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

Haikun Liu
Nanyang Technological University, Singapore
Email:hkliu@ntu.edu.sg

Bingsheng He
Nanyang Technological University, Singapore
Email: bshe@ntu.edu.sg

***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* 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 these 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 including sharing incentive, gain-as-you-contribute fairness and strategy-proofness. To demonstrate the effectiveness and efficiency of RRF in virtualization environments, we implement a system prototype on top of Xen hypervisor. Experimental results show RRF is promising for both cloud providers and tenants. Compared to the current 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.**

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* [19]. 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 may not finely fit the dynamic demands of cloud applications. The result is that tenants need to over-provision resource (costly), or risk suffering performance penalty and Service Level Agreement (SLA) violation (also costly). Second, elastic resource scaling in the cloud [2] is also costly due to the latencies involved in VM instantiating [35] and warming up [40]. These costs are ultimately borne by tenants in terms of monetary cost or performance penalty. These two disadvantages both can be attributed to inefficient resource use of clouds. For cloud providers, inefficient resource utilization translates to higher capital expense and operating expense, and decreases revenue margins.

As more and more applications with diversifying and even complementary resource requirements are already deployed in the cloud [1], [8], [16], [20], [50], [57], ideally, there are vast opportunities for resource sharing among these applications [19], [37]. 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% [10], [27]. The resource utilization can be further improved in virtualization environments [19] by using fine-grained resource sharing techniques and optimizations (e.g., resource multiplexing or overcommitting model [6], [14], [19], [23], [53]). A previous study showed that the fine-grained resource sharing model can reduce the active servers even by 50% compared to T-shirt model [19]. Modeling clouds as a market, many researchers have explored economics-based mechanisms for cost efficiency of tenants and resource efficiency of cloud providers [9], [12], [31], [34], [36], [54].

Despite the resource sharing opportunities, we find that resource sharing, especially for *multiple* resource types, pose 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 concern 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 to gain more benefit. Lying also violates sharing incentive. **(4) Gain-as-you-contribute Fairness**. It is important to guarantee the allocations obey a rule "more contribution, more gain". In summary, all the four problems can be eventually attributed to an economic fairness problem.

Given the fairness problem of resource sharing, one of the most popular allocation policies proposed so far has been (Weighted) Max-Min Fairness (WMMF) [30], 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 [3], [4], [23], [42], [43], [47], [48]. However, their focuses have so far been primarily on a single resource type. Even in multi-resource environments, Dominant Resource Fairness (DRF) [17], cannot address all the above problems in such a cooperative cloud environment (see Section II-B).

To address these problem, 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 across multiple resource types. Particularly, we advocate that different types of resource can be traded among different tenants or 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. Trading can maximize tenants' benefit from resource sharing. RRF decomposes the multiple resource fair sharing problem into two 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 guarantee no tenant can obtain unfair benefit from other tenants when there exists resource contention. Moreover, RRF is able to achieve a number of desirable properties, 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 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 the other fairness mechanisms [17], [30].

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 detials of VMbuddy. 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 representative resource sharing mechanisms, including Weighted Max-Min Fairness (WMMF) [30] and Dominant Resource Fairness (DRF) [17]. 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 [30] 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 [30]: (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 fewer resource demand will be satisfied first. (2) No user obtains a resource share larger than its demand. Thus, the over-allocated portion should be re-allocated to other users with unsatisfied demands. (3) Users with unsatisfied demands get resource shares in proportion to their weights. This policy indicates how to distribute the over-allocated resources to the 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 give 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 satisfied, what they get should be proportion to their weights. The outcome of running this WMMF algorithm is that it maximizes the minimum share of a user whose demand is not fully satisfied.

DRF [17] is the state-of-the-art 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 shares). 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 problem in multi-tenant clouds, 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 fairness so that free-riding and lying problems are avoided.

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. Study on Amazon EC2 pricing data had indicated that the hourly unit cost for 1 GB memory is twice as expensive as one EC2 Compute Unit [56]. 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 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 demand 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.

With the T-shirt Model, we distribute the total capacity

[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 between T-shirt, WMMF, DRF and the proposed RRF resource allocation polices.

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>
RRF Contributions	<0, 0>	<0, 300>	<200, 0>	<200, 300>
RRF shares	<500, 500>	<700, 200>	<800, 1300>	<2000, 2000>
RRF Allocation	<5 GHz, 2.5 GB>	<7 GHz, 1 GB>	<8 GHz, 6.5 GB>	<20 GHz, 10 GB>

among tenants in proportion to their share values of CPU and memory separately. The T-shirt model guarantees that each tenant receives the exact 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 has 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. Although WMMF can guarantee resource efficiency, it cannot fully preserve tenant's asset on resource shares, and eventually results in economic unfairness. 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.

We also apply a weighted version of DRF (**WDRF**) [17] 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 because $max(\frac{8}{20}, \frac{1}{10}) = \frac{8}{20}$. 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. That motivates us to design a new mechanism to reinforce the fairness of multi-resource sharing.

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 a overview of our resource allocation model.

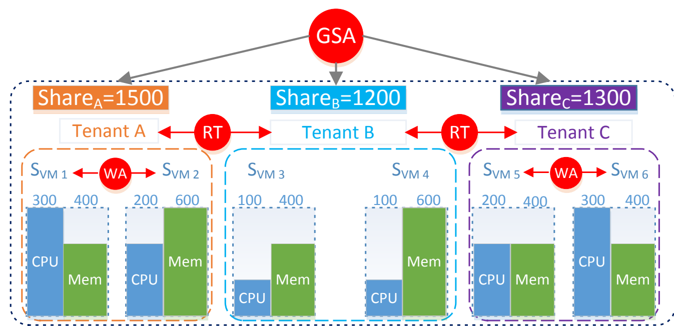

Fig. 1: Hierarchical resource allocation based on resource trading and weight adjustment.

A. Resource Sharing Requirements

DRF [17] 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 these requirements in DRF. Additionally, we highlights the following requirements.

- **A. Sharing Incentive:** Each tenant should benefit from sharing a large resource pool with others, rather than exclusively using fix-sized VMs.
- **B. Gain-as-you-contribute Fairness:** A tenant's gain from other tenants should be proportional to her contribution. The ability of a tenant to gain her unsatisfied resource is determined by the value (or shares) of her other resources that are contributed to other tenants, not by her original total share value or total resource demands.
- **C. Strategy-proofness:** Tenants should not be able to benefit by lying about their resource demands, or deliberately buying less resource than their true demands. 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 multi-resource requests 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 2 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 type of resources (e.g., CPU, Memory) are uniformly normalized based on their market price. To some extent, what a tenant actually buy is resource shares instead of fix-sized resource capacity. Cloud provider can directly use shares as billing and resource allocation polices. The concrete design of those policies is out of the scope of this paper. Thus, we simply define a function f_1 to formulate the relationship from tenants' payment to shares $payment \xrightarrow{f_1} share$, and another function f_2 to translate shares to 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 2GB memory. Thus, VM1 is allocated with total $100 \times 3 + 200 \times 2 = 700$ shares.

Normalizing multiple resources with uniform share provides the advantage to facilitate weight adjustment. It allows tenants dynamically adjust the weight of multi-resource 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 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 multi-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 m tenants and share p types of resource. The total capacity bought by the m tenants is denoted by a vector Ω, such as $\langle \Omega_{CPU}, \Omega_{RAM} \rangle$. Each tenant i may have total 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 current share vector after resource allocation.

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

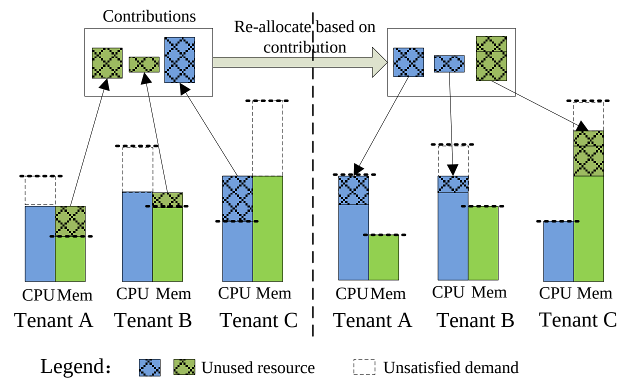

Fig. 2: Sketch of inter-tenant resource trading.

A. *Inter-tenant Resource Trading (IRT)*

For multi-resource allocation, it cannot guarantee that all types of resource demand is nicely satisfied without waste. For example, a tenant's aggregate demand of CPU resource 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 resource 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, Tenant A contributes 2 times memory resource than Tenant B, so she should receive 2 times 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 processing 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 resource re-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. The question is how to divide the tenants into three categories efficiently. Algorithm 1 describes the sorting process by using some heuristics.

Let vectors $D(i), S(i)$ and $C(i)$ denote the total demand, total share and total contribution of 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 share and then caps it at the tenant's 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 of k ($1 \leq k \leq p$), the unused resource $C_k(i)$

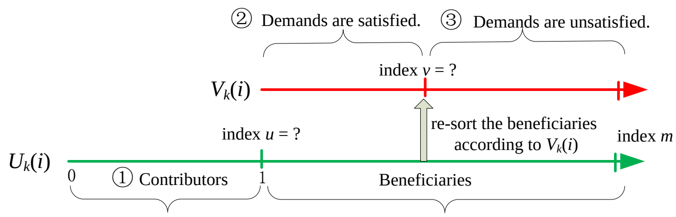

Fig. 3: Sketch of IRT algorithm.

is distributed to other unsatisfied tenants in the ratio of their total contributions.

Algorithm 1 shows the pseudo-code for IRT. We first calculate each tenant's total contribution (lines 6-8). To reduce the complexity of resource allocation, for 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 allocation of tenants with index $[1, \cdots, v]$ are capped at their demands, and the remaining contribution $\Psi(i) = \Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)$ 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 the 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 Equation (1) and (2) must be satisfied:

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

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

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 basic share plus the remaining contributions in proportion to their contributions (lines 19-20).

We illustrate the IRT algorithm using Example 1 in Section 2. Table I shows the details of IRT allocation algorithm

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:     $S'_k(i) \leftarrow S_k(i)$
5:     $U_k(i) \leftarrow D_k(i)/S_k(i)$
6:     **if** $S_k(i) \geq D_k(i)$ **then**
7:       $C_k(i) \leftarrow S_k(i) - D_k(i)$
      /*Calculate tenant(i)'s total contribution $\Lambda(i)$ on all type of resource */
8:       $\Lambda(i) \leftarrow \Lambda(i) + C_k(i)$
9: **for** $resource\_type\ \ k = 1$ **to** p **do**
10:   Sort $U_k(i)$ in ascending order;
11:   Find the index u so that $U_k(u) < 1 \leq U_k(u+1)$
12:   **for** $Tenant\ \ i = u+1$ **to** m **do**
13:     $V_k(i) \leftarrow (D_k(i) - S_k(i))/\Lambda(i)$
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:     $S'_k(i) \leftarrow D_k(i)$ /*share is capped by demand*/
  /*Ψ is the remaining contributions for re-allocation*/
18:   $\Psi_k \leftarrow \Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)$
19:   **for** $Tenant\ \ i = v+1$ **to** m **do**
20:     $S'_k(i) \leftarrow S_k(i) + \frac{\Psi_k \times \Lambda(v+1)}{\sum_{i=v+1}^{m} \Lambda(i)}$

(denoted with RRF). VM1 does not contribute anything, so it just receives its own initial shares. VM2 trades its 1.5 GB RAM for VM3's 2 GHz CPU. Both of their redundant resources can be reciprocally used to meet their unsatisfied demands by resource trading. In comparison with WMMF and DRF, RRF allows both VM2 and VM3 to gain more unsatisfied resource, and prevents free-riding.

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 among tenants, and provide 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 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. 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 share to unsatisfied VMs in ratio of their unsatisfied demands (lines 7 to 11). As VM provisioning is limited to a physical host's capacity, it is desirable to adjust weights on the same physical node, rather than across multiple nodes. In practice, we execute the IWA algorithm only within each 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 shares 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

Theorem 1. *All WMMF-derived fairness 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 losing utilized resource. Thus, all WMMF-derived fairness satisfies 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 share and try best 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 demand 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 II 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 II: 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

We implement a proof-of-concept prototype on Xen to verify the effectiveness of RRF. In the prototype, we have implemented a VM placement algorithm to facilitate resource sharing. Our VM grouping algorithm exploits statistical 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) [32]. The multi-resource provisioning can be formulated as a multi-dimensional bin packing problem [5]. 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 [33].

There are also some other important details for a complete system implementation, such as resource demand prediction, resource allocator, resource pool scaling, load balancing and discussions on IaaS cloud economics. However, due to space limitation, we present these details in Appendix A of our technical report [33].

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 in 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 to 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 in CPU demands, as shown in Figure 4. We evaluate its application performance by throughput (transactions per minute).

RUBBoS (Rice University Bulletin Board System) [44]: 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 within 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.

TABLE III: 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	$<$11.5 core, 10.3 GB$>$	$<$12.5 core, 12.6 GB$>$

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 [33]. 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 III, and configure the initial share of VM (j) based on a provisioning coefficient,

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

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 and real-time resource sharing. We evaluate VMbuddy by comparing the following alternative approaches:

- **T-shirt (static)**: Workloads are running in VMs with static resource provision. It is the current resource model adopted by most IaaS clouds [19].
- **WMMF (Weighted Max-Min Fairness)**: WMMF [30] is used to allocate CPU and Memory resources to each VM separately.
- **DRF (Dominate Resource Fairness)**: DRF [17] 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 demand and allocation for those workloads in 45 minutes. We take a senario where the four workloads are co-located on a single host as an example. Figure 4 and Figure 5 show the normalized total resource demand and total resource allocation (including CPU and Memory) for each workload, respectively. 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

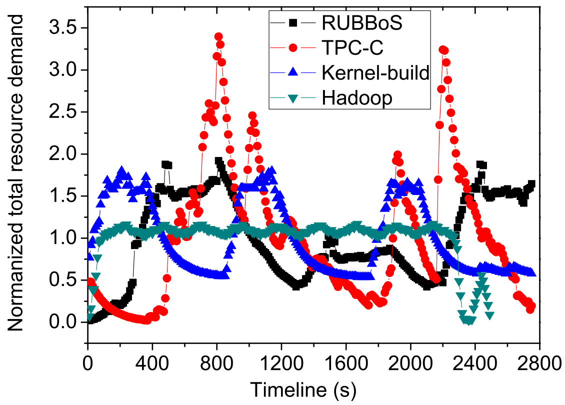

Fig. 4: Normalized total resource demand for each workloads, with respect to its initial share, namely $D_t(i)/S(i)$.

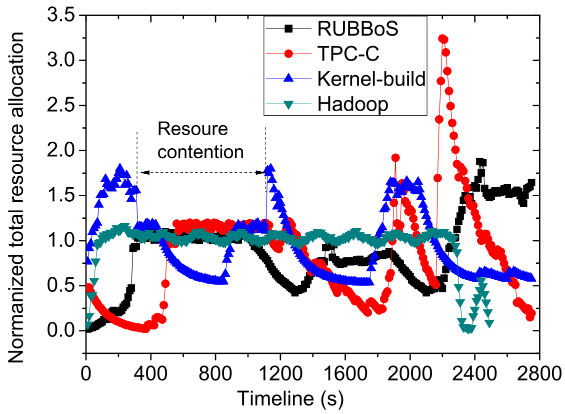

Fig. 5: Normalized total resource allocation for each workloads, with respect to its initial share, namely $S_t^{'}(i)/S(i)$. The detailed allocations are based on RRF algorithm.

of 320s - 1100s, and thus there is resouce contention between these workloads. In other periods, the server can successfully satisfy the resource demand of each workload.

Figure 5 shows the allocation detials for each workloads over time. In the peroid 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 workloads are allocated with its demand.

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 buy. We call it *fair* if a tenant can achieve this goal (i.e., sharing benefit). In contrast, it is also possible for 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:

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

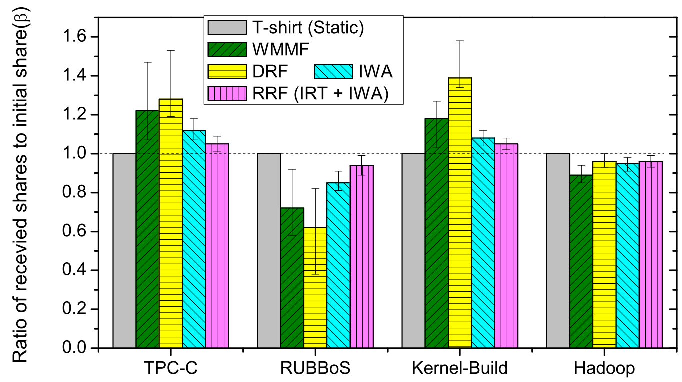

Fig. 6: Comparison of economic fairness of several resource allocation schemes.

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 for resource sharing. In contrast, $\beta(i) < 1$ implies the tenant loses her asset.

Figure 5 has illustrated a kind of fairness on resource allocation at instant 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

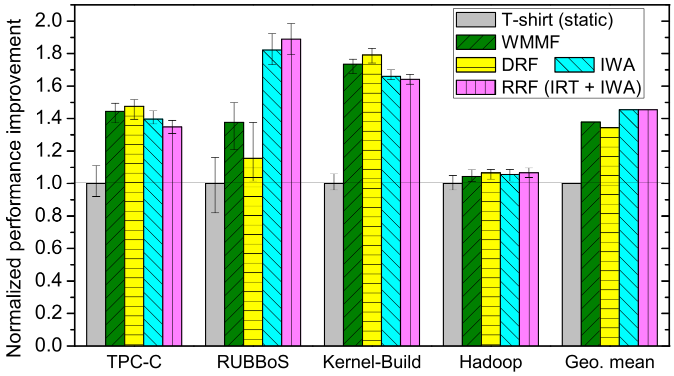

Fig. 7: Comparison of application performance improvement of several resource allocation schemes.

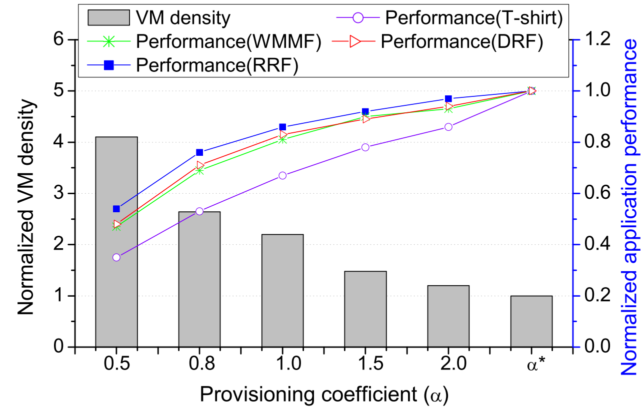

Fig. 8: Application performance corresponds to VM density.

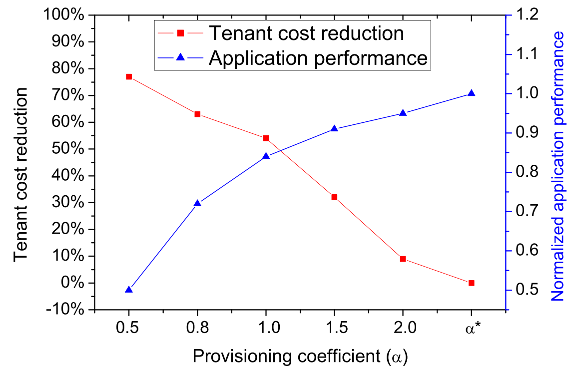

Fig. 9: The tradeoff between tenant cost reduction and application performance, compared to T-shirt model.

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 VM (except in its Reduce stage).

Fourth, inter-tenant weight adjustment (IWA) allows tenants properly distribute some VMs' spare capacity to the 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 over-provisioning and 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 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 $\alpha = \alpha^*$, RRF 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. We have analyzed the cloud provider revenue with both back-of-the-envelope calculations and simulations. We find that RRF can significantly increase the cloud provider revenue when $\frac{U_{RRF}}{U_{T-shirt}} > \frac{\alpha}{\alpha^*}$, where U_{RRF} and $U_{T-shirt}$ denote the cloud resource utilization under RRF

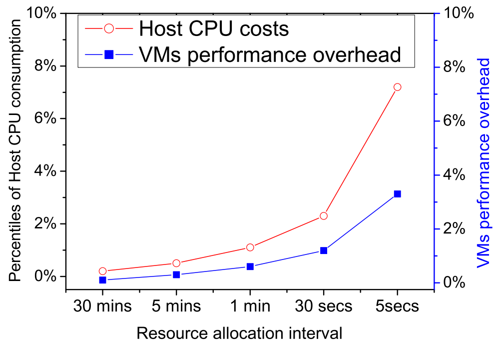

Fig. 10: Performance overhead of RRF varies with the frequency of resource dynamic allocations.

and T-shirt, respectively. The simulation results show that RRF can significantly increase the cloud provider's revenue even by a factor of 2.33 when the resource utilization is larger than 47%. Due to space limitation, we present more details in Appendix B of our technical report [33].

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.

Resource sharing and provisioning. Despite the vast amount of work on fair resource allocation in Clusters or Girds, including Grid5000-OAR [21], Nimrod-G [13], Moab/Torque [39], CCS [11] and Punch [29], 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, page-sharing [53], [6], and CPU & I/O scheduling [14], [23]. There have been a number of works focusing on VM resource multiplexing strategies to improve resource utilization. Virtual-putty [49] exploited affinities and conflicts between co-located VMs to improve the host resource utilization. Meng *et al.*[38] proposed a joint-VM provisioning strategy to consolidate multiple VMs based on the estimate of their total resource requirement. v-Bundle [26] offers elastic exchange of network resource among multiple VMs belonging to the same tenant. These studies did not consider the fairness issues in resource sharing. Furthermore, several other trace-based approaches [7], [41], [58], [60] are developed for demand prediction and capacity provisioning. These approaches are complementary to RRF, and help tenants to estimate the total amount of purchase.

Most IaaS clouds sell compute resource in a T-shirt model [19] and charge the resource on a pay-as-you-go basis. In the T-shirt model, the challenging problem is resource scaling in an on-demand manner [2]. CloudScale [46] 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. RRF is complementary to these techniques of resource auto-scaling 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 [30]. Many data center or cluster schedulers supports max-min fairness or its extensions, such as Hadoop's Fair Scheduler [24], Quincy [28], Mesos [25], and Choosy [18]. Quincy [28] achieves fairness of scheduling concurrent distributed jobs by formulating the problem as a min-cost flow problem. Mesos [25] achieves fair resource sharing by using DRF [17]. Choosy [18] is job scheduler that provides Constrained Max-Min Fairness, an extension of max-min fairness that supports hard job placement constraints. Delay scheduling [59] and fair completion scheduler [55] address fairness of resource allocation for MapReduce jobs. 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 VM level.

Many studies have considered the fairness of sharing single-type resource, such as I/O or network bandwidth. Pisces [48] is a systemic implementation of max-min fairness in multi-tenant shared key-value storage system. mclock [23] supports proportional-share fairness for disk I/O resource allocation in VMware ESX hypervisor. Seawall [47], Netshare [43], FairCloud [42], Oktopus [3] and Hadrian [4] all adopt max-min fairness to achieve statistical multiplexing on network resource. [45] 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. This work is limited to a *single type of resource and a single tenant*. In contrast, RRF advocates resource trading between multiple tenants and address the fair-sharing problem on multiple resource types.

VIII. Conclusion

As more and more workloads are deployed in the cloud, efficient and fine-grained resource sharing becomes increasingly important and attractive for new-generation cloud environments. This paper propose *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 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 multi-tenants.

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] G. B. Berriman, E. Deelman, G. Juve, M. Rynge, and J.-S. Vöckler. The application of cloud computing to scientific workflows: a study of cost and performance. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983):20120066, 2013.

[9] I. Breskovic, I. Brandic, and J. Altmann. Maximizing liquidity in cloud markets through standardization of computational resources. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 72–83. IEEE, 2013.

[10] 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.

[11] 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.

[12] 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.

[13] R. Buyya, J. Giddy, and D. Abramson. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. In *Active Middleware Services*, pages 221–230. Springer, 2000.

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

[15] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

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

[17] 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.

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

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

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

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

[22] Groupon. *http://www.groupon.com/*.

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

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

[25] 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.

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

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

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

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

[30] 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.

[31] 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*, 2013.

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

[33] H. Liu and B. He. Reciprocal resource fairness: Towards co-operative multiple-resource fair sharing in iaas clouds. Technical Report 2014-TR-106, Nanyang Technological University, Singapore, http://www3.ntu.edu.sg/home/bshe/2014-TR-106-ddf.pdf, 2014.

[34] 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.

[35] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

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

[37] M. Maurer, I. Brandic, and R. Sakellariou. Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 368–375. IEEE, 2012.

[38] 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.

[39] Moab/Torque. *http://www.adaptivecomputing.com/products/open-source/torque/*.

[40] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: elastic distributed resource scaling for infrastructure-as-a-service. In *Proc. of the USENIX International Conference on Automated Computing (ICAC13). San Jose, CA*, 2013.

[41] 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.

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

[43] 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.

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

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

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

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

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

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

[50] S. Strauch, V. Andrikopoulos, D. Karastoyanova, and K. Vukojevic. *Migrating eScience Applications to the Cloud: Methodology and Evaluation*. Cloud Computing with E-science Applications. CRC Press/Taylor & Francis, Dezember 2014.

[51] Teambuy. *http://www.teambuy.ca/*.

[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] Windows Azure. *http://www.windowsazure.com/en-us/overview/case-studies/*.

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

[59] 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.

[60] 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.

APPENDIX

APPENDIX A

MORE IMPLEMENTATION DETAILS

A. VM Grouping

Inspired by the resource time-sharing model, we co-locates tenants with complementary resource requirements to facilitate resource multiplexing. Resource sharing can motivate the cloud tenants to provision less resource than their peak demands and thus reduce the monetary cost. On the other hand, the cloud providers also benefit from the resource multiplexing because significant improvement of VM density can increase their revenues, and the improved resource efficiency can gain a competitive advantage against other cloud providers.

Our VM grouping algorithm exploits statistical resource multiplexing among different VMs to improve the sharing opportunity. Ideally, on the temporal dimension, the peaks and valleys of different VMs' resource demands should not be coincide with each other. On the spatial dimension, VMs with different dominate resource requirements can be grouped based on the skewness of multiple resources. For example, VMs with CPU-intensive workloads can be co-located with I/O-intensive workloads so that all resource types are efficiently utilized.

As VM multiplexing on the temporal dimension needs historical statistics of workload pattern, such placement optimizations have to be made at runtime. Here, we only optimize multi-resource provisioning on the spatial dimension. A desirable solution not only needs to facilitate the resource provisioning, but also maximizes the opportunities for resource sharing. Also, we should carefully consider the resource characteristics of physical machines. A physical machine can be characterized by multiple resource types $k(1 \leq k \leq p)$ and other parameters such as memory volume per CPU $\Omega_{RAM}/\Omega_{CPU}$. If the aggregate demand of co-located VMs is compatible with hardware characteristics of physical machines, we could achieve higher resource utilization and reduce the operational cost of clouds.

We introduce the concept of *correlation* to quantify the similarity of a VM and a server's capacity characteristic. Without loss of generality, we characterize the correlation of two vector X, Y by Pearson's Correlation Coefficient (PCC) [32]:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \ \sigma_Y},$$

where $cov(X,Y)$ is the covariance of the two vectors, and σ_X and σ_Y are their standard deviations. PCC is widely used to measure the degree of linear dependence between two variables. The value of PCC ranges from -1 to +1, where 1 implies completely positive correlation, 0 implies no correlation, and -1 implies completely negative correlation. Note that PCC does not reflect the relationship on quantity. For example, PCC deems vectors $\langle 10, 15\rangle$ and $\langle 30, 45\rangle$ have completely positive correlation ($\rho = 1$).

The multi-resource provisioning can be formulated as a multi-dimensional bin packing problem [5], which it is a combinatorial NP-hard problem. We proposed the following heuristics to approximate its optimal solution.

1) To improve server utilization, VMs should be placed on the servers in the decreasing order of their capacity demands. This is consistent with First Fit Decreasing (FFD) strategy [15].
2) Once we place a VM on a server, we try to minimize the skewness of the server's remaining multiple resources. It implies that we should always place VMs with inverse skewness (or inverse dominate resource demands) together.

Guided by these heuristics, we propose a VM provisioning algorithm based on iteratively eliminating the skewness of server's remaining multiple resources. We first calculate total demands of VM j by accumulating the total demand shares of p types of resource, and then place VMs in a decreasing order of the total demand shares. Once we place a VM with index j on a server, we try to find a VM $(j + a)$ (a is an integer) that has the highest PPC with VM j, so that their aggregate resource correlation relative to the physical server' capacity is maximized. Namely, we need find $max\{\rho\big(d(j)+d(j+a), \Omega\big)\}$. Once the VM is found and placed to the server, we continue placing VM $(j + 1)$ to a server and searching its siblings. By repeating this processing, the VMs should be placed to the servers compactly.

B. Other Implementation Issues

Here, we address some issues that need to be handled in a complete implementation of VMbuddy.

Resource demand prediction: So far we have assumed VM demands as one of the inputs in our algorithms. The initial share is a static parameter set by the tenants upon VM creation, while the demand is estimated periodically at run-time. The demand prediction algorithm is orthogonal to our allocation model. Despite a lot of works on demand predictions [7], [41], [58], [60], we choose a simple yet efficient one. We adjust the demand window according to the FAST-TCP update, with Exponential Weighted Moving Average (EWMA): $E(t) = \lambda * O(t) + (1-\lambda) * E(t-1), 0 \leq \lambda \leq 1$, where $E(t)$ and $O(t)$ are the estimated and observed demands at time t, respectively.

Resource allocator: A key reason to use VMs in cloud computing is that VMs are abstracted as well-encapsulated resource container. Xen and other hypervisors have provided several techniques for fairly allocating CPU time and memory capacity to VMs [6], [14], [53].

For CPU resource allocation, The open-source Xen provides several schedulers for fair sharing and allocation of CPU resource in different scenarioes [14]. We use the credit scheduler [14] in our prototype system as it is most commonly used in deployments. Xen hypervisor provide 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 (shares). Moreover, the credit scheduler supports a non-workconserving mode, in which VMs CPU time can be capped at their demands. These features are perfectly fit for our resource allocation requirements.

For memory allocation, Xen hypervisor also provide a interface to dynamically adjust each VM's memory capacity through ballooning or hotpluging [6], [53]. We only need to determine the memory share of each VM before the memory allocation, which is done by the RRF allocation algorithms.

Resource pool scaling: Current clouds charge the resource on a pay-as-you-go basis, our resource allocation model cloud be consistent with this paradigm. Tenants can still acquire or release resources by increasing/decreasing the number of VMs based on load dynamics. In addition, our model allows tenants to increase/decrease a single VM's fine-grained resources (i.e., CPU and DRAM) via dynamic payment. If the tenant's total share consistently exceeds her aggregate demand, it indicates that the VMs have likely been over-provisioned. The tenant may cautiously reduce her resource provision to save money. In contrast, persistent unsatisfied demands imply resource under-provisioning. The tenant must increase her shares by buying more resource from the cloud provider. For temporary and reasonably small load spikes, the tenant has opportunities to satisfy such demands by trading resources with other tenants. Adding/releasing resource to/from the resource pool only need to update the information of resource configuration at the node manager.

Load balancing: Although intra-tenant weight adjustment has provided a kind of load balancing mechanism for co-located VMs belonging to the same tenant, it cannot handle load imbalance between different physical nodes. Our prototype also supports VMs migration for load management. However, VM migration may affect the target server's initial ecosystem. For example, suppose VM1 on an overloaded server *A* is migrated to an unloaded server *B*. If the server *B* becomes overloaded after a period of time, VM1 should be moved out of server *B* because it would dilute other VMs' resource share. During this period, although the aggregate share of all VMs has exceeded the total capacity of server *B*, the VM1 can still stay on server *B* till the VM's total demand becomes larger than the server's capacity. VM live migration can also facilitate resource trading between tenants. For example, suppose server *A* lacks CPU resource but has spare memory resource, while server *B* has spare CPU resource but demands more memory. We should find a VM from each server that can minimize the skewness of its resource demand, and then swap the two VM's location with live migration.

IaaS cloud economics. We currently keep the pricing scheme of the cloud provider unchanged. We believe that a detailed discussion of IaaS cloud economics (for example, different pricing schemes and marketing mechanisms) is beyond the scope of this paper. Instead, we briefly present a few interesting and potential pricing schemes for the resource sharing model. First, we have witnessed the prosperousness of a group-buying mechanism in real product and service markets, which is essentially a kind of marketing strategy with benefits for both buyers and sellers [22], [51]. In the real world, group-buying offers products or services at significantly discounted prices on the condition that the item is bought in a minimum quantity or dollar amount. This market strategy not only motivates the cloud users to buy resource at a lower price, but also motivates the cloud users to provision resource based on average usage while sharing a large resource pool with other tenants. Second, cloud provider may offer some discounts for the user initiatives of sharing resources. In fact, cloud vendors such as Amazon EC2 already provide discounted pricing when tenants buy in bulk (e.g., reserved instances).

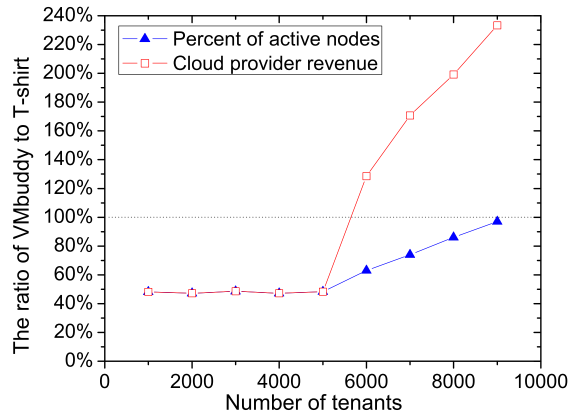

Fig. 11: Relative cloud provider revenue and resource utilization, under RRF versus under T-shirt.

APPENDIX B
CLOUD PROVIDER REVENUE

We analyze the cloud provider revenue with both back-of-the-envelope calculations and simulations.

First, we analyze the impact of resource multiplexing on the cloud provider's revenue. Assume that a cloud data center has total capacity of Ω. The tenants' total demand is $\sum_{i=1}^{m} D(i)$. Let P denote the price of each unit of resource, and $U_{T-shirt}$ and U_{RRF} denote the resource utilizations under T-shirt model and under RRF, respectively. α^* and α denote the provisioning coefficient of T-shirt model and RRF, respectively. For the T-shirt model, the cloud revenue $R_{T-shirt}$ is the tenants' payment, which is presented by $R_{T-shirt} = \sum_{i=1}^{m} D(i) \times P$. Correspondingly, the cloud provider revenue with RRF is $\frac{\alpha}{\alpha^*} R_{T-shirt}$. In this case, RRF leads to lower cloud provider revenue. The good side is that the resource utilization is $\frac{\alpha}{\alpha^*} U_{T-shirt}$, which means that RRF can activate a smaller number of servers and devices (e.g., for a smaller electricity bill). Moreover, RRF can help cloud provider to gain the competitive advantage in the cloud market: since RRF reduces the tenant cost, tenants are keen to select that cloud provider. However, even if the cloud is fully occupied by the T-shirt model, VMBuddy can still serve more tenants. In that case, the cloud revenue R_{RRF} becomes $R_{RRF} = \frac{\alpha^*}{\alpha} \times U_{RRF} \times \Omega \times P$, where $\frac{\alpha^*}{\alpha}$ represents the VM density improved by RRF. Finally, we can calculate the ratio of cloud revenue, under RRF versus under T-shirt:

$$\frac{R_{RRF}}{R_{T-shirt}} = \frac{\frac{\alpha^*}{\alpha} \times U_{RRF} \times \Omega \times P}{\sum_{i=1}^{m} D(i) \times P} = \frac{\alpha^*}{\alpha} \times \frac{U_{RRF}}{U_{T-shirt}}$$

From the above formula, we find that RRF can increase the cloud provider revenue only when $\frac{U_{RRF}}{U_{T-shirt}} > \frac{\alpha}{\alpha^*}$. This implies that cloud providers should attract more tenants to increase their resource utilization. Also, a lower provisioning coefficient leads to less tenants' cost, and thus is more attractive to tenants.

TABLE IV: A 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>

Some pricing schemes can be used to address some of these issues, as we discussed in our technical report [33].

Second, we use simulations to verify the analysis at a large scale. We simulate a data center with 1000 physical nodes to serve multiple tenants. Each node is equipped with 24 cores and 24 GB RAM. We also simulate tenants to request resource from the data center. The resource requirement of each workload is randomly generated between 2 to 5 cores and between 2 to 5 GB RAM, and follows the Poisson distribution. For T-shirt model and RRF, the VMs are provisioned according to the peak demands and average demands, respectively. We continuously increase the number of tenants until no room can serve more tenants. Figure 11 shows the ratio of cloud provider revenue and the ratio of active nodes, under RRF versus under T-shirt. T-shirt model can only meet at most 4500 tenants' requests when the percent of active nodes approach 100%. In contrast, RRF only needs to activate 47% physical node to serve these tenants. RRF delivers the same revenue as T-shirt model, excluding the power saving of 53% inactive nodes. When the percent of active nodes is more than 47%, RRF can significantly increase the cloud provider's revenue even by a factor of 2.33 because it can serve up to 9300 tenants. The simulation result is compatible with our back-of-the-envelope analysis.

APPENDIX C
A EXAMPLE FOR IRT ALGORITHM

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 priced at 100 and 200 shares, respectively. The VMs' resource demand, initial shares and demanded shares are illustrated in line 2-4 of Table IV. We first calculate each VM's contribution, as shown in line 5 of Table IV. The following lines show the details of CPU and memory allocation based on IRT algorithm. For CPU resource, VM3 and VM4 are the contributors while VM1 and VM2 are beneficiaries. However, as VM1 contributed nothing to others, VM2 receives all unused CPU shares (200+100) from VM3 and VM4. For memory resource, 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 300 unused memory shares,respectively. VM1 again receives nothing as it does not contribute anything to others.

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