Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds

Haikun Liu
Nanyang Technological University, Singapore

Bingsheng He
Nanyang Technological University, Singapore

***Abstract*—Resource sharing in virtualized environments have been demonstrated significant benefits to improve application performance and resource/energy efficiency. However, resource sharing, especially for *multiple* resource types, poses several severe and challenging problems in *pay-as-you-use* cloud environments, such as sharing incentive, free-riding, lying and economic fairness. To address those problems, we propose *Reciprocal Resource Fairness* (*RRF*), a novel resource allocation mechanism to enable fair sharing multiple types of resource among multiple tenants in new-generation cloud environments. RRF implements two complementary and hierarchical mechanisms for resource sharing: inter-tenant resource trading and intra-tenant weight adjustment. We show that RRF satisfies several highly desirable properties to ensure fairness. Experimental results show that RRF is promising for both cloud providers and tenants. Compared to existing cloud models, RRF improves virtual machine (VM) density and cloud providers' revenue by 2.2X. For tenants, RRF improves application performance by 45% and guarantees 95% economic fairness among multiple tenants.**

***Keywords*: IaaS, cloud computing, resource sharing, fairness.**

I. INTRODUCTION

Infrastructure-as-a-service (IaaS) clouds have emerged as appealing computing infrastructures, which allow tenants to acquire and release resource in the form of virtual machines (VMs) on a pay-as-you-go basis. Nowadays, most IaaS cloud providers such as Amazon EC2 offer a number of VM types (such as *small*, *medium*, *large* and *extra large*) with fixed amount of CPU cores, main memory and disk. Tenants can only purchase fixed-size VMs and scale up/down the number of VMs when the resource demands change. This is also known as *T-shirt model* [18]. From the tenants' perspective, there are two disadvantages of the T-shirt model. First, the granularity of resource acquisition/release is coarse in the sense that the fix-sized VMs are not tailored for the dynamic demands of cloud applications delicately. The result is that tenants need to over-provision resource (costly), or risk performance penalty and Service Level Agreement (SLA) violation. Second, elastic resource scaling in the cloud [3] is also costly due to the latencies involved in VM instantiating [36] and software runtime overhead [41]. These costs are ultimately borne by tenants in terms of monetary cost or performance penalty. These two disadvantages both can be attributed to inefficient use of cloud resource. For cloud providers, inefficient resource utilization translates to higher capital expense and operating cost, and decreases revenue margins.

As more and more applications with diversifying resource requirements are already deployed in the cloud [1], [13], [20], [60], there are vast opportunities for resource sharing [18], [38]. Recent studies [16], [18] have shown the data center workloads can have heterogeneous demands on multiple resource types including CPU and memory. Recent work has shown that co-locating multi-tenant jobs on a set of shared compute nodes can increase mean application performance and system resource/energy efficiency by 20% [9], [26]. The resource utilization can be further improved in virtualization environments [18] by using fine-grained resource sharing techniques and optimizations (e.g., resource multiplexing or overcommitting model [7], [12], [18], [22], [53]). A previous study showed that the fine-grained resource sharing model can reduce the number of active servers by 50% compared to T-shirt model [18]. Modeling clouds as a market, many researchers have explored economics-based mechanisms for cost efficiency of tenants and resource efficiency of cloud providers [11], [30], [35], [37], [54].

Despite the resource sharing opportunities, we find that resource sharing, especially for *multiple* resource types (i.e., multi-resource), poses several important and challenging problems in pay-as-you-use commercial clouds. (1) *Sharing Incentive*. In a shared cloud with resource contention, a tenant may have concerns about the gain/loss of her asset in terms of resource. (2) *Free Riding*. A tenant may deliberately buy less resource than her demand and always expect to benefit from others' contribution (i.e., unused resource). Free Riders would seriously hurt other tenants' sharing incentive. (3) *Lying*. When there exists resource contention, a tenant may lie about her resource demand for more benefit. Lying also hurts tenants' sharing incentive. (4) *Gain-as-you-contribute Fairness*. It is important to guarantee that the allocations obey a rule "more contribution, more gain". In summary, those problems are eventually attributed to economic fairness of resource sharing in IaaS clouds.

Given the fairness problem, one of the most popular allocation policies proposed for fair sharing so far has been (Weighted) Max-Min Fairness (WMMF) [29], which maximizes the minimum allocation received by a user in the system. There are also a large amount of work on fair allocation based on WMMF [4], [5], [22], [43], [44], [48], [49]. However, their focuses have so far been primarily on a single resource type. Although Dominant Resource Fairness (DRF) [16] was proposed for multi-resource environments, it cannot address all the four problems of resource sharing in IaaS clouds (see Section II-B).

To address these problems, we propose *Reciprocal Resource Fairness (RRF)*, a generalization of max-min fairness to multiple resource types. The intuition behind RRF is that each tenant should preserve her asset while maximizing the resource usage in a cooperative environment. Due to the vast diversity and dynamics of resource demands in the cloud, we propose a fine-grained resource sharing scheme for multiple resource

SC14, November 16-21, 2014, New Orleans, LA, USA
978-1-4799-5500-8/14/$31.00 © 2014 IEEE

types. Particularly, we advocate that different types of resource can be traded among different tenants and be shared among different VMs belonging to the same tenant. For example, a tenant can trade her unused CPU share for other tenant's over-allocated memory share. In this paper, we consider two major kinds of resources, including CPU and main memory. Resource trading can maximize tenants' benefit from resource sharing. RRF addresses the multi-resource fair sharing problem into with complementary mechanisms: inter-tenant resource trading and intra-tenant weight adjustment. These mechanisms guarantee that tenants only allocate minimum shares to their non-dominant demands and maximize the share allocations on the contended resource. RRF guarantees no tenant can obtain unfair benefit from other tenants when there exists resource contention. Moreover, RRF is able to achieve some desirable properties of resource sharing, including sharing incentive, gain-as-you-contribute fairness and strategy-proofness (see Section III-A). Therefore, RRF can address all the above fair sharing problems.

We implement a proof-of-concept prototype on Xen to verify the effectiveness of RRF. We conduct a set of experiments to evaluate the allocation fairness, resource efficiency and application performance. The preliminary results show that RRF is beneficial for both cloud providers and tenants. For cloud providers, RRF improves VM density and cloud providers' revenue by 2.2X compared to the current T-shirt model. For tenants, RRF delivers better application performance and economic fairness among tenants than other fairness mechanisms [16], [29].

The remainder of the paper is organized as follows. Section II motivates the resource sharing among multi-tenant clouds and formulates the resource fair sharing problem. Section III introduces the system overview and resource allocation model. Section IV presents the RRF-based resource dynamic allocation algorithms. Section V describes the implementation details of RRF. We present the evaluation methodologies and experimental results in Section VI. We discuss the related work in Section VII and conclude in Section VIII.

II. Background and Motivation

We first briefly introduce the typical resource sharing mechanisms, including Weighted Max-Min Fairness (WMMF) [29] and Dominant Resource Fairness (DRF) [16]. Next, we present the motivation of this study by analyzing the deficiency of WMMF and DRF for multi-resource sharing.

A. WMMF and DRF

WMMF [29] is widely used to solve the problem of allocating scarce resource among a set of users. Users can have different *weights* (or *shares*) to the resource. The *share* of a user reflects the user's priority relative to other users. The WMMF algorithm defines the following three principles to allocate the resource [29]: (1) Resources are allocated in the order of increasing demands normalized by the weight. Thus, if two users have the same weights, the user with a smaller resource demand is satisfied first. (2) No user obtains a resource share larger than its demand. Thus, the over-allocated portion can be re-allocated to other users with unsatisfied demands. (3) Users with unsatisfied demands get resource shares in proportion to their weights. This policy defines how to distribute the over-allocated resources to unsatisfied users.

Formally, we have the following operational definitions for WMMF. Consider a set of users $1, 2..., n$ that have resource demands d_1, d_2, ..., d_n, associated with weights w_1, w_2, ..., w_n, respectively. Without loss of generality, we assume $d_1 \leq d_2 \leq ... \leq d_n$. Suppose the total resource capacity is *C*. The *capacity per share* becomes $\sigma = C/\sum_{i=1}^{n} w_i$. Then, we initially allocate σw_i resource to user *i*. For user i, the allocation may be more than its demand. The unused resource should be proportionally re-allocated to other users whose demands are not fully satisfied according to their weights. We continue this process until no users get more than their demands, and, if their demands are not completely satisfied, their resource allocations should be proportion to their weights. The outcome of running WMMF is that it maximizes the minimum share of a user whose demand is not completely satisfied.

DRF [16] is a generalization of max-min fairness to multiple resource types. The core idea of DRF is that the amount of resource allocated to a user should be determined by the user's share on the dominant resource type (or dominant share). DRF always satisfies the demand of users in the ascending order of dominant shares, and thus maximizes the smallest dominant share of users in a system.

B. Motivation

We consider resource fair sharing problems in multi-tenant cloud environments, where each tenant may rent several VMs to host her applications, and VMs may have multiple resource demands. By *multi-resource*, we mean resource of *different resource types*, instead of multiple units of the same resource type. In this paper, we mainly consider two resource types: CPU and main memory. A number of tenants can form a resource pool based on the opportunities of resource sharing. The VMs of these tenants then share the same resource pool with negotiated resource *shares*, which are allocated according to tenants' payment. The question is how to guarantee the fairness of resource sharing so that free-riding and lying problems are properly addressed.

To simplify multi-resource allocation, we assume each unit of resource (such as 1 Compute Unit [1] or 1 GB RAM) has its fixed share according to its market price. A study on Amazon EC2 pricing data [56] had indicated that the hourly unit cost for 1 GB memory is twice as expensive as one EC2 Compute Unit. A tenant's *asset* is then defined as the aggregate shares that she pays for. Ideally, we can informally define a kind of *economic fairness*: each tenant should try to preserve her asset in terms of aggregate multi-resource shares if she has unsatisfied resource demand.

Example 1: Assume there are three tenants, each of which has one VM. All VMs share a resource pool consisting of total 20 GHz CPU and 10 GB RAM. Each VM has initial shares for different types of resource when it is created. For example, VM1 initially has CPU and RAM shares of 500 each, simply denoted by a vector $\langle 500, 500\rangle$. The VMs may have dynamic resource demands. At a time, VM1 runs jobs with demands of 6 GHz CPU and 3 GB RAM, simply denoted by a vector $\langle 6GHz, 3GB\rangle$. The VMs' initial shares and demand vectors are illustrated in Table I. We examine whether current resource allocation models (the T-shirt model, WMMF and DRF) can guarantee resource efficiency and economic fairness.

[1] One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, according to Amazon EC2.

TABLE I: Comparison of resource allocation polices between T-shirt, WMMF and DRF.

VMs	VM1	VM2	VM3	Total
Initial Shares	<500, 500>	<500, 500>	<1000, 1000>	<2000, 2000>
Demands	<6 GHz, 3 GB>	<8 GHz, 1 GB>	<8 GHz, 8 GB>	<22 GHz, 12 GB>
T-shirt Allocation	<5 GHz, 2.5 GB>	<5 GHz, 2.5 GB>	<10 GHz, 5 GB>	actually used <18 GHz, 8.5 GB>
WMMF Allocation	<6 GHz, 3 GB>	<6 GHz, 1 GB>	<8 GHz, 6 GB>	<20 GHz, 10 GB>
WDRF dominant share	6/20 = 3/10	8/20 CPU	8/(10*2) RAM	100%
WDRF Allocation	<6 GHz, 3 GB>	<7 GHz, 1 GB>	<7 GHz, 6 GB>	<20 GHz, 10 GB>

With T-shirt Model, we allocate the total resources to tenants in proportion to their share values of CPU and memory separately. The T-shirt model guarantees that each tenant precisely receives the resource shares that the tenant pays for. However, it wastes scare resource because it may over-allocate resource to VMs that has high shares but low demand, even other VMs have unsatisfied demand. As shown in Table I, VM2 wastes 1.5 GB RAM and VM3 wastes 2GHz CPU.

We now apply the **WMMF** algorithm on each resource type. As shown in Table 1, VM1, VM2 and VM3 initially owns $25\%, 25\%, 50\%$ of total resource shares, respectively. However, VM1 is allocated with 30% of total resources, with 5% "stolen" from other VMs. Ironically, VM2 contributes 1.5 GB RAM and VM3 contributes 2 GHz CPU to other tenants. However, they do not benefit more than VM1 from resource sharing because CPU and memory resource are allocated separately. In this case, if VM1 deliberately provisions less resource than its demand and always reckons on others' contribution, then VM1 becomes a free rider. Although WMMF can guarantee resource efficiency, it cannot fully preserve tenant's resource shares, and eventually results in economic unfairness.

We also apply weighted DRF (**WDRF**) [16] to this example. Both CPU and RAM shares of VM1, VM2 and VM3 correspond to a ratio of $1:1:2$. VM1's dominant share can be CPU or memory, both equal to $\frac{6}{20}$. VM2's dominant share is CPU share as $max(\frac{8}{20}, \frac{1}{10}) = \frac{8}{20}$. For VM3, its un-weighted dominant share is memory share $\frac{8}{10}$. Its weight is twice of that of VM1 and VM2, so its weighted dominant share is $\frac{8}{10*2} = \frac{8}{20}$. Thus, the ascending order of three VM's dominant shares is VM1 < VM2 = VM3. According to WDRF, VM1's demand is first satisfied, and then the remanding resources are allocated to VM2 and VM3 based on max-min fairness. We find that VM1 is again a free rider.

In summary, the T-shirt model is *not* resource-efficient, and WMMF and DRF are *not* economically fair for multi-resource allocation. Intuitively, in a cooperative environment, the more one contributes, the more she should gain. Otherwise, the tenants would lose their sharing incentives. This is especially important for resource sharing among multiple tenants in pay-as-you-use clouds. Thereby, a new mechanism is needed to reinforce the fairness of multi-resource sharing in IaaS clouds.

III. Overview

In this section, we first describe a series of requirements that we believe any multi-resource fair sharing model for multiple tenants should satisfy, and then give an overview of our resource allocation model.

A. Resource Sharing Requirements

DRF [16] has defined a number of requirements/properties that a multi-resource sharing mechanism should satisfy. Under the context of multi-resource sharing among multiple tenants, we consider all requirements in DRF. Particularly, we highlight the following requirements.

- **A. Sharing Incentive:** Each tenant should benefit from sharing a large resource pool with others, rather than exclusively using her own fix-sized VMs.
- **B. Gain-as-you-contribute Fairness:** A tenant's gain from other tenants should be proportional to her contribution to others. The ability of a tenant to gain her unsatisfied resource demand is determined by the value (or shares) of her other resources that are contributed to other tenants, not by her original total share or total resource demands.
- **C. Strategy-proofness:** A tenant should not be able to benefit by lying about her resource demand, or by deliberately buying less resource than her real demand. This property is compatible with sharing incentive and gain-as-you-contribute fairness, because no tenants can get more resource than their shares by lying or free-riding.

B. Resource Allocation Model

We now discuss how to effectively allocate multiple resources to a set of cooperative VMs. We use the aforementioned allocation properties to guide the development of an economic fair sharing policy. In the following, we use an example to demonstrate our resource allocation model. Figure 1 shows three tenants co-located on two physical hosts. Each tenant has two VMs. In our fine-grained resource sharing model, each unit of resource is represented by a number of shares. The shares of different resources (e.g., CPU, Memory) are uniformly normalized based on their market prices [56]. To some extent, what a tenant actually purchases is resource shares instead of fix-sized resource capacity. As share of a VM reflects the VMs priority relative to other VMs, cloud providers can directly use shares as billing and resource allocation policies. The concrete design of those policies is beyond the scope of this paper. Thus, we simply define a function f_1 to translate tenants' payment into shares $payment \xrightarrow{f_1} share$, and another function f_2 to translate shares into resource capacity $share \xrightarrow{f_2} resource$. For example, in Figure 1, one compute unit and one GB memory are priced at 100 and 200 shares, respectively. Suppose VM1 is initialized with 3 compute units and 2 GB memory. Then, VM1 is allocated with total $100 \times 3 + 200 \times 2 = 700$ shares. For cloud providers, share is only used as a mediator for resource allocation by hypervisors, such as VMware proportional-share based CPU scheduler and Xen credit scheduler. Tenants can still buy VMs according to the total amount of resource demand.

Normalizing multiple resources with uniform shares pro-

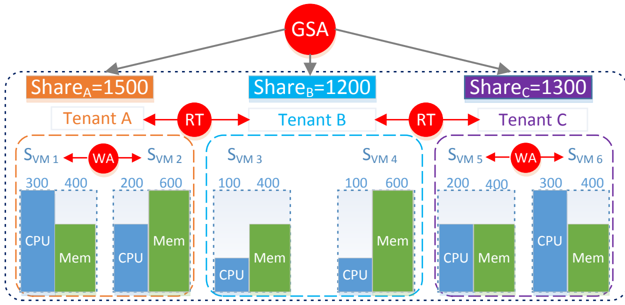

Fig. 1: Hierarchical resource allocation based on resource trading and weight adjustment.

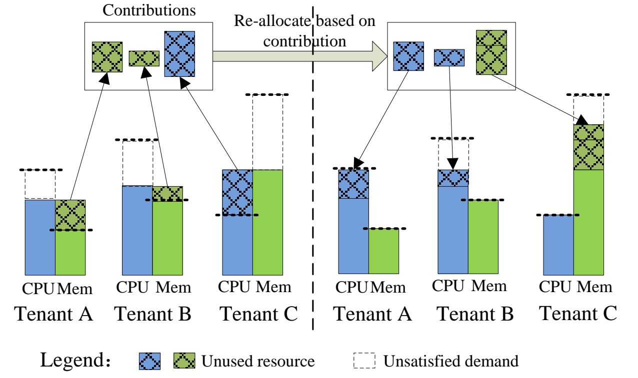

Fig. 2: Sketch of inter-tenant resource trading.

vides the advantage to facilitate resource trading and weight adjustment. It allows tenants dynamically adjust the weight of multiple resources based on their actual demands. For example, in Figure 1, tenant A may deprive 200 memory shares from VM2 and re-allocate them to VM1. Moreover, one tenant also can trade the over-allocated CPU shares for other tenants' memory shares. For example, VM1 may trade its 200 CPU shares for VM3's 100 memory shares. We thus propose dynamic resource trading between different tenants (Section IV-A), and dynamic weight adjustment among multiple VMs belonging to the same tenant (Section IV-B). Figure 1 shows the hierarchical resource allocation based on these two mechanisms. The global share allocator (GSA) first reserves capacity in bulk based on the tenants' aggregate resource demands, and then allocates shares to tenants according to their payment. The local share/resource allocator in each node is responsible for Resource Trading (RT) between tenants, and Weight Adjustment (WA) among multiple VMs belonging to the same tenant.

IV. Reciprocal Resource Fairness

We propose RRF, a new approach to multi-resource allocation that meets all the required properties described in Subsection III-A. RRF provides two complementary and hierarchical mechanisms – inter-tenant resource trading (IRT) and intra-tenant weight adjustment (IWA) to address fairness problems raised in multi-resource sharing among multiple tanents.

In the following, we consider the fair sharing model in a shared system with p types of resource and m tenants. The total system capacity bought by the m tenants is denoted by a vector Ω, i.e., $\langle\Omega_{CPU}, \Omega_{RAM}\rangle$. Each tenant i may have n VMs. Each VM j is initially allocated with a share vector $s(j)$ that reflects its priority relative to other VMs. The amount of resource share required by VM j is characterized by a demand vector $d(j)$. The resource share contributed to the other tenants is characterized by a contribution vector $c(j)$, which corresponds to the VM's initial share vector $s(j)$. Correspondingly, $s^{'}(j)$ denotes the current share vector after the resources are re-allocated.

For simplicity, we assume that resource allocation is oblivious, meaning that the current allocation is not affected by previous allocations. Thus, a VM's priority is always determined by its initial share vector $s(j)$ in each time of resource allocation.

A. Inter-tenant Resource Trading (IRT)

For multi-resource allocation, it is hard to guarantee that the demands of all resource types are nicely satisfied without waste. For example, a tenant's aggregate CPU demand may be less than her initial CPU share, but memory demand exceeds her current allocation. In this case, she may expect to trade her CPU resource with other tenants' memory resource. Thus, the question is how to trade resources of different types among tenants while guaranteeing economic fairness. RRF embraces an IRT mechanism with the core idea that a tenant's gain from other tenants should be proportional to her contribution. The only basis for resource distribution is the tenant's contribution, rather than her initial resource share or unsatisfied demand. As shown in Figure 2, the memory resource contributed by Tenant A is 2 times more than that of Tenant B, so Tenant A should receive 2 times more unused CPU resource (contributed by Tenant C) than Tenant B at first. Then, we need to check whether the CPU resource of Tenant A is over-allocated. If so, the unused portion should be re-distributed to other tenants. This process should be *iteratively* performed by all tenants because each allocation may affect other tenants' allocations. This naive approach is easy to understand but can cause unacceptable computation overhead.

We propose a work backward strategy to speed up the unused resource distribution. For each type of resource, we divide the tenants into three categories: contributors, beneficiaries whose demands are satisfied, and beneficiaries whose demands are unsatisfied, as shown in Figure 3. Tenants in the first two categories are *directly* allocated with their demands exactly, and tenants in the third category are allocated with their initial share plus a portion of contributions from the first category. However, A challenging problem is how to divide the tenants into three categories efficiently. Algorithm 1 describes the sorting process by using some heuristics.

Let vectors $D(i), S(i), C(i)$ and $S^{'}(i)$ denote the total demand, initial share, contribution and current share of the tenant i, respectively. Correspondingly, let $D_k(i), S_k(i), C_k(i)$ and $S_k^{'}(i)$ denote her total demand, initial share, contribution and current share of resource type k, respectively. We consider a scenario where m tenants share a resource pool with capacity Ω, with resource contentions ($\sum_{i=1}^{m} D(i) \geq \Omega$). Our algorithm first divides the total capacity on the basis of each tenant's initial share, and then caps each tenant's allocation at her total demand. Actually, each tenant will receive her initial total share $S(i)$, and then her total contribution becomes $C(i) = S(i) - D(i)$ (if $S(i) > D(i)$). For resource type k ($1 \leq k \leq p$), the unused resource $C_k(i)$ is re-distributed to other unsatisfied tenants in the ratio of their total contributions.

Algorithm 1 shows the pseudo-code for IRT. We first

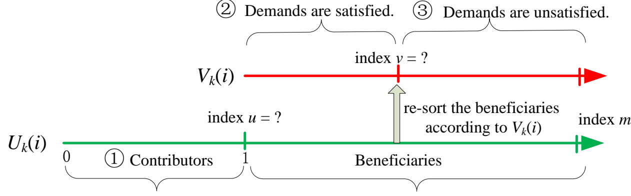

Fig. 3: Sketch of IRT algorithm.

calculate each tenant's total contribution $\Lambda(i)$ (Lines 6-8). To reduce the complexity of resource allocation, for each resource type k, we define the normalized demand of tenant i as $U_k(i) = D_k(i)/S_k(i)$, and re-index the tenants so that the $U_k(i)$ are in the ascending order, as shown in Figure 3. Then, we can easily find the index u so that $U_k(u) < 1$ and $U_k(u+1) \geq 1$. The tenants with index $[1, \cdots, u]$ are contributors and the remaining are beneficiaries. For tenants with index $[u+1, \cdots, m]$ $(U_k(i) \geq 1)$, we define the ratio of unsatisfied demand of resource type k to her total contribution as $V_k(i) = (D_k(i) - S_k(i))/\sum_{k=1}^{p} C_k(i)$ (Lines 12-13), and re-index these tenants according to the ascending order of $V_k(i)$, as shown in Figure 3. Thus, tenants with index $[1, \cdots, u]$ are ordered by $U_k(i)$ while tenants with index $[u+1, \cdots, m]$ are ordered by $V_k(i)$. The demand of tenants with the largest index will be satisfied at last. We need to find a pair of successive indexes $v, v+1, (v \geq u)$, so that the share allocations of tenants with index $[1, \cdots, v]$ are capped at their demands, and the remaining contribution $\Psi_k = \Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)$ is distributed to tenants with index $[v+1, \cdots, m]$ in proportion to their total contributions. Some heuristics can be employed to speed up the index searching. First, searching should start from index $u+1$ because tenants with index $[1, \cdots, u]$ are contributors. Second, we can use binary search strategy to find two successive indexes $v, v+1, (v \geq u)$ till the demand of tenant with index v is satisfied by receiving her proportion of other tenant's contribution, while the demand of tenant with index $v+1$ still cannot be satisfied. Namely, the following inequality (1) and (2) must be satisfied:

$$S_k(v) + \frac{\left(\Omega_k - \sum_{i=1}^{v-1} D_k(i) - \sum_{i=v}^{m} S_k(i)\right) \times \Lambda(v)}{\sum_{i=v}^{m} \Lambda(i)} \geq D_k(v) \tag{1}$$

$$S_k(v+1) + \frac{\left(\Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)\right) \times \Lambda(v+1)}{\sum_{i=v+1}^{m} \Lambda(i)} < D_k(v+1) \tag{2}$$

where the expressions in the big parenthesis represent the remaining contributions that will be re-distributed to tenants with unsatisfied demands. Once the index v is determined, we can calculate the remaining contribution. The tenants with index $[1, \cdots, v]$ receive shares capped at their demands (Lines 16-17), and the tenants with index $[v+1, \cdots, m]$ receive their initial shares plus the remaining resource in proportion to their contributions (Lines 19-20).

We illustrate the IRT algorithm using an example in which total capacity of 30 GHz CPU and 15 GB RAM are allocated to 4 VMs. Assume one GHz CPU and one GB memory are

Algorithm 1 Inter-tenant Resource Trading (IRT)

Input: $D = \{D(1), ..., D(m)\}, S = \{S(1), ..., S(m)\}, \Omega$
Output: $S' = \{S'(1), ..., S'(m)\}$
Variables: $[i, C(i), \Lambda(i), U(i), V(i), \Psi(i)] \leftarrow 0$

```
1:  for resource_type k = 1 to p do
2:    for Tenant i = 1 to m do
3:      /*Allocate each tenant (i) her initial share S(i) */
4:      S'_k(i) ← S_k(i)
5:      U_k(i) ← D_k(i)/S_k(i)
6:      if S_k(i) ≥ D_k(i) then
7:        C_k(i) ← S_k(i) − D_k(i)
          /*Calculate tenant(i)'s total contribution Λ(i) on all type of resource */
8:        Λ(i) ← Λ(i) + C_k(i)
9:  for resource_type k = 1 to p do
10:   Sort U_k(i) in ascending order;
11:   Find the index u so that U_k(u) < 1 ≤ U_k(u+1)
12:   for Tenant i = u+1 to m do
13:     V_k(i) ← (D_k(i) − S_k(i))/Λ(i)
14:   Sort V_k(i) in ascending order;
15:   Find the index v using binary search algorithm so that Equation (1) and (2) are satisfied;
16:   for Tenant i = 1 to v do
17:     S'_k(i) ← D_k(i)    /*share is capped by demand*/
      /*Ψ is the remaining contributions for re-allocation*/
18:   Ψ_k ← Ω_k − Σ_{i=1}^{v} D_k(i) − Σ_{i=v+1}^{m} S_k(i)
19:   for Tenant i = v+1 to m do
20:     S'_k(i) ← S_k(i) + (Ψ_k × Λ(v+1)) / (Σ_{i=v+1}^{m} Λ(i))
```

priced at 100 and 200 shares, respectively. The VMs' resource demand, initial shares and demanded shares are illustrated in Lines 2–4 of Table II. We first calculate each VM's contribution, as shown in Line 5 of Table II. The following lines show the details of CPU and memory allocation based on IRT algorithm. For CPU, VM3 and VM4 are the contributors while VM1 and VM2 are beneficiaries. However, as VM1 contributes nothing to others, VM2 receives all unused CPU shares (200+100) from VM3 and VM4. For memory, only VM2 contributes 300 unused memory shares and the other VMs are beneficiaries. As VM3 and VM4 contribute 200 and 100 shares of CPU resource to the group, VM3 and VM4 receive $\frac{200}{100+200}$ and $\frac{100}{100+200}$ of total total 300 unused memory shares (i.e., 200 and 100 shares), respectively. VM1 again receives nothing as it does not contribute anything to others.

Form this example, we have witnessed that unused resources are properly re-distributed based on VMs' contributions and free riders can not benefit from others. IRT always tries to prevent tenants from share losing and thus guarantee economic fairness.

Although IRT algorithm is designed for multi-resource trading between multiple tenants, it is also applicable to resource trading among VMs, oblivious to the VMs' attribution. However, resource trading at VM-level sometimes may not guarantee economic equity, and may not preserve tenant's asset completely. Therefore, we limit resource trading to multiple tenants, and propose dynamic weight adjustment between VMs belonging to the same tenant to further preserve the tenant's asset.

B. *Intra-tenant Weight Adjustment (IWA)*

In the cloud, a tenant usually needs more than one VM to host her applications. Workloads in different VMs may have dynamic and heterogeneous resource requirements. Thus, dynamic resource flows among VMs belonging to the same

TABLE II: An example of IRT algorithm.

VMs	VM1	VM2	VM3	VM4	Total
Resource Demand	<6 GHz, 3 GB>	<8 GHz, 1 GB>	<8 GHz, 8 GB>	<9 GHz, 6 GB>	<31 GHz, 17 GB>
Initial Shares	<500, 500>	<500, 500>	<1000, 1000>	<1000, 1000>	<3000, 3000>
Demanded Shares	<600, 600>	<800, 200>	<800, 1600>	<900, 1200>	<3100, 3600>
Contributions	<0, 0>	<0, 300>	<200, 0>	<100, 0>	<300, 300>
CPU allocation	**Sort by** $U_{CPU}(VM_i)$	VM3 (0.8) , VM4 (0.9) , VM1 (1.1) , VM2 (1.3)			
	Sort by $V_{CPU}(VM_i)$	VM3 (-) , VM4 (-) , VM2 ($\frac{800-500}{300} = 1$) , VM1 ($\frac{600-500}{0} = +\infty$)			
	CPU share $S_{CPU}(VM_i)$	VM3 (800) , VM4(900) , VM2 (500+(200+100)=800) , VM1 (500)			
Memory allocation	**Sort by** $U_{mem}(VM_i)$	VM2 (0.4) , VM1 (1.1) , VM4 (1.2) , VM3 (1.4)			
	Sort by $V_{mem}(VM_i)$	VM2 (-) , VM4 ($\frac{1200-1000}{100} = 2$) , VM2 ($\frac{1600-1000}{200} = 3$) , VM1 ($\frac{600-500}{0} = +\infty$)			
	Mem share $S_{mem}(VM_i)$	VM2 (200) , VM4 ($1000 + \frac{300\times100}{100+200} = 1100$) , VM3 ($1000 + \frac{300\times200}{100+200} = 1200$) , VM1 (500)			
Shares Allocation	<500, 500>	<800, 200>	<800, 1200>	<900, 1100>	<3000, 3000>
Resource Allocation	<5 GHz, 2.5 GB>	<8 GHz, 1 GB>	<8 GHz, 6 GB>	<9 GHz, 5.5 GB>	<30 GHz, 15 GB>

tenant can prevent loss of tenant's asset. We propose dynamic intra-tenant weight adjustment among VMs belonging to the same tenant. We allocate *share* (or weight) for each VM using a policy similar to WMMF. For each type of resource, we first reset each VM's current weight to its initial share. However, if the allocation made to a VM is more than its demand, its allocation should be capped at its real demand, and the unused share should be re-allocated to its sibling VMs with unsatisfied demands. In contrast to WMMF that re-allocates the unused resource in proportion to VMs' share values, we re-allocate the excessive resource share to the VMs in the ratio of their unsatisfied demands. Note that, once a VM's resource share is determined, the resource allocation made to the VM is simply determined by the function $share \xrightarrow{f_2} resource$.

Algorithm 2 shows the pseudo-code for IWA. A tenant with n VMs is allocated with total resource share S. Note that S is a global allocation vector which corresponds to the output of Algorithm 1. Thus, Algorithm 2 is performed accompanying with Algorithm 1. For each tenant, we first calculate her total unsatisfied demand and total remaining capacity for re-allocation, respectively (Lines 2 to 6), and then distribute the remaining capacity to unsatisfied VMs in ratio of their unsatisfied demands (Lines 7 to 11). As VM provisioning is constrained to physical hosts' capacity, it is desirable to adjust weights of VMs on the same physical node, rather than across multiple nodes. In practice, we execute the IWA algorithm only on each single node.

Algorithm 2 Inter-tenant Weight Adjustment (IWA)

Input: $d = \{d(1), ..., d(n)\}, s = \{s(1), ..., s(n)\}, S$
Output: $s^{'} = \{s^{'}(1), ..., s^{'}(n)\}$
Variables: $[j, \Gamma, \Phi] \leftarrow 0$
/* Allocate initial share $s(j)$ to each VM(j) */
1: $\Phi \leftarrow S - \sum_{j=1}^{n} s(j)$ /*Calculate the difference of initial total share and new allocated capacity */
2: **for** $VM\ \ j = 1$ **to** n **do**
3: **if** $d(j) \geq s(j)$ **then**
4: $\Gamma \leftarrow \Gamma + (d(j) - s(j))$ /*total unsatisfied demand*/
5: **else**
6: $\Phi \leftarrow \Phi + (s(j) - d(j))$ /*total remaining capacity*/
/* distribute remaining capacity to VMs with unsatisfied demand */
7: **for** $VM\ \ j = 1$ **to** n **do**
8: **if** $d(j) \geq s(j)$ **then**
9: $s^{'}(j) \leftarrow s(j) + \frac{d(j)-s(j)}{\Gamma} \times \Phi$
10: **else**
11: $s^{'}(j) \leftarrow d(j)$

C. Analysis of Fairness Properties

This subsection makes comparison between RRF, WMMF and DRF on the properties presented in Section III-A.

Theorem 1. *All WMMF-derived algorithms including DRF and RRF satisfy the sharing incentive property.*

Sketch of Proof: According to max-min fairness, each tenant should not receive more resource than her demand if other tenants have not received their demands. This rule ensures that there is no wasted capacity. At any time, if the demand of tenant i is less than her initial share (i.e.,$D(i) < S(i)$), she receives her demand and the unused portion should be distributed to other tenants with unsatisfied demands. If $D(i) \geq S(i)$, she receives the same amount of resource as that she pays for. She still has opportunities to gain more resource from other contributors. We have seen that all tenants benefit from sharing without wasting resource. Thus, all WMMF-derived algorithms satisfy the sharing incentive property. ■

Theorem 2. *Both WMMF and DRF violate gain-as-you-contribute fairness property that RRF naturally satisfied.*

Sketch of Proof: Recall that RRF leverages a resource trading mechanism to preserve tenants' unused resource and to meet the unsatisfied demands of other resource types. For each tenant, the resource gained from trading is determined by the ratio of her contribution to that of others, rather than her initial resource share or current demand. In contrast, WMMF and DRF always try to maximize the minimum share and the smallest dominant share of a tenant, respectively, they both satisfy the smallest demand first. This is not fair for tenants that have large contributions and also large unsatisfied demands. Consider Example 1, both WMMF and DRF first satisfy VM1's unsatisfied demand, even it doesn't contribute anything to the other two VMs. To this end, neither WMMF nor DRF satisfy gain-as-you-contribute fairness. ■

Theorem 3. *RRF satisfies strategy-proofness property, while WMMF and DRF cannot satisfy it completely.*

Sketch of Proof: Recall that resource allocation of RRF is not directly determined by tenants' demands, all tenants cannot benefit by lying about their resource demands. In fact, lying may lead to less benefit from other tenants. Consider the following example, suppose tenant *A* needs 6 GHz CPU and 3 GB RAM, and its initial shares can supply 4 GHz CPU and 4 GB RAM. If tenant *A falsely* claims the demand to be 8 GHz CPU and 5 GB RAM, she will receive 4 GHz CPU

and 4 GB RAM only. However, 1 GB RAM is wasted and her CPU demand is not satisfied yet. In contrast, if she claims her demands honestly, there are opportunities for her to trade the unused 1 GB RAM for more CPU resource. In addition, RRF is also immune to free-riding as "no contribution, no gain".

As to WMMF, it distributes the unused resource of a tenant based on other tenants' initial share, so it is also immune against lying. However, WMMF cannot prevent tenants from free-riding, as demonstrated by VM1 in Example 1.

As to DRF, consider Example 1, if VM1 lies its demands to be $< 7GHz, 3.5GB >$, DRF also satisfies these demands first because its dominant share becomes $\frac{7}{20}$ but is still less than the dominant shares of VM2 and VM3 (both equal to $\frac{8}{20}$). VM1 benefits from lying and thus violates strategy-proofness. Also, DRF cannot prevent tenants from free-riding, as demonstrated by VM1 in Example 1. ■

Table III summarizes the fairness properties presented in Section III-A that are satisfied by WMMF, DRF, and RRF. WMMF and DRF are *not* suitable for pay-as-you-use clouds due to the lack of support for two important desired properties, whereas RRF can achieve all these properties.

TABLE III: Properties of WMMF, DRF, and RRF.

Property	Allocation Policy		
	WMMF	DRF	RRF
Sharing Incentive	√	√	√
Gain-as-you-contribute Fairness			√
Strategy-proofness			√

V. Prototype on Xen

To verify the effectiveness of RRF, we implement a proof-of-concept prototype on Xen, which is supported by many cloud stacks and public cloud providers [15]. RRF is also applicable for other common-used hypervisors and cloud platforms such as KVM and VMware, because they all support dynamic resource allocation techniques such as proportional-share based CPU scheduler and memory ballooning.

In the prototype, we have implemented a VM placement algorithm to facilitate resource sharing. Our VM grouping algorithm exploits resource multiplexing among different VMs to improve the sharing opportunity. On the spatial dimension, VMs with different dominant resource requirements can be grouped based on the *skewness* of multiple resources. The skewness means the similarity of two VMs or servers quantified by Pearson's Correlation Coefficient (PCC) [31]. The multi-resource provisioning can be formulated as a multi-dimensional bin packing problem [6]. We approximate its optimal solution by always placing VMs to servers with reverse skewness (PCC value). More details can be found in Appendix A of our technical report [32].

There are also some other important details for a complete system implementation, such as resource demand prediction, resource allocator, resource pool scaling and load balancing. Here, we mainly describe the resource allocator in the following, and present other details in Appendix A of our technical report [32].

Xen and other hypervisors have provided several techniques for fairly allocating CPU time and memory to VMs [7], [12], [53]. For CPU resource allocation, The open-source Xen provides several schedulers for fair sharing and allocation of CPU resource in different scenarioes [12]. We use the credit scheduler [12] in our prototype system as it is most commonly used in real deployments. Xen hypervisor provides a convenient interface to configure VMs' CPU weight (or share). The credit scheduler then allocates CPU cycles to each VM in proportional to its weight (share). Moreover, the credit scheduler supports a non-workconserving mode, in which VMs CPU time can be capped at their demands. Those features are perfectly fit for our resource allocation requirements.

For memory allocation, Xen hypervisor also provides an interface to dynamically adjust each VM's memory capacity through ballooning mechanism [7], [53]. One limitation of ballooning mechanism is that it can't extend a VM's memory footprint beyond its original maximum allocation, so dynamic memory adjustment is constrained to the VMs' max_memory configuration. To take full advantage of ballooning, the hypervisor needs to create a VM with max_memory equal to the host's maximum memory capacity, and then "inflates" the guest balloon driver to shrink the VM's memory capacity according to the tenant's provisioned size. When the VM needs more memory, the hypervisor "deflates" the balloon driver to add memory to the VM. In our previous work, we had developed the memory hotplugging technique for VMs [33] to overcome the limitation of ballooning. Hypervisor can freely add/remove memory to/from VMs without suffering the constraint of memory cap. However, all those actions are up to the Xen hypervisor to control what VMs can decrease their memory footprints, and vice versa. In our system, RRF allocation algorithm makes the decision to determine how many memory shares a VM should be allocated in each time of memory adjustment.

Recently, there have been some Software-as-a-Service virtualization technologies such as Docker [14] and LXC [34]. They are lightweight resource containers that provide several promising features, such as portability, more efficient scheduling and resource management, and less virtualization overhead. We conjuncture that our RRF algorithm is also applicable for the container-based resource fair sharing.

Security issue is likely to be a major concern in any shared systems. Although multiple tenants share a large resource pool in our system, each unit of resource is exclusively used by users at any instantaneous times. For memory resource, RRF uses ballooning or hotplugging to adjust each VM's memory allocation. No memory pages are shared by different VMs at the same time. To this end, RRF may not causally result in a security issue.

VI. Evaluation

In this section, we evaluate RRF in terms of fairness of multi-resource allocation, improvement of application performance and VM density, and performance overhead.

A. Experimental Setup

We have implemented RRF on Xen 4.1. We deploy our prototype in a cluster with 10 nodes. Each node is equipped with six quad-core Intel Xeon X5675 3.07 GHz processors, 24 GB DDR3 memory and 1 Gbit Ethernet interface. The host machines are with 64-bit Linux RHEL 5 distribution and the hypervisor is Xen 4.1 with Linux 2.6.31 kernel. The VMs run in para-virtualization mode, with the OSes the same distribution Linux as the hosts. Each host is configured with 2 CPU cores and 1 GB RAM, and the remaining cores are allocated to the VMs. Each VM is configured with 4 vCPUs, while its initial share value is set according to the workload's

average demand. Note that Xen credit scheduler allocates CPU cycles to each VM in proportional to its weight (shares). Two VMs with the same amount of shares should get the same amount of CPU cycles, regardless of the number of vCPUs in each VM.

We use the following workloads with diversifying and variable resource requirements.

TPC-C: it is an on-line transaction processing (OLTP) benchmark [52]. We use a public benchmark DBT-2 as clients, to simulate a complete computing environment where a number of users execute transactions against a database. The clients and MySQL database server run in two VMs separately. We assume these two VM belong to the same tenant. We configure 600 terminal threads and 300 database connections. The benchmark shows a irregular on-off load pattern on CPU demands, as shown in Figure 4. We evaluate its application performance by throughput (transactions per minute).

RUBBoS (Rice University Bulletin Board System) [45]: it is a typical multi-tier web benchmark. RUBBoS simulates an on-line news forum like slashdot.org. We deploy the benchmark in a 3-tier architecture using Apache 2.2, Tomcat 7.0, and MySQL 5.5. Each tier runs in a single VM. The three VMs belong to the same tenant. Some other VMs are used to run client workload generators. We configure 500 and 1000 concurrent users to alternately access the forum so as to simulate a cyclical workload pattern. The workload generates considerable CPU and memory load on the VM running DB tier. We evaluate its application performance using the request response time.

Kernel-build: We compile Linux 2.6.31 kernel in a single VM. It generates a moderate and balanced load of CPU and memory.

Hadoop: We use Hadoop WordCount micro-benchmark to setup a virtual cluster consisting of 10 worker VMs and one master VM. The WordCount program sums up the number of each word from a 60 GB dataset generated by *RandomTextWriter* program contained in the Hadoop distribution. Each worker VM run 800 map jobs and followed by 400 reduce jobs on average. The map stage take almost 95% of the total execution time. The WordCount workload is CPU and memory bound with regular and small fluctuations of resource demands.

On our test bed, we consider multiple tenants sharing the cluster. Each tenant only runs one kind of the above workloads. We continuously launch the tenants' applications to the cluster one by one until no room to accommodate any more applications. All VMs are placed on servers based on our tenant grouping algorithm [32]. To study the relationship between application performance and the amount of resource provisioned, we first use an off-line profiling scheme to measure applications' demands, as shown in Table IV, and configure the initial share of tenant (i) based on a provisioning coefficient,

$$\alpha = S(i)/\overline{D(i)},$$

which reflects the ratio of initial resource share to the average demand. In our experiments, we specify the values of 1 CPU core (3.07 GHz) and 1 GB RAM to be 300 and 200 shares, respectively. This setting accords with the ratio of CPU price to memory price in Amazon EC2 clouds [56].

The dynamic resource allocation algorithm is periodically executed at each node (domain 0). By default, the period (or *window size*) is set to 5 seconds for the purposes of fine-grained resource sharing. We evaluate RRF by comparing the following alternative approaches:

TABLE IV: Workloads' aggregated CPU and memory demands.

App	Average Demand	Peak Demand
TPC-C	<1.4 core, 2.2 GB>	<3.2 core, 2.8 GB>
RUBBoS	<8.1 core, 4.6 GB>	<16.5 core, 8.4 GB>
Kernel-build	<1.0 core, 0.6 GB>	<1.5 core, 0.8 GB>
Hadoop	<11.5 core, 10.3 GB>	<12.5 core, 12.6 GB>

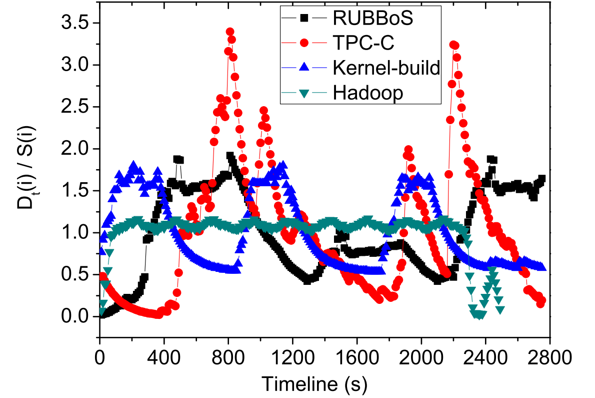

Fig. 4: For each workload, the ratio of total resource demand to total initial share, namely $D_t(i)/S(i)$, varies with time.

- **T-shirt (static)**: Workloads are running in VMs with static resource provision. It is the current resource model adopted by most IaaS clouds [18].
- **WMMF (Weighted Max-Min Fairness)**: WMMF [29] is used to allocate CPU and Memory resources to each VM separately.
- **DRF (Dominate Resource Fairness)**: DRF [16] is used to allocate multiple resources to each VM.
- **IWA (Intra-tenant Weight Adjustment)**: We conduct only weight adjustment for VMs belonging to the same tenant, without considering inter-tenant resource trading. This is to assess the individual impact of inter-tenant resource trading.
- **RRF (IRT + IWA)**: We conduct hierarchical resource allocation using both inter-tenant resource trading and intra-tenant weight adjustment.

B. Dynamic Resource Sharing

To understand the resource allocation mechanism of RRF, we track total resource demands and allocations of those workloads in 45 minutes. All VMs' capacity are provisioned based on their average demands (i.e., $\alpha = 1$). We use a scenario where the four workloads are co-located on a single host to illustrate the dynamic resource sharing. In Figure 4, the curves represent the ratio of instantaneous resource demand (i.e., the sum of CPU and memory shares) of each workload to its average demand (initial resource share). Figure 4 shows that RUBBOS, TPC-C and Kernel-build have significant dynamic of resource demand over time, while Hadoop shows relatively stable resource demand. Particularly, we observe that the total resource demand exceed the server's capacity during the period of 320s–1100s, and thus there is resource contention between these workloads. In other periods, the server can successfully

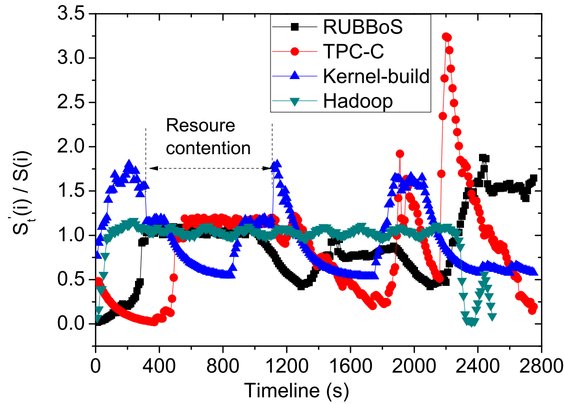

Fig. 5: For each workload, the ratio of total resource allocation to total initial share, namely $S_t'(i)/S(i)$, varies with time. The detailed allocations are based on RRF algorithm.

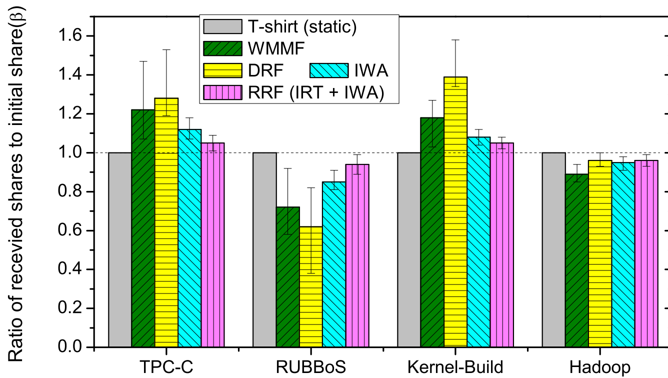

Fig. 6: Comparison of economic fairness of several resource allocation schemes.

satisfy the resource demand of each workload.

Figure 5 shows the allocation details for each workloads over time. In the period of 320s–1100s, we see that RRF achieves balanced resource allocations for RUBBoS, TPC-C and Hadoop. To some extent, the allocations imply the degree of economic fairness. As Kernel-build is over-allocated in this period, it contributes a small amount of resource to other workloads. In the other periods, each workload is allocated with its demand because the aggregated demand is less than the server's capacity.

C. Results on Fairness

In general, in a shared computing system with resource contention, every tenant wants to receive more resource or at least the same amount of resource than that she buys. We call it *fair* if a tenant can achieve this goal (i.e., sharing benefit). In contrast, it is also possible the total resource a tenant received is less than that without sharing, which we call *unfair* (i.e., sharing loss). This is because the resource exchanging may not equivalent, depending on the VM's local ecosystem. To evaluate the fairness of RRF, we define the economic fairness degree $\beta(i)$ for tenant i in a time window T as follows:

$$\beta(i) = \frac{\sum_{t=1}^{T} S_t'(i)}{T \times S(i)},$$

It represents the ratio of average resource share received to the tenant's initial share in the time window T, or more exactly speaking, it denotes the ratio of resource value to tenant's payment. $\beta(i) = 1$ implies absolute economic fairness. $\beta(i) > 1$ implies the tenant benefits from resource sharing. In contrast, $\beta(i) < 1$ implies the tenant loses her asset.

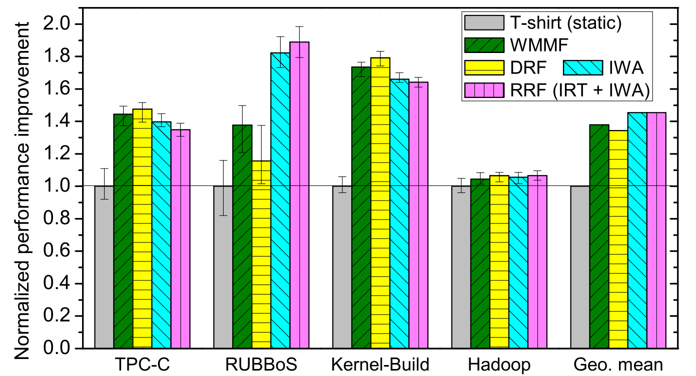

Fig. 7: Comparison of application performance improvement of several resource allocation schemes.

Figure 5 has illustrated a kind of fairness on resource allocation at instantaneous times. In the following, we evaluate the economic fairness of different schemes for the four workloads in a long period of time. All VMs' capacity are provisioned based on their average demands (i.e., $\alpha = 1$). Figure 6 shows the comparison of economic fairness of different resource allocation schemes. Each bar shows the average result of the same workload run by multiple tenants. Overall, RRF achieves much better economic fairness than other approaches. Specifically, RRF leads to smaller difference of β between different applications, indicating 95% economic fairness (geometric mean) for multi-resource sharing among multi-tenants.

We make the following observations.

First, the T-shirt (static) model achieves 100% economic fairness as VMs share nothing with each other. However, it results in the worst application performance for all workloads, as shown in Figure 7.

Second, both WMMF and DRF show significant differences on β for different workloads. As all WMMF-based algorithms always try to satisfy the demand of smallest applications first, both kernel-build and TPC-C gain more resources than their initial shares. This effect is more significant for DRF if the application shows tiny skewness of multi-resource demands (such as kernel-build). DRF always completely satisfies the demand of these applications first. Thus, DRF and WMMF are susceptible to application's load patterns. Applications with large skewness of multi-resource demands usually lose their asset. Even for a single resource type, large deviation of resource demand also lead to distinct economic unfairness.

Third, workloads with different resource demand patterns show different behavior in resource sharing. RUBBoS has a cyclical workload pattern, its resource demand shows the largest temporal fluctuations. When the load is below its average demand, it contributes resource to other VMs and thus loses its asset. However, when it shows large value of $D_t(i)/S(i)$, its demand always can not be fully satisfied if there exists resource contention. Thus, RUBBoS has smaller value of β than other applications. For TPC-C and Kernel-build, although they show comparable demand deviation and skewness with RUBBoS, they has much more opportunities to benefit from RUBBoS because their absolute demands are

much smaller. For Hadoop, although it requires a large amount of resource, it demonstrates only a slight deviation of resource demands, and there is few opportunities to contribute resource to other VMs (except in its *reduce* stage).

Fourth, inter-tenant weight adjustment (IWA) allows tenants properly distribute some VMs' spare resource to their sibling VMs in proportional to their unsatisfied demands. It guarantees that tenants can effectively utilize their own resource. Inter-tenant resource trading can further preserve tenants' asset as each tenant always tries to maximize the value of her spare resource. In addition, RRF is immune to free-riding.

We also studied the impact of different α values. For space limitation, we omit the figures and briefly discuss the results. When we decrease the provisioning coefficient α (i.e., reduce the resource provision of each application), the β of all applications approach to one. In contrast, a larger α value leads to a decrease of β. That means applications preserve their resource when there exist intensive resource contention, but lose their asset of resource over-provisioned. Nevertheless, a larger value of α implies better application performance.

D. Improvement of Application Performance

Figure 7 shows the normalized application performance for different resource allocation schemes. All schemes provision resource at applications' average demand ($\alpha = 1$). When sharing is not enabled, i.e., T-shirt (static), all applications show the worst performance, we thus refer the T-shirt model as a baseline. All applications show performance improvement due to resource sharing. For RUBBoS, RRF leads to much more application performance improvement than other schemes. The reason behind this is that RRF provides two mechanisms (IRT + IWA) to preserve tenants' asset, and thus RRF allow RUBBoS reveive more resource than other schemes, as shown in Figure 6. For other workloads, RRF is also comparable to the other resource sharing schemes. In summary, RRF achieves 45% performance improvement for all workloads on average (geometric mean). DRF achieves the best performance for Kernel-build and TPC-C, but achieves very bad performance for RUBBoS. It shows the largest performance differentiation for different workloads. The reason is that DRF always tends to satisfy the demand of the application with smallest dominant share, and thus applications that have resource demand of small sizes or small skewness always benefit more from resource sharing. In contrast, Hadoop shows small performance differentiation between different allocation schemes because of its rather stable resource demands.

E. VM Density and Tenant Cost

VM density is a very important metric to evaluate cloud resource efficiency. Higher VM density usually implies a higher resource utilization and more revenue for cloud providers. Our experiments demonstrate that RRF achieves desirable VM density by fine-grained resource sharing among different tenants. Figure 8 shows the geometric mean of normalized applications performance and VM density with varying α values. $\alpha = \alpha^*$ denotes all VMs are provisioned at its peak demands. We refer the configuration α^* as a baseline for comparison. RRF and other dynamic allocation schemes always show much better application performance than the T-shirt model due to resource sharing. In general, the VM density can be improved by α^*/α. Particularly, when α is equal to one, in comparison with provisioning resource at peak demand (i.e., $\alpha = \alpha^*$), RRF

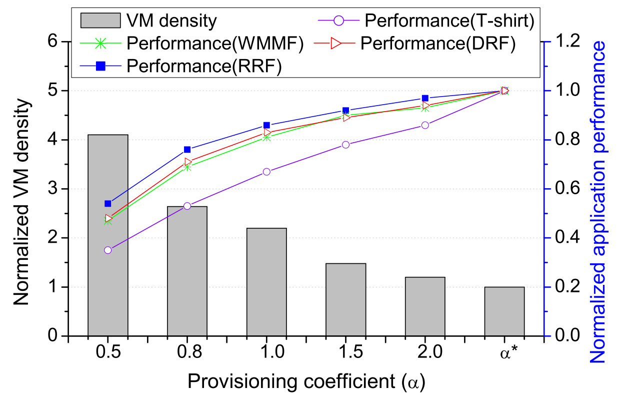

Fig. 8: Application performance corresponds to VM density.

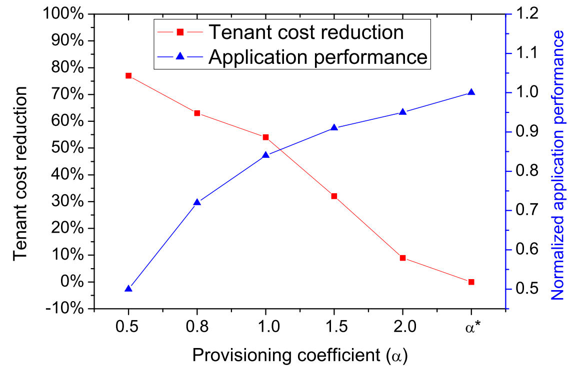

Fig. 9: The tradeoff between tenant cost reduction and application performance, compared to T-shirt model.

can significantly improve VM density by a factor of 2.2 at the expense of around 15% performance penalty. This allows cloud providers to make the tradeoff between resource efficiency and quality of service. We conjecture that the cloud provider can serve more tenants and increase revenue due to higher VM density.

From the perspective of tenants, resource sharing implies significant cost saving. Figure 9 shows the tenant cost reduction and application performance with increasing α values. Tenant cost is reduced by $(1-\alpha/\alpha^*)$ as tenant only provisions α/α^* resource compared to T-shirt model. When α is equal to one, RRF can reduce up to 55% resource cost, while the application performance degradation is less than 15%. This observation can guide tenants to make tradeoff between application performance and resource cost.

F. Runtime Overhead

We study the runtime overhead caused by dynamic resource allocations in different window sizes. As RRF is deployed at each physical node, we demonstrate the results of 10 VMs on a node coordinated by RRF in decreasing window sizes (Figure 10). RRF causes reasonable CPU load on the host machine (domain 0) even when the window size is 5 seconds. We also measure the performance overhead of VMs due to resource demand prediction in RRF. We can see that the overhead is negligible relative to the gain from resource sharing.

VII. Related Work

We review the related work in the following two categories.

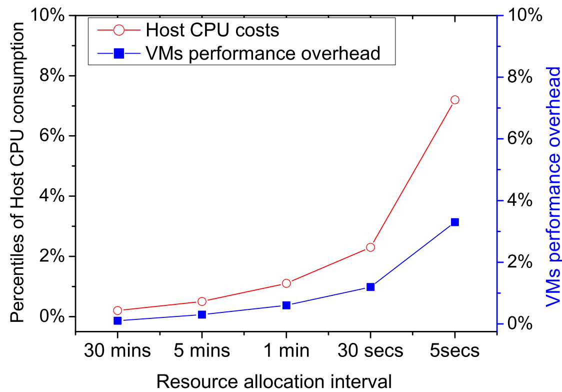

Fig. 10: Performance overhead of RRF varies with the frequency of resource dynamic allocations.

Resource sharing and provisioning. Despite the vast amount of work on fair resource allocation in clusters or girds, including Grid5000-OAR [21], Moab/Torque [40], CCS [10] and Punch [28], we focus on the related work in virtualization environments. Current hypervisors such as VMware and Xen have provided a number of techniques to facilitate fine-grained resource multiplexing, such as memory ballooning/hotplugging [33], page-sharing [7], [53], and CPU & I/O scheduling [12], [22]. There have been a number of studies focusing on VM resource multiplexing strategies to improve resource utilization. Virtual-putty [50] exploited affinities and conflicts between co-located VMs to improve the host resource utilization. Meng *et al.* [39] proposed a joint-VM provisioning strategy to consolidate multiple VMs based on the estimate of their total resource requirement. v-Bundle [25] offers elastic exchange of network resource among multiple VMs belonging to the same tenant. Those studies have not considered the fairness issues in resource sharing.

Most IaaS clouds follow the T-shirt model [18]. In the T-shirt model, the challenging problem is resource scaling in an on-demand manner [3]. A number of experimental studies [19], [54] have been conducted to study resource and performance interference on real clouds. CloudScale [47] is a trace-driven elastic resource scaling system that performs online resource demand prediction and provides adaptive padding and reactive estimation errors correction to reduce SLA violations. Furthermore, several other trace-based approaches [8], [42], [57], [59] are developed for demand prediction and capacity provisioning. These approaches are complementary to RRF, and can help tenants to estimate the total amount of purchase. RRF is also complementary to these techniques for cost saving, in the sense that it enables fine-grained sharing among VMs within one tenant and among multiple tenants.

Fairness of resource sharing: Fairness is a major concern of any shared computer system. (Weighted) max-min fairness is one of the most widely used fairness mechanisms [29]. Many data center or cluster schedulers supports max-min fairness or its extensions, such as Hadoop's Fair Scheduler [23], Quincy [27], Mesos [24], and Choosy [17]. Quincy [27] achieves fairness of scheduling concurrent distributed jobs by formulating the problem as a min-cost flow problem. Mesos [24] achieves fair resource sharing by using DRF [16]. Choosy [17] is job scheduler that provides Constrained Max-Min Fairness, an extension of max-min fairness that supports hard job placement constraints. Delay scheduling [58] and fair completion scheduler [55] address fairness of resource allocation for MapReduce jobs. Our previous work proposed Long-Term Resource Fairness (LTRF) for the single-type resource allocation and developed a long-term job scheduler [51] for YARN platform [2]. These job schedulers all address the fairness problems at the level of fixed-size partitions of nodes, called slots. In contrast, RRF achieves fairness of multi-resource allocation at fine-grained resource level (i.e. CPU and memory).

Many studies have considered the fairness of sharing a single-type resource, such as I/O or network bandwidth. Pisces [49] is a systemic implementation of max-min fairness in multi-tenant shared key-value storage system. mclock [22] supports proportional-share fairness for disk I/O resource allocation in VMware ESX hypervisor. Seawall [48], Netshare [44], FairCloud [43], Oktopus [4] and Hadrian [5] all adopt max-min fairness to achieve statistical multiplexing on network resource. [46] is perhaps the closest to our work. It relies on max-min fairness to allocate bulk resource to multiple VMs belonging to the same tenant. However, it is limited to a *single type of resource and a single tenant*. In contrast, RRF advocates resource trading between multiple tenants and addresses the fair-sharing problem on multiple resource types.

VIII. Conclusion

Efficient and fine-grained resource sharing becomes increasingly important and attractive for new-generation cloud environments. This paper proposes *reciprocal resource fairness* (RRF) to address the economic fairness issues in supporting fine-grained resource sharing across multiple resource types in IaaS clouds. RRF embraces two complementary and hierarchical mechanisms: inter-tenant resource trading and intra-tenant weight adjustment. RRF achieves a number of desirable properties for fine-grained resource sharing in cloud environments, including sharing incentive, gain-as-you-contribute fairness and strategy-proofness. We implement RRF on Xen platform. The preliminary results show that RRF is beneficial for both cloud providers and tenants. For cloud providers, RRF improves VM density than current IaaS cloud models by 2.2X and has almost negligible runtime overhead for real-time resource sharing. For tenants, RRF delivers better application performance and 95% economic fairness among multiple tenants.

Acknowledgment

The authors would like to thank the shepherd, Judy Qiu, and the anonymous reviewers for their valuable comments. We acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09.

References

[1] Amazon. *http://aws.amazon.com/solutions/case-studies/*.

[2] Apache YARN. *http://hadoop.apache.org/docs/current2/*.

[3] AutoScaling. *http://aws.amazon.com/autoscaling*.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *SIGCOMM*, 2011.

[5] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea. Chatty tenants and the cloud network sharing problem. In *NSDI*, 2013.

[6] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

[7] 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. In *SOSP*, 2003.

[8] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *TEEE ICAC*, 2005.

[9] A. D. Breslow, A. Tiwari, M. Schulz, L. Carrington, L. Tang, and J. Mars. Enabling fair pricing on hpc systems with node sharing. In *SC*, 2013.

[10] M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency - Practice and Experience*, 11(15):887–911, 1999.

[11] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.

[12] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[13] E. Deelman. Grids and clouds: Making workflow applications work in heterogeneous distributed environments. *IJHPCA*, 24(3):284–298, 2010.

[14] Docker. *https://www.docker.com/*.

[15] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems.

[16] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *NSDI*, 2011.

[17] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Choosy: max-min fair sharing for datacenter jobs with constraints. In *EuroSys*, 2013.

[18] D. Gmach, J. Rolia, and L. Cherkasova. Selling t-shirts and time shares in the cloud. In *CCGRID*, 2012.

[19] Y. Gong, B. He, and D. Li. Finding constant from change: Revisiting network performance aware optimizations on iaas clouds. In *ACM/IEEE SC*, 2014.

[20] Google. *https://cloud.google.com/customers/*.

[21] Grid5000. *https://www.grid5000.fr/mediawiki/index.php/Advanced_OAR*.

[22] A. Gulati, A. Merchant, and P. J. Varman. mclock: handling throughput variability for hypervisor io scheduling. In *OSDI*, 2010.

[23] Hadoop Fair Scheduler. *http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html/*.

[24] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[25] L. Hu, K. D. Ryu, D. Da Silva, and K. Schwan. v-bundle: Flexible group resource offerings in clouds. In *ICDCS*, 2012.

[26] C. Iancu, S. Hofmeyr, F. Blagojevic, and Y. Zheng. Oversubscription on multicore processors. In *IPDPS*, 2010.

[27] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SOSP*, 2009.

[28] N. H. Kapadia and J. A. Fortes. Punch: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, 2(2):153–164, 1999.

[29] S. Keshav. *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[30] C. Lee, P. Wang, and D. Niyato. A real-time group auction system for efficient allocation of cloud internet applications. *IEEE Transactions on Services Computing*, PP(99):1–30, 2013.

[31] J. Lee Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[32] H. Liu and B. He. Reciprocal resource fairness: Towards cooperative multiple-resource fair sharing in iaas clouds. Technical Report 2014-TR-106, Nanyang Technological University, Singapore, http://pdcc.ntu.edu.sg/xtra/tr/2014-TR-106-RRF.pdf, 2014.

[33] H. Liu, H. Jin, X. Liao, W. Deng, B. He, and C.-z. Xu. Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–14, 2014.

[34] LXC. *https://linuxcontainers.org/*.

[35] M. Malawski, K. Figiela, and J. Nabrzyski. Cost minimization for computational applications on hybrid cloud infrastructures. *Future Generation Computer Systems*, 29(7):1786–1794, 2013.

[36] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012.

[37] P. Marshall, K. Keahey, and T. Freeman. Improving utilization of infrastructure clouds. In *CCGrid*, 2011.

[38] M. Maurer, I. Brandic, and R. Sakellariou. Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012.

[39] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *ICAC*, 2010.

[40] Moab/Torque. *http://www.adaptivecomputing.com/products/opensource/torque/*.

[41] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: elastic distributed resource scaling for infrastructure-as-a-service. In *USENIX ICAC*, 2013.

[42] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *EuroSys*, 2009.

[43] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: sharing the network in cloud computing. In *SIGCOMM*, 2012.

[44] S. Radhakrishnan, R. Pan, A. Vahdat, G. Varghese, et al. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *ACM SIGCOMM Computer Communication Review*, 42(3):5–11, 2012.

[45] RUBBOS. *http://jmob.ow2.org/rubbos.html*.

[46] G. Shanmuganathan, A. Gulati, and P. Varman. Defragmenting the cloud using demand-based resource allocation. In *SIGMETRICS*, 2013.

[47] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *SoCC*, 2011.

[48] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *NSDI*, 2011.

[49] D. Shue, M. J. Freedman, and A. Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *OSDI*, 2012.

[50] J. Sonnek and A. Chandra. Virtual putty: Reshaping the physical footprint of virtual machines. In *HotCloud*, 2009.

[51] S. Tang, B.-s. Lee, B. He, and H. Liu. Long-term resource fairness: towards economic fairness on pay-as-you-use computing systems. In *ICS*, 2014.

[52] TPC-C. *http://www.tpc.org/tpcc*.

[53] C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI*, 2002.

[54] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou. Distributed systems meet economics: pricing in the cloud. In *HotCloud*, 2010.

[55] Y. Wang, J. Tan, W. Yu, L. Zhang, X. Meng, and X. Li. Preemptive reducetask scheduling for fair and fast job completion. In *ICAC*, 2013.

[56] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon. Overdriver: handling memory overload in an oversubscribed cloud. In *VEE*, 2011.

[57] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. In *Middleware*, 2008.

[58] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys*, 2010.

[59] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Middleware*, 2007.

[60] A. Zhou and B. He. Transformation-based monetary costoptimizations for workflows in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):85–98, Jan 2014.