F2C: Enabling Fair and Fine-grained Resource Sharing in Multi-tenant IaaS Clouds

Haikun Liu, *Member, IEEE,* and Bingsheng He, *Member, IEEE*

Abstract—This paper presents *F2C*, a cooperative resource management system for Infrastructure-as-a-Service (IaaS) clouds. Inspired by group-buying mechanisms in real product and service markets, F2C advocates a group of cloud tenants (called *tenant coalition*) to buy resource capacity in bulk and share the resource pool in the form of virtual machines(VMs). Tenant coalitions leads to vast opportunities for fine-grained resource sharing among multiple tenants. However, resource sharing, especially for *multiple* resource types, poses several 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 on multiple resource types within a tenant coalition. RRF is implemented in two complementary and hierarchical mechanisms: inter-tenant resource trading and intra-tenant weight adjustment. RRF satisfies several highly desirable properties to ensure fairness. We implement F2C in Xen platform. The experimental results show F2C is promising for both cloud providers and tenants. For cloud providers, F2C improves VM density and cloud providers' revenue by 2.2X compared to the current IaaS cloud models. For tenants, F2C improves application performance by 45 percent and guarantees 95 percent economic fairness among multiple tenants.

Index Terms—Cloud computing, Fairness, IaaS, Resource sharing, Virtual machine.

1 Introduction

The Infrastructure-as-a-Service(IaaS) cloud has emerged as an appealing paradigm for elastic computing over the Internet. IaaS clouds 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, main memory and disk. Tenants can only purchase fixed-size VMs and increase/decrease the number of VMs when the resource demands change. This is known as *T-shirt and scale-out model* [14]. However, the T-shirt model leads to inefficient allocation of cloud resource, which translates to higher capital expense and operating cost for cloud providers, and increase of monetary cost for tenants. First, the granularity of resource acquisition/release is coarse in the sense that the fix-sized VMs are not tailored for cloud applications with dynamic demands delicately. As a result, tenants need to over-provision resource (costly), or risk performance penalty and Service Level Agreement (SLA) violation. Second, elastic resource scaling in clouds [3], also known as scale-out model, is also costly due to the latencies involved in VM instantiating [28] and software runtime overhead [31]. These costs are ultimately borne by tenants in terms of monetary cost or performance penalty.

- *Haikun Liu is with Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China. E-mail: haikunliu@gmail.com.*
- *Bingsheng He is with Nanyang Technological University, Singapore. E-mail: he.bingsheng@gmail.com*

Manuscript received November 12, 2014

Resource sharing is a classic and effective approach to resource efficiency. As applications with diversifying and heterogeneous resource requirements are already deployed in the cloud [2], [11], [15], there are vast opportunities for resource sharing [12], [14], [29]. Recent work has shown that fine-grained and dynamic resource allocation techniques (e.g., resource multiplexing or overcommitting [7], [9], [14], [17], [43]) can significantly improve the resource utilization compared to T-shirt model [14]. As adding/removing resource is directly performed on the existing VMs, fine-grained resource allocation is also known as scale-up model and the cost tends to much smaller compared to the scale-out model. Unfortunately, current IaaS clouds do not offer resource sharing among VMs, even if those VMs belong to the same tenant. We seek whether there is a better resource model than the T-shirt model to enable resource sharing for better cost efficiency of tenants and higher resource utilization of cloud providers.

Recently, we have witnessed the prosperousness of a group-buying mechanism in real product and service markets, such as Groupon [16] and Teambuy [40]. Group-buying offers products or services at discounted prices when the item is bought in a minimum quantity or dollar amount. This market-based mechanism can benefit both cloud tenants and providers in IaaS clouds. The group-buying mechanism binds different VMs to form a cooperative group, which even goes beyond a single tenant. In this paper, we call it *tenant coalition*. A tenant coalition can buy resource in bulk and cooperatively share it as a large resource pool. This mechanism leads to opportunities for fine-grained resource sharing among multiple tenants.

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, 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. Unfortunately, the popular allocation policies such as (Weighted) Max-Min Fairness (WMMF) [4], [5], [21] and Dominant Resource Fairness (DRF) [12] cannot address all the four problems of resource sharing in IaaS clouds (see Section 3.1).

This paper proposes **F2C**, a cooperative resource management system for IaaS clouds. F2C exploits statistical resource complementarity to realize resource sharing opportunities among tenants, and adopts a novel resource allocation policy to guarantee fairness of resource sharing. Particularly, we develop *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. 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-provisioned 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 decomposes the multi-resource fair sharing problem into a combination of two complementary mechanisms: Inter-tenant Resource Trading (IRT) and Intra-tenant Weight Adjustment (IWA). These mechanisms guarantee that tenants only allocate minimum shares to their non-dominant demands and maximize the share allocations on the contended resource. 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 3.2).

F2C is implemented on top of Xen hypervisor. We conduct a set of experiments to evaluate the allocation fairness, resource efficiency and application performance. The experimental results show that F2C is beneficial for both cloud providers and tenants. For cloud providers, F2C improves VM density and cloud providers' revenue by 2.2X compared to the current T-shirt model. For tenants, F2C delivers better application performance and economic fairness than other state-of-the-art fairness mechanisms [12], [21].

The remainder of the paper is organized as follows. Section 2 introduces the system overview and resource allocation model. Section 3 studies the fairness problems of multi-resource sharing in multi-tenant clouds. Section 4 presents RRF-based resource dynamic allocation algorithms. Section 5 describes the implementation details of F2C. We present the experimental results in Section 6. We study the related work in Section 7 and conclude in Section 8.

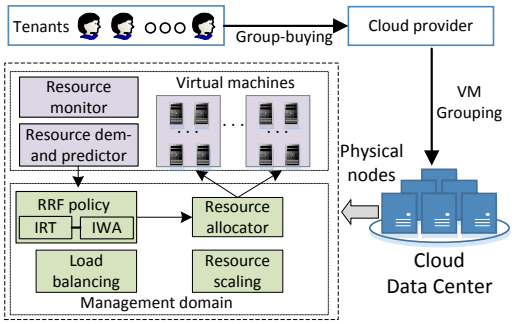

Fig. 1: System architecture of F2C.

2 System Overview

2.1 System Architecture

Figure 1 shows the system architecture of F2C. A number of tenants can request VMs from cloud providers through a group-buying mechanism. F2C exploits statistical resource complementarity to form a tenant coalition, and these VMs are co-located together to share a large resource pool. On each physical node, F2C adopts RRF policy to guarantee fairness of resource sharing. In each VM, we deploy a resource demand predictor to predict the VM's fine-grained resource requirement based on a resource monitor. The resource demands of VMs are taken as inputs of RRF algorithm, which is deployed in each management domain (i.e. domain 0). RRF (including IRT and IWA) is periodically executed to determine resource allocation of each VM. The resource allocator changes the parameters of Xen credit scheduler and ballooning driver to adjust resource allocation of each VM. In domain 0, we also implement two modules for load balancing and resource scaling. More details can be referred to Subsection 5.

2.2 Resource Allocation Model

We consider the resource sharing model in multi-tenant cloud environments, where each tenant may rent several VMs to host her applications, and each VM has multi-resource demands. By *multi-resource*, we mean resource of *different types*, instead of multiple units of the same resource type. In this paper, we mainly consider two resource types: CPU and main memory. Tenants can have different *weights* (or *shares*) to the resource. The *share* of a tenant reflects the tenant's priority relative to other tenants. A number of tenants can form a resource pool based on resource complementarity. The VMs of these tenants then share the same resource pool with negotiated resource *shares*, which are determined by tenants' payment.

In our resource allocation model, each unit of resource is represented by a number of shares. To simplify multi-resource allocations, 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 [44] had indicated that the hourly unit cost for one 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.

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. The shares of different resources (e.g., CPU, Memory) are uniformly normalized based on their market prices [44]. To some extent, a tenant actually purchases resource shares instead of fix-sized resource capacity. Cloud providers can directly use shares as billing and resource allocation policies. Here, 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 2, one compute unit and one GB memory are priced at 100 and 200 shares, respectively. If VM1 is initialized with 3 compute units and 2 GB memory, VM1 is allocated with total $100 \times 3 + 200 \times 2 = 700$ shares.

This model enables fine-grained resource allocation and dynamic resource sharing based on shares. Now, the fairness has become a major concern in such a shared system. Informally, we define a kind of *economic fairness*: each tenant should try to maximize her aggregate multi-resource shares if she has unsatisfied resource demands.

We find that resource trading between different tenants is able to reinforce the economic fairness of resource sharing. Normalizing multiple resources with uniform shares is able to facilitate resource trading. For example, one tenant can trade her over-provisioned CPU shares for other tenants' underutilized memory shares. In Figure 2, VM1 may trade its 200 CPU shares for VM3's 100 memory shares. Resource trading can prevent tenants from losing underutilized resource. Moreover, a tenant can dynamically adjust share allocation of her VMs according to actual demands. For example, tenant *A* may deprive 200 memory shares from VM2 and re-allocate them to VM1. In this paper, we propose resource trading between different tenants (Section 4.1), and dynamic weight adjustment among multiple VMs belonging to the same tenant (Section 4.2). Figure 2 shows the hierarchical resource allocation based on the two mechanisms. The global share allocator (GSA) first reserves capacity in bulk based on 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.

3 Multi-resource Fair Sharing Issues

We start with analyzing the problems of two popular resource allocation policies including Weighted Max-Min Fairness (WMMF) [21] and Dominant Resource Fairness (DRF) [12]. That motivates us to define a number of desirable requirements for multi-resource allocation.

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

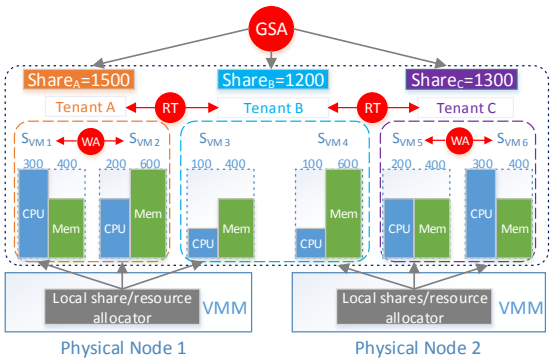

Fig. 2: Hierarchical resource allocation based on resource trading and weight adjustment.

3.1 Motivations

WMMF [21] is widely used to solve the problem of allocating scarce resource among a set of users. WMMF algorithm defines three principles to allocate resources [21]: (1) Resources are allocated in ascending order of 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-provisioned portion should be re-allocated to other unsatisfied users. (3) Users with unsatisfied demands get resource shares in proportion to their weights. This policy defines how to distribute the over-provisioned resources to unsatisfied users. The outcome of running WMMF is that it maximizes the minimum share of a user whose demand is not completely satisfied.

DRF [12] is a generalization of max-min fairness to multiple resource types. The core idea of DRF is that the 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.

In the following, we demonstrate the deficiency of WMMF and DRF for multi-resource sharing in clouds.

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 1. We examine whether T-shirt model, WMMF and DRF can achieve resource efficiency and economic fairness.

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

TABLE 1: 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>

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 1, 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**) [12] 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.

3.2 Resource Sharing Requirements

In the following, we describe a series of requirements that we believe any multi-resource fair sharing model for multiple tenants should satisfy. We consider all requirements/properties defined in DRF [12]. Particularly, we highlight the following requirements.

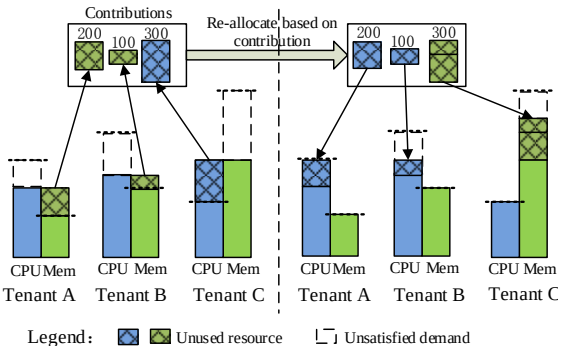

Fig. 3: Sketch of inter-tenant resource trading.

- **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, i.e., the gain is determined by the value (or shares) of her underutilized 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 requirement **A** and requirement **B**, because no tenants can get more resource than their shares by lying or free-riding.

4 Reciprocal Resource Fairness

F2C uses RRF, a new approach to multi-resource allocation that meets all the required properties described in Subsection 3.2. This section presents the detailed design of RRF.

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., $\langle \Omega_{CPU}, \Omega_{RAM} \rangle$, denoting the amount of CPU and memory resource, respectively. 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 $d(j)$. Correspondingly, the resource share lent to other tenants becomes $s(j) - d(j)$, and we call it a contribution vector $c(j)$. At last, let $s^{'}(j)$ denote the current share vector when resources are re-allocated. For simplicity, a VM's priority is always determined by its initial share vector $s(j)$ in each time of resource allocation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2499769, IEEE Transactions on Parallel and Distributed Systems

4.1 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 underutilized resource allocation is the tenant's contribution, rather than her initial resource share or unsatisfied demand. As shown in Figure 3, the memory resource contributed by Tenant A is twice more than that of Tenant B, and Tenant A should receive twice 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-provisioned. If so, the unused portion should be re-distributed to other tenants. This process should be *iteratively* performed by all tenants because each time of resource distribution may affect other tenants' allocations. While this naive approach works, it can cause unacceptable computation overhead. We further 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 4. 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 and resource contention ($\sum_{i=1}^{m} D(i) \geq \Omega$). 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 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 4. 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 4. 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 $\sum_{i=v+1}^{m} \Psi_k(i) = \Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)$ 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 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:

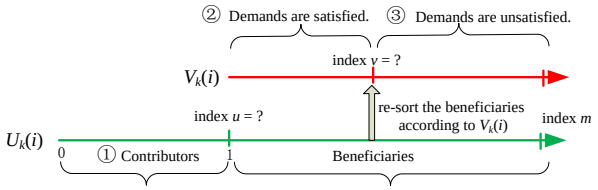

Fig. 4: Sketch of IRT algorithm.

$$S_k(v) + \frac{\sum_{i=v}^{m} \Psi_k(i) \times \Lambda(v)}{\sum_{i=v}^{m} \Lambda(i)} \geq D_k(v) \quad (1)$$

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

where $\sum_{i=v}^{m} \Psi_k(i)$ and $\sum_{i=v+1}^{m} \Psi_k(i)$ 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 the following example. Suppose 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 Lines 2–4 of Table 2. We first calculate each VM's contribution, as shown in Line 5 of Table 2. 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

TABLE 2: 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 ($1000+\frac{300\times100}{100+200}$) , VM3 ($1000+\frac{300\times200}{100+200}$) , 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>

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*/
/*Calculate the remaining contributions for re-allocation*/
18: $\sum_{i=v+1}^{m} \Psi_k(i) \leftarrow \Omega_k - \sum_{i=1}^{v} D_k(i) - \sum_{i=v+1}^{m} S_k(i)$
19: **for** $Tenant$ $i=v+1$ **to** m **do**
20: $S^{'}_k(i) \leftarrow S_k(i) + \frac{\sum_{i=v+1}^{m} \Psi_k(i) \times \Lambda(v+1)}{\sum_{i=v+1}^{m} \Lambda(i)}$

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 300 unused memory shares (i.e., 200 and 100 shares), respectively. VM1 again receives nothing as it does not contribute anything to others.

In summary, with IRT, unused resources are properly re-distributed based on VMs' contributions and free riders can not benefit from others. IRT always tries to minimize resource losing and thus guarantee economic fairness.

4.2 Intra-tenant Weight Adjustment (IWA)

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. In F2C, we use IWA to adjust the resource 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.

4.3 Analysis of Fairness Properties

This subsection makes comparison between RRF, WMMF and DRF on the properties presented in Section 3.2.

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

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** $VM \quad j = 1$ **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** $VM \quad j = 1$ **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: $s^{'}(j) \leftarrow d(j)$

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 satisfies.*

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 her contribution to 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 does not 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.*

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 unused resource of a tenant based on other tenants' initial shares, 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 3 summarizes the fairness properties 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.

TABLE 3: Properties of WMMF, DRF, and RRF.

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

5 System Implementation

We implement F2C on top of Xen 4. The following describes implementation details of key components in F2C.

5.1 VM Grouping

Our VM grouping algorithm co-locates tenants with complementary resource requirements to increase sharing opportunities. Cloud tenants submit their resource requirements, VM configurations, workload patterns and other preferences or constraints. The VM grouping algorithm then place tenants in a suitable coalition to enhance the statistical resource multiplexing among different VMs. VMs with different dominant resource requirements can be grouped based on the skewness of multi-resource requirements. For example, VMs with CPU-intensive workloads can be co-located with memory-intensive workloads so that all resources are efficiently utilized.

A desirable solution not only needs to facilitate the resource provisioning, but also maximizes the opportunities of 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) [22]: $\rho(X, Y) = \frac{cov(X,Y)}{\sigma_X \ \sigma_Y}$, 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.

The multi-resource provisioning can be formulated as a multi-dimensional bin packing problem [6], which it is a combinatorial NP-hard problem. We propose the following heuristics that are specially designed for our resource sharing and trading mechanisms.

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 [10].
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 PCC (or inverse dominant resource demands) together.

Guided by those 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 total demand shares of p types of resource, and then place VMs in a decreasing order of 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 most inverse PPC with VM j, so that their aggregate resource correlation relative to the physical server's 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 can be placed to the servers compactly.

5.2 Resource Demand Prediction

Resource demand is taken as one of the inputs in our algorithms. It should be estimated periodically at run-time. Despite a lot of works on demand predictions [8], [32], [45], we choose a simple yet efficient one. We periodically 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. In practice, EWMA has a high level of accuracy. Our experiments demonstrate the prediction errors are less than 4% for 95^{th} percentiles of estimations.

5.3 Resource Allocator

For CPU resource allocation, Xen hypervisor provides several schedulers for fair allocation of CPU resource in different scenarios [9]. We use the credit scheduler [9] in our prototype system as it is most commonly used in real deployments. Xen hypervisor provides a convenient interface to configure VMs' CPU weights (or shares). Weighting mechanism guarantees that CPU cycles allocated to different VMs are proportional to their weights. Moreover, the credit scheduler supports a non-workconserving mode, in which VMs' CPU time can be capped at their demands. We also use capping mechanism to guarantee that no VM receives more CPU resource than its demand.

For memory allocation, Xen hypervisor also provides an interface to dynamically adjust each VM's memory capacity through ballooning mechanism [7], [43]. One limitation of ballooning mechanism is that dynamic memory adjustment is constrained to the VMs' *max_memory* configuration. In our previous work, we had developed a memory hotplugging technique for VMs [23] 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 determines how many memory shares a VM should be allocated in each time of memory adjustment.

5.4 Resource Pool Scaling

Tenants can still acquire or release resources by increasing/decreasing the number of VMs according to the scale-out model. However, tenants should suffer the latency involved in VM instantiating. In addition, our model supports fine-grained (i.e., CPU and DRAM) resource allocation to VMs via dynamic payment. For example, 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. 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.

5.5 Load Balancing

To handle load imbalance of different physical nodes, our prototype supports live VM migration for load management. VM live migration can also facilitate resource trading between tenants. A sophisticated migration strategy needs to handle the 3W (When, Which and Where) problems. We adopt a hotspot detection strategy [46] to determine *when* VM migrations are required. We use a migration performance model [27] to determine *which* VM should be migrated. Furthermore, we determine the target host *where* a VM should be migrated using the VM grouping scheme, as described in Subsection 5.1.

We should mention that VM migration over a WAN usually results in a high latency of VM instantiating and service downtime [25], [26], [27]. In this paper, we only consider VM migration within a single cloud data center. In the future, we can incorporate our previous work [25], [26] to study WAN delays in F2C.

5.6 IaaS Cloud Economics

Recently, we have witnessed the prosperousness of a group-buying mechanism in product and service markets, which is essentially a kind of marketing strategy with benefits for both buyers and sellers [16], [40]. This strategy motivates

cloud users not only to buy resource at a lower price, but also to provision resource based on average usage while sharing resource with other tenants. On the other hand, the sales amount of cloud provider may increase due to discounted price and competition advantages. In fact, cloud vendors such as Amazon EC2 already provide discounted pricing when tenants buy in bulk (e.g., reserved instances). Cloud provider can exploit different pricing models to enlarge their revenue. A detailed discussion of IaaS cloud economics is beyond the scope of this paper.

6 EVALUATION

We evaluate F2C in terms of fairness of multi-resource allocation, improvement of application performance and VM density, and performance overhead.

6.1 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.

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

TPC-C: it is an on-line transaction processing (OLTP) benchmark [41]. We use a public benchmark DBT-2 as clients, to simulate a number of users executing transactions against a database. The clients and MySQL database run in two VMs separately. We assume these two VM belong to the same tenant. We configure 600 terminal threads and 300 database connections. We evaluate its application performance by throughput (transactions per minute).

RUBBoS (Rice University Bulletin Board System) [34]: 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. For 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 DB tier. We evaluate its application performance by 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. Each worker VM run 800 map jobs and followed by 400 reduce jobs. The 10 worker VMs are evenly distributed in the 10 physical machines and co-run with other workloads. The map stage take almost 95% of the total execution time. The WordCount workload is CPU and memory bound and shows regular resource demands.

TABLE 4: Workloads' aggregated CPU/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 more applications. All VMs are placed on servers based on our VM grouping algorithm. To study the impact of resource allocation on application performance, we first use an kernel-based profiling scheme to monitor applications' resource usage, as shown in Table 4, 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 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 [44].

The dynamic resource allocation algorithm is periodically executed at each node (domain 0). By default, the interval (or *window size*) is set to 5 seconds for the purposes of fine-grained resource sharing. We evaluate F2C by comparing the following approaches for IaaS clouds:

- **T-shirt (static)**: Workloads are running in VMs with static resource provision. It is the current resource model adopted by most IaaS clouds [14].
- **WMMF** (Weighted Max-Min Fairness): WMMF [21] is used to allocate CPU and Memory resources to each VM separately.
- **DRF** (Dominate Resource Fairness): DRF [12] is used to allocate multiple resources to each VM.
- **IWA** (Intra-tenant Weight Adjustment): We conduct weight adjustment for VMs belonging to the same tenant, without considering inter-tenant resource trading.
- **RRF (IRT + IWA)**: We conduct hierarchical resource allocation using both inter-tenant resource trading and intra-tenant weight adjustment.

6.2 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2499769, IEEE Transactions on Parallel and Distributed Systems

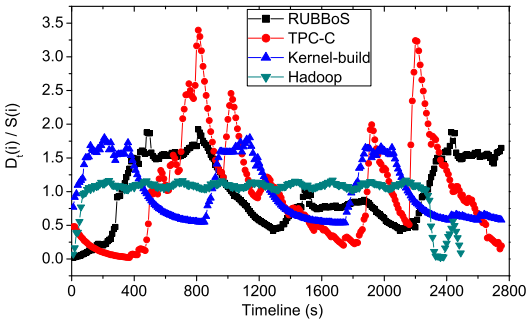

Fig. 5: The ratio of total resource demand to total initial share, i.e., $D_t(i)/S(i)$, varies with time.

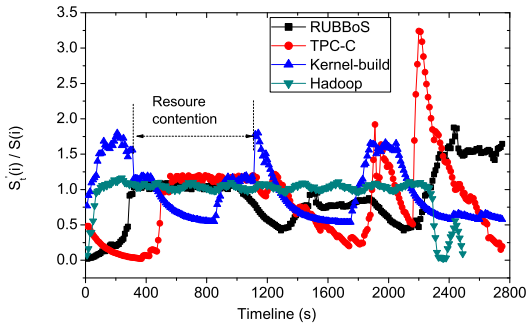

Fig. 6: The ratio of total resource allocation to total initial share, i.e., $S_t^{'}(i)/S(i)$, varies with time.

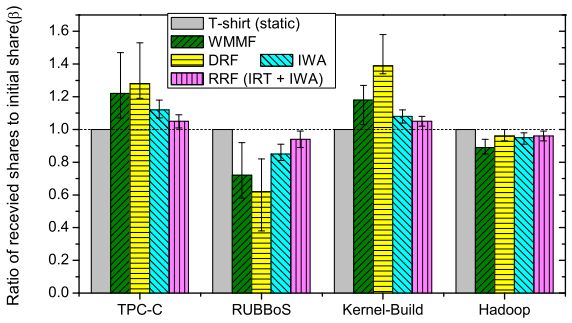

Fig. 7: Comparison of economic fairness of several resource allocation schemes.

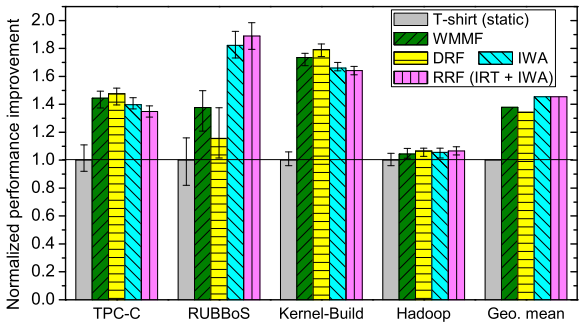

Fig. 8: Comparison of application performance improvement of several resource allocation schemes.

workload to its average demand (initial resource share). Figure 5 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 exceeds 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 satisfy the resource demand of each workload.

Figure 6 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-provisioned 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.

6.3 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). Thus, 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 while $\beta(i) < 1$ implies the tenant loses her asset.

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 7 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 8.

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-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2499769, IEEE Transactions on Parallel and Distributed Systems

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 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 tries to maximize the value of 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 provisioned for applications), the β of all applications approach to one. In contrast, a larger α leads to a decrease of β. That means applications tends to preserve their resource when there exist intensive resource contention. Nevertheless, a larger value of α implies better application performance.

6.4 Improvement of Application Performance

Figure 8 shows the normalized application performance for different resource allocation schemes. All schemes provision resource at applications' average demand ($\alpha = 1$). In the T-shirt model, all applications show the worst performance and we refer it as a baseline. In other models, all applications show performance improvement due to resource sharing. For RUBBoS, RRF leads to much more application performance improvement than other schemes. This is because, 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 7. 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. 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, the performance of Hadoop shows slight variations between different allocation schemes due to its rather stable resource demands.

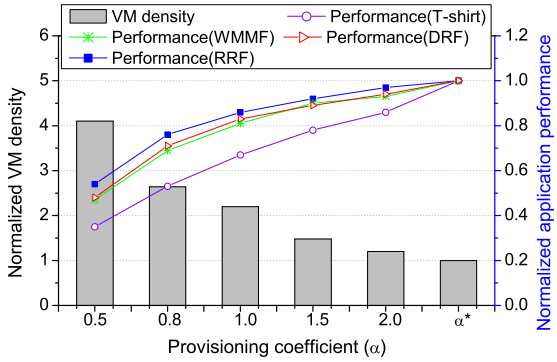

Fig. 9: Application performance corresponds to VM density.

6.5 VM Density and Tenant Cost

VM density is a very important metric on cloud resource efficiency. Higher VM density usually implies a higher resource utilization and more revenue for cloud providers. Figure 9 shows the geometric mean of normalized applications performance and VM density with varying α values. $\alpha = \alpha^*$ denotes each VM is provisioned at its peak demands. We refer the setting α^* as a baseline for comparison. RRF and other dynamic allocation schemes always show much better application performance than 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 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 tradeoff between resource efficiency and quality of service. We conjecture that cloud providers can serve more tenants and increase revenue due to higher VM density.

From the perspective of tenants, resource sharing implies significant cost saving. Figure 10 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.

6.6 Cloud Provider Revenue

In this section, we analyze the impact of resource multiplexing on the cloud provider's revenue. For comparison, we compare F2C with T-shirt model, a join-VM resource provisioning approach [30] and a resource overbooking approach [42]. The joint-VM provisioning approach [30] advocates that multiple VMs should be consolidated and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2499769, IEEE Transactions on Parallel and Distributed Systems

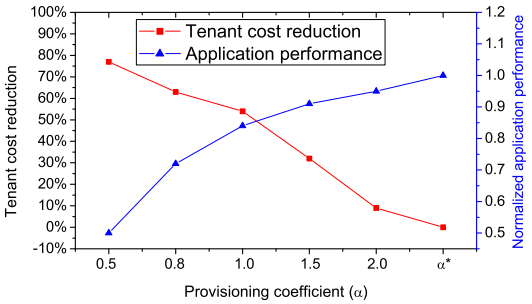

Fig. 10: The trade-off between tenant cost reduction and application performance, compared to T-shirt model.

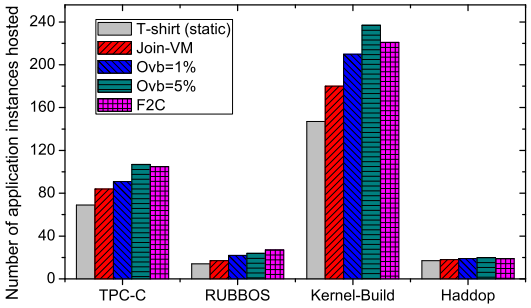

Fig. 11: Benefits of resource multiplexing for different workloads, under T-shirt, join-VM provisioning, resource overbooking (Ovb) and F2C approaches.

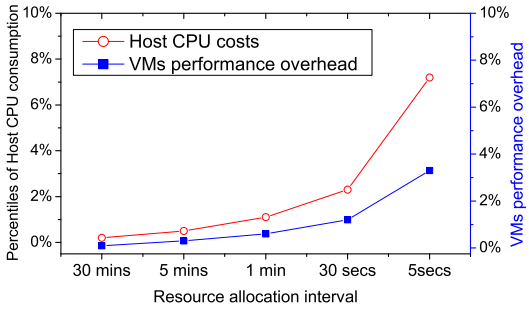

Fig. 12: Performance overhead of F2C varies with the frequency of resource dynamic allocations.

provisioned together, according to their aggregate capacity requirements, to exploit statistical resource multiplexing among the multiple VMs. Similarly, the resource overbooking strategy [42] allows users to provision resources based on a high percentile of the application demands while employing statistical resource multiplexing among different users to maximize the revenue of hardware resource.

We conduct experiments to examine the efficiency of resource multiplexing in our cluster. For each type of workload described in Subsection 6.1, we continuously place as many the same application as possible on the cluster until there is no room to accommodate any more applications. For joint-VM provisioning and resource overbooking approaches, we implement simulation programs based on their core ideas and use the same parameter settings as described in [30] and [42], respectively.

Figure 11 shows the number of application instances the platform can support. The T-shirt model shows the lowest application density in the cluster as the applications are provisioned according to their peak resource demands. Although the same type of application instances have similar workload patterns, there is still opportunities of resource multiplexing on the temporal dimension, i.e., the peaks and valleys of different VMs' resource demands may not necessarily coincide with the others because they are not started concurrently. For join-VM provisioning, it can lower the peak demand of multiple VMs' aggregate capacity requirements compared to individual-VM based provisioning. It leads to much higher resource utilization than T-shirt model. However, as the aggregate peak demands may still much larger than the average demands, join-VM provisioning is not able to fully utilize the over-provisioned resource at any time. For the overbooking approach, the more resource overbooked, the larger is the number of application instances that can be hosted on the cluster. Specifically, the number of RUBBoS application that can be hosted by the cluster increases from 14 (T-shirt model) to 25 for 5% overbooking (a factor of 1.8 increase). For F2C, the experimental results demonstrate that it show comparable feasibility and benefits of resource multiplexing with the overbooking approach. Essentially, F2C is an overbooking approach. The proportion of resource overbooked can be configured by adjusting the provisioning coefficient α.

We also find that the feasibility and efficacy of resource multiplexing mainly depends on the workload pattern. For example, the Hadoop application demonstrates slight fluctuation of resource demands, so there is less opportunities for those applications to exploit resource multiplexing. Even 5 percent resource is overbooked, cloud providers can gain trivial benefit from the overbooking strategy.

6.7 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, as shown in Figure 12. 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 find that the overhead is negligible relative to the gain from resource sharing.

We also study time cost of resource allocation under scale-out model vs. under scale-up model. We choose the Hadoop workload as a case study. Suppose a typical VM has fix-sized resource capacity (for example, 1 CPU core and 1 GB RAM). To add the same amount of resource capacity, scale-out model needs to instantiate a number of VMs from scratch, while scale-up model only needs to reconfigure the existing VMs' resource shares. Figure 13 shows that the overhead of scale-up model is negligible compared to the scale-up model. We also study VM instantiation in different approaches: instantiating VMs one by one, i.e., Sequential VM Instantiation (SI), Parallel VM Instantiations on a Single Host (PISH), and Parallel VM Instantiations on Multiple Hosts (PIMH). For SI, the

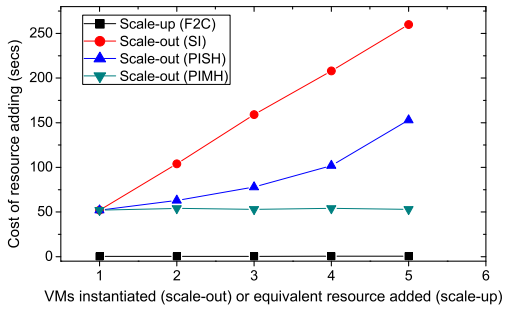

Fig. 13: Resource allocation overhead of scale-out model and scale-up model (F2C).

total completion time of VM instantiations is proportional to the number of VMs. For PISH, the instantiation time of each VM increases with the number of VMs that are instantiating concurrently. More VM Instantiations results in higher degree of resource contention in a single host. In contrast, PIMH distributes the load of VM instantiations on multiple hosts, and demonstrates the best scalability.

7 Related Work

We review the related work in the following two categories.

Resource sharing and provisioning. 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 [23], page-sharing [7], [43], and CPU & I/O scheduling [9], [17]. There have been a number of studies focusing on VM resource multiplexing strategies to improve resource utilization. Virtual-putty [38] exploited affinities and conflicts between co-located VMs to improve the host resource utilization. Meng *et al.* [30] proposed a joint-VM provisioning strategy to consolidate multiple VMs based on the estimate of their total resource requirement. v-Bundle [19] 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 and scale-up model [3], [14]. CloudScale [36] is a trace-driven elastic resource scaling system. The challenging problem is that resource scaling results in a high latency due to VM instantiating [3]. Furthermore, several other trace-based approaches [32], [45] are developed for demand prediction and capacity provisioning. F2C is complementary to these techniques, in the sense that it enables fine-grained sharing among VMs within one tenant and among multiple tenants.

Fairness of resource sharing: (Weighted) max-min fairness is one of the most widely used fairness mechanisms [21]. Many data center or cluster schedulers supports max-min fairness or its extensions, such as Hadoop's Fair Scheduler [1], Quincy [20], Mesos [18], and Choosy [13]. For example, Quincy [20] achieves fairness of scheduling concurrent distributed jobs by formulating the problem as a min-cost flow problem. Mesos [18] achieves fair resource sharing by using DRF [12]. Our previous work proposed Long-Term Resource Fairness (LTRF) for the single resource type [39]. Most of previous studies are at the level of fixed-size partitions of nodes, or tasks. In contrast, F2C 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. Pisces [37] is a systemic implementation of max-min fairness in multi-tenant shared key-value storage system. mclock [17] supports proportional-share fairness for disk I/O resource allocation in VMware ESX. U-tube supports proportional share fairness for memory load balancing among co-located VMs [23]. Flex [33] reinforces fairness at VM-level scheduling and improves the efficiency of SMP VMs. Shanmuganathan et al.'s work [35] 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 resources. This paper goes beyond our preliminary study [24] by (1) developing a full-fledged resource allocation system for IaaS clouds, and (2) performing extensive end-to-end comparison at the system level.

8 Conclusion

Efficient and fine-grained resource sharing becomes increasingly important and attractive for new-generation cloud environments. This paper presents *F2C*, a cooperative resource management system for IaaS clouds. To address the economic fairness issues, we proposes *reciprocal resource fairness* (RRF) to enable fine-grained multi-resource fair sharing among multiple tenants. We implement F2C on Xen platform. The experimental results show that F2C is beneficial for both cloud providers and tenants. For cloud providers, F2C improves VM density of current IaaS cloud models by 2.2X and has almost negligible runtime overhead for real-time resource sharing. For tenants, F2C delivers better application performance and 95 percent economic fairness among multiple tenants.

Acknowledgments

The work of Haikun Liu is supported by NSFC under grants No. 61300040. Bingsheng is partially supported by a MoE AcRF Tier 1 (2014-T1-001-145) of Singapore.

References

[1] http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html/.
[2] Amazon, *http://aws.amazon.com/solutions/case-studies/*.
[3] AutoScaling, *http://aws.amazon.com/autoscaling*.
[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM Conf. (SIGCOMM '11)*, 2011, pp. 242–253.
[5] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. 10th USENIX Conf. on Networked Systems Design and Implementation (NSDI'13)*, 2013, pp. 171–184.
[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*, vol. 31, no. 1, pp. 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 *Proc. ACM Symp. Operating Systems Principles(SOSP'03)*, 2003, pp. 164–177.

[8] M. N. Bennani and D. A. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. International Conf. Autonomic Computing*, 2005, pp. 229–240.

[9] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[10] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A updated survey," in *Approximation algorithms for NP-hard problems*, 1996, pp. 46–93.

[11] E. Deelman, "Grids and clouds: Making workflow applications work in heterogeneous distributed environments," *IJHPCA*, vol. 24, no. 3, pp. 284–298, 2010.

[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: fair allocation of multiple resource types," in *Proc. 8th USENIX conf. Networked systems design and implementation (NSDI'11)*, 2011, pp. 1–14.

[13] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: max-min fair sharing for datacenter jobs with constraints," in *Proc. ACM European Conf. Computer Systems (EuroSys'13)*, 2013, pp. 365–378.

[14] D. Gmach, J. Rolia, and L. Cherkasova, "Selling t-shirts and time shares in the cloud," in *Proc. 12th IEEE/ACM Symp. Cluster, Cloud and Grid Computing (CCGRID'12)*, 2012, pp. 539–546.

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

[16] Groupon, *http://www.groupon.com/*.

[17] A. Gulati, A. Merchant, and P. J. Varman, "mclock: handling throughput variability for hypervisor io scheduling," in *Proc. USENIX conf. Operating Systems Design and Implementation(OSDI'10)*, 2010, pp. 437–450.

[18] 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 *Proc. 8th USENIX Conf. Networked Systems Design and Implementation (NSDI'11)*, 2011, pp. 1–14.

[19] L. Hu, K. D. Ryu, D. Da Silva, and K. Schwan, "v-bundle: Flexible group resource offerings in clouds," in *Proc. International Conf. Distributed Computing Systems (ICDCS'12)*, 2012, pp. 406–415.

[20] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. ACM SIGOPS Symp. Operating Systems Principles (SOSP'09)*, 2009, pp. 261–276.

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

[22] J. Lee Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[23] 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 Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1350–1363, 2015.

[24] H. Liu and B. He, "Reciprocal resource fairness: Towards cooperative multiple-resource fair sharing in iaas clouds," in *International Conf. High Performance Computing, Networking, Storage and Analysis (SC'14)*, 2014, pp. 970 – 981.

[25] ——, "Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1192–1205, 2015.

[26] H. Liu, H. Jin, X. Liao, C. Yu, and C. Xu, "Live virtual machine migration via asynchronous replication and state synchronization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 1986–1999, 2011.

[27] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proc. 20th International Symp. High Performance Distributed Computing (HPDC'11)*, 2011, pp. 171–182.

[28] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *IEEE 5th International Conf. Cloud Computing (CLOUD'12)*, 2012, pp. 423–430.

[29] M. Maurer, I. Brandic, and R. Sakellariou, "Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures," in *IEEE International Conf. Cloud Computing (CLOUD'12)*, 2012, pp. 368–375.

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

[31] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proc. 10th International Conf. Autonomic Computing (ICAC'13)*, 2013, pp. 69–82.

[32] 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 *Proc. 4th ACM European Conf. Computer Systems(EuroSys'09)*, 2009, pp. 13–26.

[33] J. Rao and X. Zhou, "Towards fair and efficient smp virtual machine scheduling," in *Proc. 19th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'14)*, 2014, pp. 273–286.

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

[35] G. Shanmuganathan, A. Gulati, and P. Varman, "Defragmenting the cloud using demand-based resource allocation," in *Proc. ACM SIGMETRICS International Conf. Measurement and Modeling of Computer Systems (SIGMETRICS'13)*, 2013, pp. 67–80.

[36] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proc. 2nd ACM Symp. Cloud Computing (SoCC'11)*, 2011, pp. 1–14.

[37] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proc.10th USENIX Conf. Operating Systems Design and Implementation*, 2012, pp. 349–362.

[38] J. Sonnek and A. Chandra, "Virtual putty: Reshaping the physical footprint of virtual machines," in *Proc. 2009 Conf. Hot Topics in Cloud Computing (HotCloud'09)*, 2009, p. 11.

[39] 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 *Proc. 28th ACM International Conf. Supercomputing (ICS'14)*, 2014, pp. 251–260.

[40] Teambuy, *http://www.teambuy.ca/*.

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

[42] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in a shared internet hosting platform," *ACM Trans. Internet Technol.*, vol. 9, no. 1, pp. 1:1–1:45, Feb. 2009.

[43] C. A. Waldspurger, "Memory resource management in vmware esx server," in *Proc.5th Symp. Operating Systems Design and Implementation (OSDI'02)*, 2002, pp. 181–194.

[44] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: handling memory overload in an oversubscribed cloud," in *Proc.7th ACM SIGPLAN/SIGOPS International Conf. on Virtual Execution Environments (VEE'11)*, 2011, pp. 205–216.

[45] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proc. 9th ACM/IFIP/USENIX International Conf. Middleware (Middleware'08)*, 2008, pp. 366–387.

[46] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th Conf. Networked Systems Design and Implementation*, 2007, p. 17.

Haikun Liu received the Ph.D. degree in Huazhong University of Science and Technology, China. He was the recipient of outstanding doctoral dissertation award in Hubei province, China. He is an associate professor in School of Computer Science and Technology, HUST. His current research interests include virtualization technologies, cloud computing, and distributed systems.

Bingsheng He received the bachelor degree in computer science from Shanghai Jiao Tong University, and the PhD degree in computer science in Hong Kong University of Science and Technology. He is an associate professor in School of Computer Engineering, Nanyang Technological University, Singapore. His research interests are high performance computing, cloud computing, and database systems. He has been awarded with the IBM Ph.D. fellowship (2007-2008) and with NVIDIA Academic Partnership (2010-2011).