymbol{r} = (\triangle r_1, \triangle r_2, \ldots, \triangle r_R)^T$ and $\triangle r_k > 0$, then $t_{(i)}$'s execution time is able to be reduced by increasing $r^*_{(i)k}$.*

Proof. For $t_{(i)}$, if $\boldsymbol{b}^T \cdot \boldsymbol{r}^*_{(i)} < B_{(i)}$, it is obvious that $\boldsymbol{r}^*_{(i)} \prec \boldsymbol{r}^{(*)}_{(i)}$, because this statement is a contraposition of the statement "$\exists\, \boldsymbol{r}^*_{(i)} = \boldsymbol{r}^{(*)}_{(i)} \Rightarrow \boldsymbol{b}^T \cdot \boldsymbol{r}^*_{(i)} = B_{(i)}$" according to Algorithm 1. Since $\triangle r_k > 0$, there must exist an increment $\delta > 0$ along the kth dimension, such that the new execution time of task $t_{(i)}$ (i.e., $\frac{l_1}{r_1} + \cdots + \frac{l_k}{r_k + \delta} + \cdots + \frac{l_R}{r_R}$) is smaller than its original execution time (i.e., $\frac{l_1}{r_1} + \cdots + \frac{l_k}{r_k} + \cdots + \frac{l_R}{r_R}$).

Let us discuss the situation that $\boldsymbol{b}^T \cdot \boldsymbol{r}^* = B_{(i)}$ for $t_{(i)}$. Recall that $\boldsymbol{r}^{(*)}$ is the resource vector calculated based on the Formula (12) (i.e., without the constraint of the capacity on each dimension), while $\boldsymbol{r}^*$ is the output of the Algorithm 1 (i.e., with the capacity constraint on each dimension). Since Formula (12) is derived from $\boldsymbol{b}^T \cdot \boldsymbol{r} = B_{(i)}$, it is obvious that $\boldsymbol{b}^T \cdot \boldsymbol{r}^{(*)} = B_{(i)}$. So, we get $\boldsymbol{b}^T \cdot \boldsymbol{r} = B_{(i)}$ holds for both $\boldsymbol{r} = \boldsymbol{r}^*$ and $\boldsymbol{r} = \boldsymbol{r}^{(*)}$. Hence, if there exists a dimension k such that $r^*_k < r^{(*)}_k$, there must exist another dimension (say j) such that $r^*_j > r^{(*)}_j$. Otherwise, $\boldsymbol{b}^T \cdot \boldsymbol{r}^* < \boldsymbol{b}^T \cdot \boldsymbol{r}^{(*)} = B$, which is contradicting to $\boldsymbol{b}^T \cdot \boldsymbol{r}^* = B_{(i)}$. Based on the above analysis, let us introduce a small increment $\delta(> 0)$ to $b_k \cdot r_k$ and the same amount of decrement $(-\delta)$ to $b_j \cdot r_j$ such that $\boldsymbol{r}^*_k + \frac{\delta}{b_k} \leq r^{(*)}_k$ and $r^*_j - \frac{\delta}{b_j} \geq r^{(*)}_j$. Then, we just need to prove that such an adjustment will make $t_{(i)}$'s execution time $f(\boldsymbol{r}(t_{(i)}))$ become shorter. We denote the original execution time of $t_{(i)}$ as X and its execution time after the adjustment as Y. We will show $X - Y > 0$. Note that the last deduction is due to (14)

$$X = \frac{l_1}{r_1} + \cdots + \frac{l_k}{r^*_k} + \cdots + \frac{l_j}{r^*_j} + \cdots + \frac{l_R}{r_R} \tag{19}$$

$$Y = \frac{l_1}{r_1} + \cdots + \frac{l_k}{r^*_k + \frac{\delta}{b_k}} + \cdots + \frac{l_j}{r^*_j - \frac{\delta}{b_j}} + \cdots + \frac{l_R}{r_R} \tag{20}$$

$$\begin{aligned} X - Y &= \left(\frac{l_k}{r^*_k} + \frac{l_j}{r^*_j}\right) - \left(\frac{l_k}{r^*_k + \delta/b_k} + \frac{l_j}{r^*_j - \delta/b_j}\right) \\ &= \delta\left(\frac{l_k/b_k}{r^*_k\left(r^*_k + \delta/b_k\right)} - \frac{l_j/b_j}{r^*_j\left(r^*_j + \delta/b_j\right)}\right) \\ &> \delta\left(\frac{l_k/b_k}{\left(r^{(*)}_k - \delta/b_k\right)r^{(*)}_k} - \frac{l_j/b_j}{\left(r^{(*)}_j + \delta/b_j\right)r^{(*)}_j}\right) \\ &> \delta\left(\frac{l_k/b_k}{r^{(*)}_k \cdot r^{(*)}_k} - \frac{l_j/b_j}{r^{(*)}_j \cdot r^{(*)}_j}\right) = 0. \end{aligned}$$

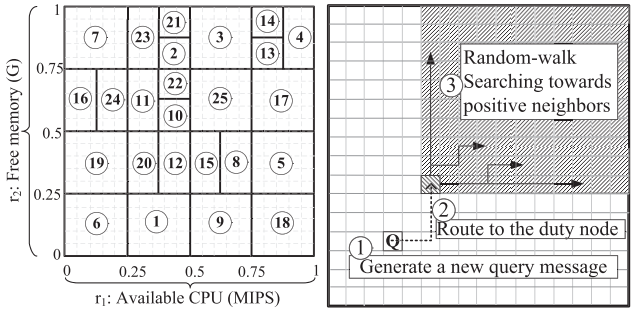
□

Theorem 4. *Based on Algorithm 3, there would not appear the conflict problem that two tasks with suboptimal resource allocation (i.e., $\exists\, k, r^*_k < r^{(*)}_k$) compete for the released resource along the same dimension upon the completion of a task.*

Proof. Provided that a task was just finished and the released resources are $\triangle \boldsymbol{r} = (\triangle r_1, \triangle r_2, \ldots, \triangle r_R)^T$ along R dimensions. Our objective is equivalently to prove that at any given dimension (denoted k), there exists at most one task (denoted $t_{(i)}$) such that $r^*_{(i)k} < r^{(*)}_{(i)k}$ based on our resource redistribution scheme described in Algorithm 3 (i.e., Line 5-11).

To prove by contradiction, suppose there do exist two tasks running on p_s (denoted as $t_{(i)}$ and $t_{(j)}$), where $t_{(i)}$ arrives earlier than $t_{(j)}$ and they have been scheduled before the newly completed task. In addition, both $t_{(i)}$ and $t_{(j)}$ need to use resource on the kth dimension (i.e., $r^*_{(i)k} > 0$ and $r^*_{(j)k} > 0$), while $r^*_{(i)k} < r^{(*)}_{(i)k}$ and $r^*_{(j)k} < r^{(*)}_{(j)k}$ hold simultaneously.

We will prove such scenario cannot happen as follows: If $r^*_{(i)k} < r^{(*)}_{(i)k}$ holds, this implies that after $t_{(i)}$ is scheduled (Line 3 in Algorithm 3), $t_{(i)}$ must already use up the



(a) A CAN Topology Example (b) Basic Search Procedure

Fig. 2. Range query on CAN.

resource at the dimension k. If $t_{(i)}$ had not used up the resource at dimension k (i.e., $r^*_{(i)k} < a_k$), this could only happen when $r^*_{(i)k} \not\leq r^{(*)}_{(i)k}$. If so, the original resource allocation on $t_{(i)}$ would not be optimized, which was proved by Theorem 3. Hence, $t_{(i)}$ must have already used up the resource on the k dimension if $r^*_{(i)k} < r^{(*)}_{(i)k}$. This contradicts to the assumption that the succeeding task $t_{(j)}$ is also able to be assigned with $r^*_{(i)k}$, where $0 < r^*_{(j)k} < r^{(*)}_{(j)k}$, from the kth dimension upon executing Line 3 in Algorithm 3 because the available amount of resource (i.e., $a_{(j)}$) along the dimension k is equal to 0, if $t_{(j)}$ is scheduled after $t_{(i)}$. This contradicts to the assumption that $r^*_{(j)} > 0$. Note $r^*_{(j)} = 0 \Rightarrow f_{(j)} = \infty$. Hence, it is impossible for more than one task to keep suboptimal share simultaneously along the same dimension. □

5 Pointer-Gossiping CAN

Our resource allocation approach relies on the assumption that all qualified nodes must satisfy Inequalities (6) and (7) (i.e., Lemma 1). To meet this requirement, we design a resource discovery protocol, namely *pointer-gossiping CAN*, to find these qualified nodes. We choose CAN [17] as the DHT overlay to adapt to the multidimensional feature.

Like traditional CAN, each node (a.k.a., *duty node*) under PG-CAN is responsible for a unique *multidimensional range* zone randomly selected when it joins the overlay. Fig. 2a illustrates an example of CAN overlay network. Suppose there are 25 joined nodes, each taking charge of a single zone. If a new node (node 26) joins, a random point such as (`0.6 Gflops, 0.55 GB`) will be generated and its zone will be set as the new zone evenly split along a dimension from the existing zone (node 25 in Fig. 2a) that contains this point. If there is only one nonoverlapped range dimension between two nodes (e.g., p_i and p_j) and they are adjacent at this dimension, we call them *neighbors* to each other. Furthermore, if the nonoverlapped range of p_i is always no less than p_j's, p_i is called p_j's *positive neighbor* and p_j is called p_i's *negative neighbor*. In Fig. 2a, Nodes 9, 12, and 20 are positive neighbors of node 1.

Every node will periodically propagate the state-update messages about its available resource vector $\boldsymbol{a}(p_s)$ to the *duty node* whose zone encloses this vector. After a task t_{ij} generates a query (Step 1 in Fig. 2b) with the constraints (6) and (7), the query message will be routed to the duty node containing the expected vector $\boldsymbol{e}(t_{ij})$. We could justify that the state messages (or state records) of all qualified nodes must be kept in those onward nodes (i.e., shadow area in Fig. 2b) of the duty node.

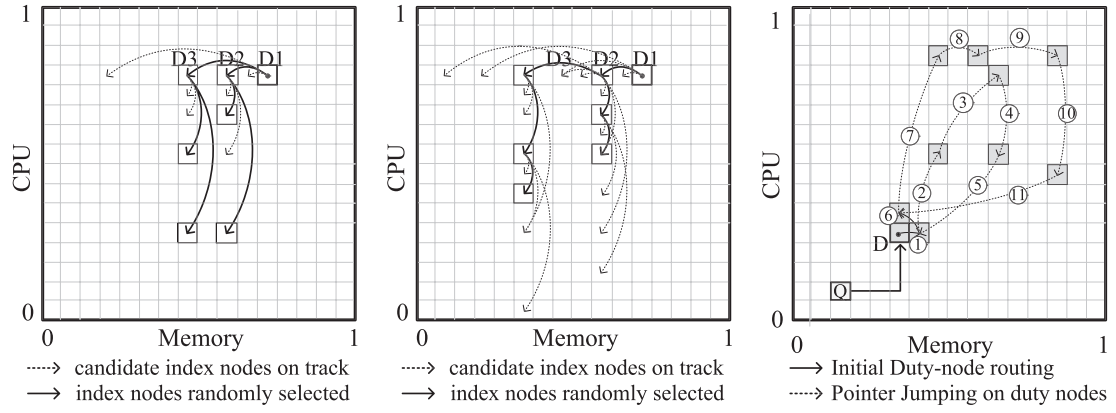
Obviously, the searching area may still be too large for the complete resource query without flooding, so the existing solutions [19] usually adopt random walk to get an approximated effect. However, according to our observation (to be presented), this will significantly reduce the likelihood of finding qualified resources, finally degrading the system throughput and user's QoS. Alternatively, we improve the mechanism by periodically diffusing a few pointer messages for any duty nodes owning state-update messages (or records) to the distant nodes (with distance as 2^k hops, where $k = 0, 1, \ldots$) toward negative directions, so that these duty nodes could be more easily found. In Fig. 2, for instance, Node 4's negative pointer nodes along CPU dimension are Nodes 14, 3, and 23. By periodically sending pointer-recovery messages, each with empty payload outward, each node could easily maintain the connection to the negative pointer nodes. On the other hand, each query routed to the duty node will check its stored records and the pointed duty nodes. If it finds qualified resource records on the current or other pointed duty nodes, it will return those information to the requesting node; otherwise, it will continue searching next positive neighbor duty nodes.

Each duty node (such as $D1$) will cache state-update messages received from its neighbors, which are checked periodically and removed if outdated (i.e., beyond their TTL). In the meanwhile, it propagates its own identifier (such as IP) to a few randomly selected pointer nodes toward it negative direction. For those duty nodes containing valid state messages, we call them *nonempty-cache nodes*.

Basically, there are two manners to propagate the duty nodes' identifiers (or pointers) backward—*spreading manner* (Fig. 3a) and *hopping manner* (Fig. 3b), thus the PG-CAN can also be split into two types, namely *spreading manner-based PG-CAN* (SPG-CAN) and *hopping manner-based PG-CAN* (HPG-CAN). In Fig. 3a, the duty node D_1 sends a pointer-message containing D_1's identifier to its selected pointer nodes (such as D_2 and D_3), notifying them that D_1 has records. Upon receiving the message, the pointer nodes (D_2 and D_3) will further gossip D_1's identifer to their negative direction pointer nodes along next dimension. In Fig. 3b, the identifer of any nonempty-cache node will be forwarded from pointer node to pointer node along each dimension. Obviously, the former results in fewer number of hops for message delivery, but its identifers cannot be diffused as widely as the latter's. In fact, we can prove that the delay complexity of identifier delivery for the hopping manner is $O(\log_2 n)$ (Theorem 5), so the hopping manner is likely to be better than the spreading manner (to be confirmed in our simulation).

Theorem 5. *The delay complexity of hops by hopping manner for relaying any node's index to any of its negative-direction nodes is $O(\log_2 n)$, where n refers to the total number of nodes.*

Note that $\log_2 n = d \cdot \log_2 n^{\frac{1}{d}}$, so our objective is to prove the delay is bounded under $d \cdot \log_2 n^{\frac{1}{d}}$. The strict proof can be found in our previous work [24]. Here, we just use an



(a) Pointer gossiping (Spreading Manner) (b) Pointer gossiping (Hopping Manner) (c) Range-query procedure

Fig. 3. The procedure of resource matching.

example (shown in Fig. 4) to illustrate the idea. In this example, suppose there are $n^{\frac{1}{d}} = 19$ nodes along each dimension, it is obvious that the top-most node (Node 1) will take longest time (less than $O(\log(19)) = 4$) to diffuse its own index. Specifically, over the first hop, Nodes 2, 3, 5, 9, and 17 could receive the index (Node 1's identifier). Via the second hop, Nodes 4, 6, 7, 10, 11, and 13 could receive the relayed index. For instance, Node 7 could receive Node 1's index forwarded from Node 5 or Node 3. With just three hops, most of the negative-direction nodes of Node 1 could receive its index notification.

Obviously, it is infeasible for peer nodes to broadcast their indexes (either their own identifiers or those of other nodes to forward) due to the probably considerable message delivery overhead. Suppose that L negative index nodes are selected along each dimension as the notification targets, the total number of the messages (denoted as ω) to deliver for any index is equal to $L + L^2 + \cdots + L^d = \frac{L \cdot (L^d - 1)}{L - 1}$. Hence, the message overhead could be controlled by setting L to a small value. For example, if $L = 2$ and $d = 3$, the total number of messages is always only 14. In other words, L has to be seriously limited in the design. L will always be set to 2 in our following design. The whole pointer gossiping procedure is conducted by two algorithms, *pointer-sender* and *pointer-relay*. We just show the pseudo-codes for hopping manner (as presented in Algorithm 4 and Algorithm 5), since the spreading manner's can be easily converted from it. Specifically, unlike the line 3-4 of Algorithm 4, the spreading manner-based pointer-sender algorithm will randomly select L negative pointer nodes along the dimension #1 and send its state message to them. The corresponding pointer-relay algorithm of these pointer nodes will not only store these messages but also forward them to L randomly selected negative-direction pointer nodes at the next dimension.

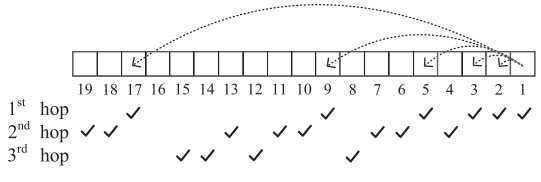


Fig. 4. Quick backward index diffusion.

Algorithm 4. POINTER-SENDER (HOPPING MANNER)
This program is invoked as long as p_d detects that it owns records.

1: **while** (TRUE) **do**
2: Construct a pointer-message containing p_d's identifier, i.e., {p_i's ID, #1}, where #1 refers to the first dimension;
3: Randomly select a negative pointer node PN_i at dimension #1;
4: Send {p_i's ID, #1, #1} to PN_i;/*the 2nd field and 3rd field indicate the current dimension number and the number of pointer-messages sent along this dimension*/
5: Sleep for a tiny cycle;
6: **end while**

Algorithm 5. POINTER-RELAY (HOPPING MANNER)
This is invoked upon receiving a pointer-message {p_i's ID, #j, #k}.

1: Put p_i's ID in *PointerList* on the current node;
2: **if** ($k < L$) **then**
3: Randomly select a negative pointer node PN_i along dimension #j;
4: Send {p_i's ID, #j, #k} to PN_i;
5: **else**
6: $j = j + 1$;
7: **if** ($j < R$) **then**
8: Construct a new pointer-message: {p_i's ID, #j #1};
9: Randomly select one negative pointer node along dimension #j;
10: Send {p_i's ID, #j, #1} to PN_i;
11: **end if**
12: **end if**

The procedure of resource query is shown in Fig. 3c. When a node (denoted as Q) generates a query message, it will first be routed to its duty node (denoted as D). On Node D, each stored record will be checked against the message's demand (i.e., Conditions (6) and (7)). If Node D keeps enough qualified records for the query, they will

TABLE 2
System Setting

Parameter	Value
# of nodes	2000 $\sim$ 12000
# of processors per node	1,2,4,8
computation rate per processor	1,2,2.4,3.2 Hz (or 10MI)
I/O speed per node	20,40,60,80 MbPS
memory size per node	512, 1024, 2048, 4096 MB
disk size per node	20, 60, 120, 240 Gb
LAN network bandwidth	5 $\sim$ 10 Mbps
WAN network bandwidth	0.2 $\sim$ 2 Mbps

be returned to the requesting node and the query will be terminated. If there are no matched records, a few other duty nodes pointed by the current duty node will be randomly selected and encapsulated in a so-called *pointer-jump* message, which will be sent outward in a relay fashion (Steps 2, 3, 4, 5 in Fig. 3b) until it meets the qualified records midway through the traversal (then the query will be terminated) or all of the pointed duty nodes are checked (then the query message will be forwarded to D's next neighbor (Step 6)). We present the pseudocode in Algorithm 6. *FoundList* is used to keep the matched records after traversing all the ones stored on the current node. At Line 10, the current node sends the query message to another duty node; upon receiving such a message, the remote duty node will also perform Line 2-12, yet at line 7 *JumpList* will be extracted from the received message instead.

Algorithm 6. RESOURCE QUERY ALGORITHM
This program at node p_q is invoked upon receiving a query message $\{\boldsymbol{e}, \boldsymbol{w}, B\}$.

```
1: if (the current node is the duty node) then
2:     Search p_q's record list and put the qualified records
       in FoundList;
3:     if (FoundList is not empty) then
4:         Send FoundList to the requesting node;
5:         Return;/*Query is terminated here*/
6:     else
7:         Construct JumpList by randomly selecting a few
           pointed duty nodes;
8:         if (JumpList is not empty) then
9:             Randomly take out a duty node and remove it
               from the JumpList;
10:            Send the query message with JumpList to the
               selected duty node;
11:        end if
12:    end if
13: else
14:    Forward the query message {e, w, B} based on
       CAN's routing rules;
15: end if
```

Note that the returned query result *FoundList* could be a set of qualified resource nodes based on Algorithm 3. Consequently, upon receiving the query result, the requesting node will randomly choose one out of them as the final resource node for executing the submitted task. With this random selection policy, we can effectively mitigate the decision conflict among different tasks (i.e., different nodes with analogous resource demands select the same node for executing their tasks, resulting in an abrupt resource overutilization situation) due to the un-coordinated node selection process from those autonomous participants. However, even with such an opportunistic scheduling policy, resource overutilization and load unbalancing phenomena cannot be totally eliminated if the number of tasks to be executed at a node cannot be controlled. We investigate three different policies to control imported tasks or disperse the load distribution, namely *double-check policy* [21], *queue-assistant policy*, and *extra-virtual-dimension policy* [19]. For the double-check policy, each requesting task will recheck the current resource availability of the selected node before the task is actually migrated. If the remote node does not allow extra load importing because this could make it overutilized due to an earlier task admission from another node, the task could get one more chance to select another qualified node. Unlike double-check policy, queue-assistant policy allows user tasks to be temporally queued on the selected resource node even though its current resource cannot fit the new demand immediately. Extra-virtual-dimension policy, which adopts an additional dimension for any new node joining the CAN overlay, is also a candidate policy in dispersing the zone distribution. We will evaluate all these policies in the next section.

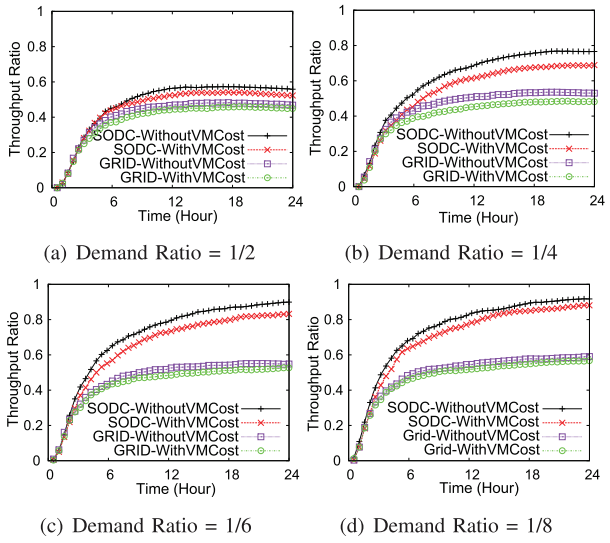
TABLE 3
User Task's Demand

Parameter	Value	Parameter	Value
demand ratio λ	$1/8 \sim 1$	cpu rate	$\lambda \sim 25.6\lambda$
I/O speed	$20\lambda \sim 80\lambda$	memory size	$512\lambda \sim 4096\lambda$
disk size	$20\lambda \sim 240\lambda$	bandwidth	$0.1\lambda \sim 10\lambda$

6 Performance Evaluation

6.1 Experimental Setting

To conduct the simulation, we first build an emulated proportional-share scheduler in accordance with Xen's credit-scheduler. We use PeerSim [26] to implement the proposed CAN-based range query protocols on a large network containing 2,000 to 12,000 participating nodes. The hardware configuration of each node is randomly selected according to system parameters specified in Table 2. Via this table, we can derive the $min\_capacity$ and $max\_capacity$ at each resource dimension. For instance, along the CPU dimension, $min\_capacity$ and $max\_capacity$ are $1 \times 1 = 1$ Gflops and $8 \times 3.2 = 25.6$ Gflops, respectively, which happen when there is only one core at a node running at the speed of 1 Gflops and eight cores per node, each operating at 3.2 Gflops. Each node's resource prices are randomly generated from the range $[\frac{1}{min\_capacity}, \frac{100}{max\_capacity}]$. To investigate the contention issue in the course of resource query among user requests, we use a parameter called *demand ratio* (denoted as λ, where $1/8 \leq \lambda \leq 1$) to control the generation of resource demand from each user task, as shown in Table 3. Intuitively, λ indicates different contention levels on each resource dimension in presence of large number of analogous queries injected from participants. For instance, if λ is set to 1.0, each task's expected resources would be randomly set in $[1.0 \times min\_capacity, 1.0 \times max\_capacity]$. If λ is set to 0.2, the resource amounts demanded by all the



(a) Demand Ratio = 1/2 (b) Demand Ratio = 1/4

(c) Demand Ratio = 1/6 (d) Demand Ratio = 1/8

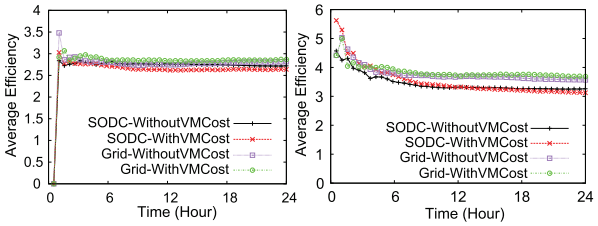
Fig. 5. SOC versus P2P Grid (on throughput).

tasks will be distributed in $[0.2 \times min\_capacity, 0.2 \times max\_capacity]$ at each dimension, leading to a higher level of contention.

Each experiment simulates 86,400 seconds (i.e., one day) and each node will periodically receive the user requests whose workload on each attribute (such as CPU, I/O) will be randomized based on a Poisson process with 4,000 seconds as its mean. For example, if a request's workload vector is (CPU $= 2.4$ GFLOP, disk data =100 Mb, network data $= 200$ Mb), it will be finished until all the workloads are processed by subtracting the allocated shares of resource over time. Such a request could be analogous to the jobs which contain sequentially submitted tasks in Google's trace [27], where job lengths are from dozens of minutes to several days. Moreover, Google's trace shows that most of jobs contain only one single service, which conforms to our multiattribute resource allocation model. The weight vector of each task is generated based on the ratio of its workload on different resource dimensions (or phases). In practice, the weight vector could be estimated by statistics based on normalized usage data like Google's trace. The TTL (i.e., age) of each state-update message is 600 seconds and message updating cycle is 400 seconds.

We first compare the execution efficiency of SOC to that of P2P desktop Grid [19], [20] by taking into account the VMM overhead, to validate the efficacy of the DOPS algorithm. According to Google's trace, task's execution may be related to different attributes like CPU and memory. We consider five types of resource demand: CPU, disk speed, network, memory size, and disk space; thus constituting a 5D resource attribute space. The last two will not impact the task's execution time but are regarded as the constraints during the resource discovery phase. According to the existing experimental report [29], we set the cost in maintaining one VM instance as follows: processor rate $= 5\%$, IO speed $= 10\%$, network bandwidth $= 5\%$, and memory cost $= 5$ MB.

We compare PG-CAN with hopping manner and spreading manner to other solutions, including the basic *newscast* model [30], *random diffusion CAN* (RD-CAN) [19],



(a) Demand Ratio = 1/2 (b) Demand Ratio = 1/6

Fig. 6. SOC versus P2P grid (on execution efficiency).

and *virtual-dimension* support [19]. Under the newscast setting, each node maintains a fixed-size cache containing $2\log(n)$ neighbors which are randomly selected from the whole node set. Each node will periodically push its state message to three sampled random neighbors and be further gossiped for three more hops. Any node's cache could be refreshed by merging with one of its neighbors periodically. In RD-CAN, any duty node diffuses its received state messages over CAN toward the negative neighbors for a few hops, and any query message is raised with Condition (6) and (7). With VD support, every state records is inserted into the CAN space based on $R + 1$ dimensions, in order to mitigate the analogous query contention.

6.2 Experimental Results

Fig. 5 presents the throughput ratio between SOC and P2P desktop Grid (2,000 nodes), using spreading manner-based PG-CAN with the extra-virtual-dimension policy support and different demand ratios $\lambda\,(= 1/2, 1/4, 1/6, \text{ and } 1/8)$. The throughput ratio is defined as the ratio of the number of finished tasks and the total number of generated tasks in the whole system over time. In SOC, every task is allowed to share the multiple types of resources on the same node, so the resources can be utilized more abundantly. For example, a CPU-bound task and an IO-bound task could run at the same physical node at the same time by leveraging VM resource isolation technology.

We observe that SOC would achieve up to about 60 percent improvement as task sizes are relatively small on average (say $\lambda = 1/6$ or $1/8$). When all the task sizes are relatively large (say $\lambda = 1/2$), SOC could still get about 15 percent improvement compared to P2P Grid model. Another observation is that the additional cost of maintaining VM instances is always constant, which becomes neglectable with smaller task sizes.

In addition to the throughput, we also evaluate task's execution efficiency from the perspective of execution time. According to user's expected resource vector $\boldsymbol{e}(t_{ij})$, we define t_{ij}'s *expected execution time* as $\sum_{k=1}^{R} \frac{l_k}{e_k}$. Then, we define t_{ij}'s *execution efficiency* (denoted as ϵ_{ij}) as its expected execution time divided by its real turnaround time (from the task's submission to its completion). Apparently, higher value of ϵ_{ij} implies shorter execution time of task t_{ij}. From Fig. 6, we observe that both SOC and P2P Grid deliver satisfactory average execution efficiency, which is calculated based on all the finished tasks. The reason why the average execution efficiency in P2P Grid appears a little higher than that in SOC is due to the fact that exclusive queuing model in P2P Grid may allocate relatively more resource amount to

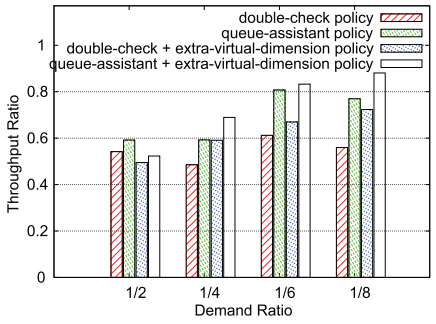


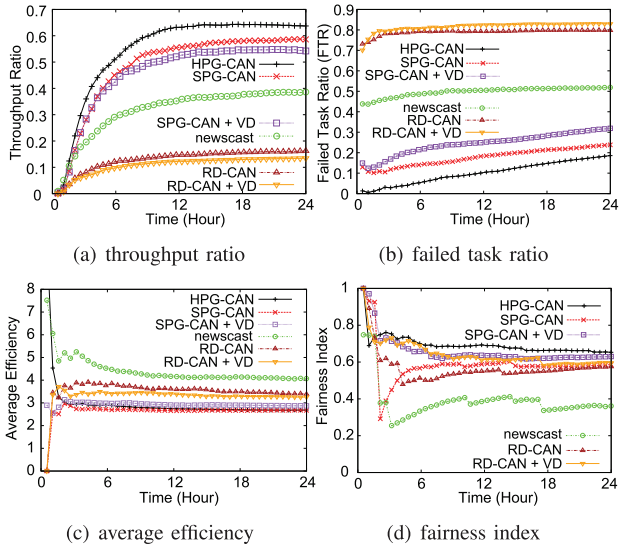
Fig. 7. Throughput ratio under different load control policies.

each task. While this could result in shorter execution time for each individual task, the tradeoff is a lower throughput as reported in Fig. 5.

Fig. 7 shows the converged throughput ratio under the SPG-CAN with different policies (or their combinations) to control the load congestion with different demand ratios. We observe that the combination of double-check/queue-assistant policy and the extra-virtual-dimension policy performs better than the pure policies without extra-virtual-dimension. We also observe that the queue-assistant policy outperforms double-check policy in most of cases. Recall that in the queue-assistant policy, the task will be failed (i.e., the searching is terminated) if there were no qualified resources found. This means that this policy suffers the least query cost compared to others. As such, the combination of queue-assistant and extra-virtual-dimension policy seems the best. As follows, we will show that HPG-CAN without any load control policy would still outperform the SPG-CAN with extra-virtual-dimension policy. In the rest of this section, we uniformly adopt the queue-assistant mode with VM cost.

Figs. 8 and 9 present the effectiveness of different range query protocols on SOC with 2,000 nodes, under various load ratios (λ). The *failed task ratio* in Fig. 8b refers to the number of tasks that cannot find qualified node divided by the total number of submitted tasks.

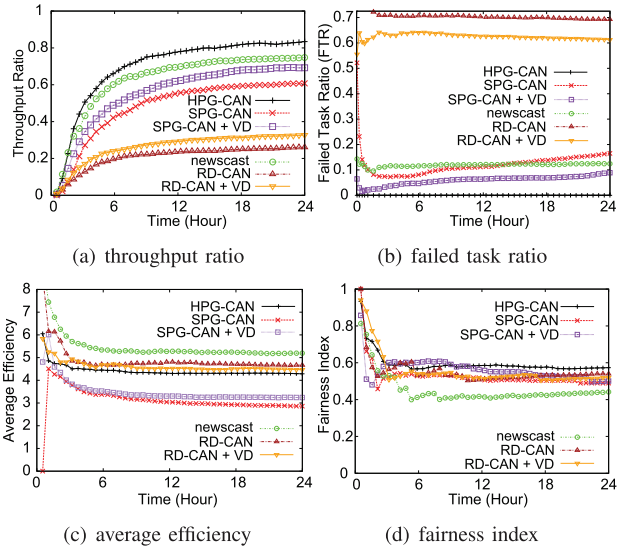
We observe that HPG-CAN leads to the best performance (including highest throughput ratio and lowest failed task



(a) throughput ratio (b) failed task ratio

(c) average efficiency (d) fairness index

Fig. 8. The effectiveness of query protocols ($\lambda = \frac{1}{2}$).



(a) throughput ratio (b) failed task ratio

(c) average efficiency (d) fairness index

Fig. 9. The effectiveness of query protocols ($\lambda = \frac{1}{4}$).

ratio) in that it could efficiently discover the global idle resources. Moreover, its failed task ratio can be limited down to 0.00007 in the situation with smaller load-ratio λ (Fig. 9b). Note that smaller λ means higher degree of resource contention in that all such small-demand queries would always be routed to the similar duty nodes located at the lower position of the CAN space. In comparison, SPG-CAN works notably inferior to HPG-CAN because of its suboptimal effect in gossiping nonempty-cache nodes' identifiers. Fig. 9c shows an interesting result about average execution efficiency: task's execution efficiency under HPG-CAN converges up to 4, which is much higher than that of SPG-CAN. In other words, HPG-CAN outperforms the other solutions on all the three key metrics.

Furthermore, we use Jain's *fairness index* [31] (denoted $\varphi \in [0,1]$) to evaluate the fairness among user tasks' completion time. The fairness index is given in (21), where ϵ_{ij} refers to task t_{ij}'s execution efficiency. Higher φ implies more steady execution efficiency

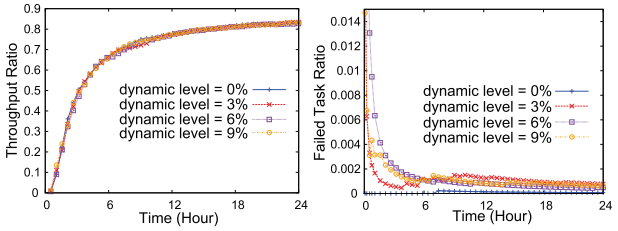
$$\varphi = \frac{\left(\sum_{i=1}^{n}\sum_{j=1}^{m_i}\epsilon_{ij}\right)^2}{\left(\sum_{i=1}^{n} m_i\right)\cdot\left(\sum_{i=1}^{n}\sum_{j=1}^{m_i}\epsilon_{ij}^2\right)}. \quad (21)$$

Figs. 8c and 9c present the fairness of all the *completed* tasks' execution. We observe that HPG-CAN reaches the highest fairness, which means it provides most stable results compared to other solutions.

We evaluate the system scalability (as shown in Table 4) of the PG-CAN protocol. All the values in this table are recorded after the one-day duration test. With the increasing system scale (e.g., up to 12,000 nodes), the performance metrics (including throughput, average efficiency (i.e., the

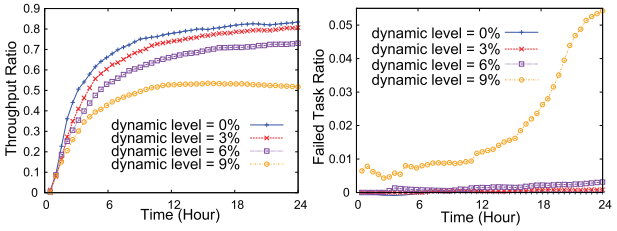
TABLE 4
System Scalability of PG-CAN

metric \ scale	2000	4000	6000	8000	10000	12000
throughput ratio	0.598	0.592	0.571	0.568	0.572	0.575
failed task ratio	23.7%	26.4%	27.1%	27.8%	27.9%	27.1%
average efficiency	2.72	2.61	2.60	2.58	2.57	2.56
fairness index	0.665	0.673	0.677	0.666	0.681	0.672
msg delivery cost	2913	3613	4220	4701	5067	5280



(a) throughput ratio (b) failed task ratio

Fig. 10. **HPG-CAN** under dynamic environment.



(a) throughput ratio (b) failed task ratio

Fig. 12. **HPG-CAN** under dynamic environment with checkpointing and task migration cost.

mean value of ϵ_{ij}), etc.) will not change notably. The *message delivery cost* is defined as the number of messages to be sent/forwarded by each node on average during the whole 24 hours. Our test shows that this cost increases roughly with a logarithmic speed, which is much better than linear rate. Moreover, most of messages are actually lightweight pointer messages, each containing just an identifier. Thus, the PG-CAN protocol can result in little message delivery overhead.

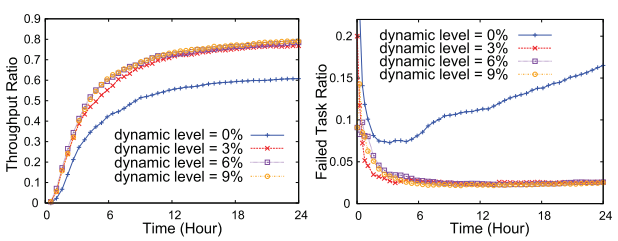
We evaluate the PG-CAN protocol under node churning situations. We assume there are X percent of nodes arbitrarily joining/departuring the entire system every minute. We faithfully implement the node departure maintenance on each departure node's neighbors to refresh their neighborhoods and a binary partition tree-based *background zone reassignment* algorithm [17] to ensure each node always corresponds to one globally unique zone. Specifically, each node does not actually maintain the global view of such a tree but only needs to distributively communicate with its neighbors.

In our simulation, the demand ratio λ is set to 1/4 and X percent (also called *dynamic level*) will be set to 0, 3, 6, and 9 percent. Note that the environment with node churning rate (dynamic level) set to 3 percent per minute is already very volatile, especially compared to the 4,000-seconds average completion time for all tasks (as mentioned in Section 6.1).

In order to observe the impact of node churning to the PG-CAN, we first conduct our simulation under an assumption that the tasks would not be suspended/interrupted on the departure nodes. Later, we eliminate this assumption by considering task checkpoint/migration cost to observe the synthetic system performance. Figs. 10 and 11 show the PG-CAN's working efficiency under the noninterrupting task condition, based on hopping manner and spreading manner, respectively. It is not surprising that HPG-CAN works much better than SPG-CAN in the dynamic environment, due to considerably higher system throughput (HPG-CAN converges up to 0.83 while SPG-CAN converges to about 0.78) and lower failed task ratio (HPG-CAN converges to about 0.0005 while SPG-CAN converges to 0.025).

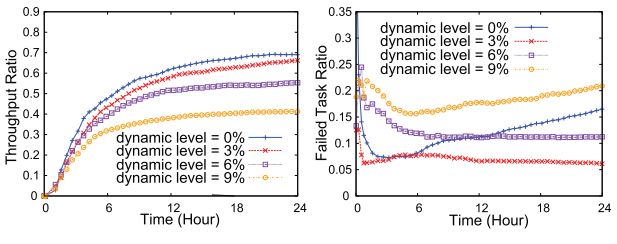
In Fig. 11b, we observe that the blue curve (the case without any churning nodes) decreases at the beginning and increases linearly for the rest of time. This is reasonable due to the following analysis: at the beginning, the whole system is relatively idle such that most of the hosts are qualified for any submitted tasks; however, with increasing number of tasks submitted, the failure probability of finding qualified nodes for any task will be increased accordingly, especially when the rate of processing tasks becomes lower than that of importing new tasks. What is most interesting for SPG-CAN is that the overall system performance (including throughput ratio, failed task ratio) will not get worse with increasing dynamic level of the environment, but get prominent improvement on contrary (See Fig. 11). This is sound, since in the dynamic environment, the nodes would frequently change their neighbors and the pointer-cache maintained would also be changed accordingly. As such, the pointers and state messages would be more widely diffused than the original SPG-CAN, leading to a higher probability of finding qualified resource nodes.

We further analyze the overall system performance by taking into account the tasks' interruption and their checkpointing/migration cost on the departure nodes. In practice, the tasks running on the departure nodes will probably be interrupted or suspended until the nodes are restored. Hence, we assume the scheduled (or running) tasks on any departure node would take longer time to complete, and the wasted time is set to be equal to that for executing 10 percent more load of the current remaining workload at each resource attribute. From Figs. 12 and 13, we observe that HPG-CAN with 6 percent-dynamic-level can still outperform SPG-CAN with 0 percent-dynamic-level. Although the performance degradation could be observed with increasing dynamic level (Fig. 12), the whole system could still perform very well when there are 3 percent churning nodes per minute (i.e., about $1 - 0.97^{23} = 50.4$ percent churning nodes per 23 minutes), confirming the high adaptability of our solution.



(a) throughput ratio (b) failed task ratio

Fig. 11. **SPG-CAN** under dynamic environment.



(a) throughput ratio (b) failed task ratio

Fig. 13. **SPG-CAN** under dynamic environment with checkpointing and task migration cost.

7 Related Work

SOC is different from the traditional Grid model (including P2P desktop Grid [32], [33]) in the resource consumption manner. Grids generally assume exclusive resource usage to ensure user QoS. The problem of job scheduling in Grids is usually categorized as a *multiprocessor scheduling (MPS)* problem [5], [6] (a kind of combinatorial optimization problem), which has been proved to be NP-complete [7]. Accordingly, many approximation algorithms as well as (meta)heuristics applied to various versions of the MPS problem in the Grid environment have been studied. For example, Rossi et al. [34] proposed a metaheuristic for solving the fixed job scheduling problem where processors are subject to spread time constraints, i.e., the time spent between the submission time and the completion time should not exceed a given duration. *Generalized Extremal Optimization* (GEO) [35] is another metaheuristic for solving the MPS problem. Singh et al. [36] approached the Grid scheduling problem through a cost-based provisioning model and a multiobjective genetic algorithm for getting approximately optimized performance (such as throughput). In P2P desktop Grids, Kim et al. [19] proposed a heuristic load balancing method for improving the task scheduling throughput on desktop Grids over CAN overlay. Similarly, Zheng et al. [37] formulated the problem to be a *bins-and-balls model* with herds phenomenon and tried to get the approximately optimal performance using a stochastic algorithm atop a DHT overlay. Lee and Snavely [38] studied a user-centric utility function of task turnaround time to improve the system performance based on simulation. Compared with these existing works, we devise an autonomous VM-multiplexing resource consumption model, namely SOC, which allows each task to dynamically make full use of the resource slices isolated by VM technology.

Although there are also a few existing research studies on VM-multiplexing strategies, they are not well suited to the SOC for most of them mainly focus on a few specific attributes such as CPU or memory. For instance, *virtual-putty* [9] used an application-load forecasting method as well as a strategy for reshaping the involved VMs to improve a single host's CPU and I/O resource utilization. Gupta et al. [10] proposed a method allowing memory sharing to happen within page boundaries only. Govindan et al. [11] adopted statistical multiplexing of applications to make applications fit into the given power budgets. In contrast, Meng et al. [8] explicitly endeavored to maximize the VM-multiplexing resource utilization by analyzing VM-pairs' compatibility in terms of the forecasted workload and estimated VM sizes. However, two significant drawbacks still remain: 1) *poor scalability* due to the central management of VM-correlation matrix; 2) *restrictive constraints* on implementation since they only identify the compatibility of VM-pairs. To overcome these problems, we formulate multiattribute resource allocation as a convex optimization problem and devise a resource allocation algorithm to minimize the task execution time with $O(R^3)$ time complexity.

Since the node identifiers over the DHT are often generated based on some hash functions, it is uneasy to directly perform range queries. Some existing strategies [39] have to build an extra layer to reorganize all of nodes over the DHT, whereas others (such as [19]) leverage a CAN topology. Many other existing works [39], [40], [41], [42], [43], [44] mainly focus on how to locate the duty nodes that satisfy the user-specified range in all dimensions with limited delays. However, for most tasks with low resource requirements (which is true in most cloud-based applications), nearly all the nodes in the network can be qualified. This will generate a vast amount of network traffic and also adds large burden to user on the filtering process. Indeed, most ordinary users just want the system to quickly locate a qualified node to meet its QoS goals. This issue however becomes more complex due to the adoption of rather flexible VM-enabled allocation scheme and the high-dimensional range query conditions. In view of this problem, we propose a new distributed protocol to search resources with the mitigated contention among requesters and strictly limited query-message traffic cost.

8 Conclusions and Future Work

This paper proposes a novel scheme (DOPS) for virtual resource allocation on a SOC, with three key contributions listed below.

- *Optimization of task's resource allocation under user's budget*: With a realistic monetary model, we propose a solution which can optimize the task execution performance based on its assigned resources under the user budget. We prove its optimality using the KKT conditions in the convex-optimization theory.
- *Maximized resource utilization based on PSM*: In order to further make use of the idle resources, we design a dynamic algorithm by combining the above algorithm with PSM and the arrival/completion of new tasks. This can give incentives to users by gaining an extra share of unused resource without more payment. Experiments confirm achieving a superoptimal execution efficiency of their tasks is possible. DOPS could get an improvement on system throughput by 15 percent $\sim$60 percent than the traditional methods used in P2P Grid model, according to the simulation.
- *Lightweight resource query protocol with low contention*: We summarize the resource searching request as two range query constraints, (6) and (7). We prove them to be the sufficient and necessary conditions for getting the optimal resource allocation. Experiments confirm the designed PG-CAN protocol with lightweight query overhead is able to search qualified resources very effectively.

So far, we have successfully built a prototype supporting live migration of VMs between any two nodes on the Internet (even though they are behind different NATs). In the future, we will study fault-tolerance support for a (DOPS-based, PG-CAN-enabled) SOC system; we will also conduct sensitivity analysis of how violation of our model assumptions would impact the optimal resource allocation.

Acknowledgments

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

References

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

[2] 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. Middleware (Middleware '06)*, pp. 342-362, 2006.
[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. 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, Feb. 2009.
[4] D.P. Anderson, "Boinc: A System for Public-Resource Computing and Storage," *Proc. IEEE/ACM Fifth Int'l Workshop Grid Computing*, pp. 4-10, 2004.
[5] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*. ftp://ftp.nada.kth.se/Theory/Viggo-Kann/compendium.pdf, 2012.
[6] O. Sinnen, *Task Scheduling for Parallel Systems*, Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2007.
[7] O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *J. ACM*, vol. 24, pp. 280-289, Apr. 1977.
[8] X. Meng et al., "Efficient Resource Provisioning in Compute Clouds via vm Multiplexing," *Proc. IEEE Seventh Int'l Conf. Autonomic Computing (ICAC '10)*, pp. 11-20, 2010.
[9] J. Sonneck and A. Chandra, "Virtual Putty: Reshaping the Physical Footprint of Virtual Machines," *Proc. Int'l HotCloud Workshop in Conjunction with USENIX Ann. Technical Conf.*, 2009.
[10] D. Gupta et al., "Difference Engine: Harnessing Memory Redundancy in Virtual Machines," *Proc. Eighth Int'l USENIX Symp. Operating Systems Design and Implementation*, pp. 309-322, 2008.
[11] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini, "Statistical Profiling-Based Techniques for Effective Power Provisioning in Data Centers," *Proc. Fourth ACM Conf. European Conf. Computer Systems (EuroSys '09)*, pp. 317-330. 2009,
[12] M. Feldman, K. Lai, and L. Zhang, "The Proportional-Share Allocation Market for Computational Resources," *IEEE Trans. Parallel and Distributed Systems*, vol. 20, no. 8, pp. 1075-1088, Aug. 2009.
[13] S. Soltesz, H. Poetzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *Proc. Second ACM Int'l European Conf. Computer Systems (Euro '07)*, pp. 275-287. 2007,
[14] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three cpu Schedulers in Xen," *SIGMETRICS Performance Evaluation Rev.*, vol. 35, no. 2, pp. 42-51, 2007.
[15] "The Role of Memory in Vmware Esx Server 3: On Line At: http://www.vmware.com/pdf/esx3_memory.pdf,"technical report, 2012.
[16] dm-ioband: http://sourceforge.net/apps/trac/ioband, 2012.
[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM '01*, pp. 161-172, 2001.
[18] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM '01*, pp. 149-160, 2001.
[19] J.S. Kim et al., "Using Content-Addressable Networks for Load Balancing in Desktop Grids," *Proc. 16th ACM Int'l Symp. High Performance Distributed Computing (HPDC '07)*, pp. 189-198, 2007.
[20] A. Leite, H. Mendes, L. Weigang, A. de Melo, and A. Boukerche, "An Architecture for P2P Bag-of-Tasks Execution with Multiple Task Allocation Policies in Desktop Grids," *Proc. IEEE Int'l Conf. Cluster Computing*, pp. 1-11, Feb. 2011.
[21] Y. Drougas and V. Kalogeraki, "A Fair Resource Allocation Algorithm for Peer-to-Peer Overlays," *Proc. IEEE INFOCOM '05*, pp. 2853-2858, 2005.
[22] D. Gross and C.M. Harris, *Fundamentals of Queueing Theory*, Wiley Series in Probability and Statistics. Wiley-Interscience, Feb. 1998.
[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge Univ. Press, 2009.
[24] S. Di, C.-L. Wang, W. Zhang, and L. Cheng, "Probabilistic Best-Fit Multi-Dimensional Range Query in Self-Organizing Cloud," *Proc. IEEE 40th Int'l Conf. Parallel Processing*, pp. 763-772, 2011.
[25] 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.
[26] Peersim Simulator: http://peersim.sourceforge.net, 2012.
[27] Google Cluster-Usage Traces: http://code.google.com/p/googleclusterdata, 2012.
[28] C.A. Waldspurger, "Memory Resource Management in Vmware Esx Server," http://www.usenix.org/events/osdi02/tech/waldspurger.html, 2012.
[29] J.P. Walters, V. Chaudhary, M. Cha, S. Guercio Jr., and S. Gallo, "A Comparison of Virtualization Technologies for hpc," *Proc. IEEE 22nd Int'l Conf. Advanced Information Networking and Applications (AINA '08)*, pp. 861-868, 2008.
[30] W.K. Mark Jelasity and M. van Steen, "Newscast Computing," technical report, Vrije Universiteit Amsterdam, 2006.
[31] R.K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*. John Wiley & Sons, Apr. 1991.
[32] J. Cao, F.B. Liu, and C.Z. Xu, "P2pgrid: Integrating P2P Networks Into the Grid Environment: Research Articles," vol. 19, no. 7, pp. 1023-1046, 2007.
[33] H. Abbes, C. Cerin, and M. Jemni, "Bonjourgrid: Orchestration of Multi-Instances of Grid Middlewares on Institutional Desktop Grids," *Proc. IEEE 23rd Int'l Symp. Parallel & Distributed Processing (IPDPS '09)*, pp. 1-8, 2009.
[34] A. Rossi, A. Singh, and M. Sevaux, "A Metaheuristic for the Fixed Job Scheduling Problem under Spread Time Constraints," *Computational Operation Research*, vol. 37, pp. 1045-1054, June 2010.
[35] P. Switalski and F. Seredynski, "Generalized Extremal Optimization for Solving Multiprocessor Task Scheduling Problem," *Proc. Seventh Int'l Conf. Simulated Evolution and Learning*, pp. 161-169, 2008.
[36] G. Singh, C. Kesselman, and E. Deelman, "A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids," *Proc. 16th ACM Symp. High Performance Distributed Computing (HPDC '07)*, 117-126, 2007.
[37] Q. Zheng, H. Yang, and Y. Sun, "How to Avoid Herd: A Novel Stochastic Algorithm in Grid Scheduling," *Proc. 15th ACM Int'l Symp. High Performance Distributed Computing (HPDC '06)*, pp. 267-278, 2006.
[38] C.B. Lee and A.E. Snavely, "Precise and Realistic Utility Functions for User-Centric Performance Analysis of Schedulers," *Proc. 16th ACM Int'l Symp. High Performance Distributed Computing (HPDC '07)*, pp. 107-116, 2007.
[39] A.R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *Proc. ACM SIGCOMM '04*, pp. 353-366, 2004.
[40] D. Li, J. Cao, X. Lu, and K.C.C. Chen, "Efficient Range Query Processing in Peer-to-Peer Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 21, no. 1, pp. 78-91, Jan. 2009.
[41] A. Gonzalezbeltran, P. Milligan, and P. Sage, "Range Queries Over Skip Tree Graphs," *Computer Comm.*, vol. 31, no. 2, pp. 358-374, Feb. 2008.
[42] S. Wang, Q.H. Vu, B.C. Ooi, A.K. Tung, and L. Xu, "Skyframe: A Framework for Skyline Query Processing in Peer-to-Peer Systems," *J. VLDB*, vol. 18, pp. 345-362, Jan. 2009.
[43] M.A. Arefin, M.Y.S. Uddin, I. Gupta, and K. Nahrstedt, "Q-Tree: A Multi-Attribute Based Range Query Solution for Tele-Immersive Framework," *Proc. IEEE 29th Int'l Conf. Distributed Computing Systems (ICDCS '09)*, pp. 299-307, 2009.
[44] J. Wang, S. Wu, H. Gao, J. Li, and B.C. Ooi, "Indexing Multi-Dimensional Data in a Cloud System," *Proc. ACM Int'l Conf. Management of Data (SIGMOD '10)*, pp. 591-602, 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 postdoctor 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 a member of the IEEE.