Microprocessors and Microsystems 52 (2017) 365–380

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Joint affinity aware grouping and virtual machine placement

Jianhai Chen[a], Qinming He[a], Deshi Ye[a,*], Wenzhi Chen[a], Yang Xiang[b], Kevin Chiew[c], Liangwei Zhu[a]

[a] College of Computer Science, Zhejiang University, Hangzhou, China
[b] School of Information Technology, Deakin University, Burwood, Vic. 3125, Australia
[c] Handal Indah Pte. Ltd., Singapore

ARTICLE INFO

Article history:
Received 2 February 2016
Revised 25 November 2016
Accepted 9 December 2016
Available online 13 December 2016

Keywords:
Virtualization
Affinity grouping
Resource allocation
VM colocation
Affinity scheduling
Bin packing

ABSTRACT

The **N**on-**A**ffinity **A**ware **G**rouping based resource **A**llocation (NAGA) method toward the ***G**eneral VM**P**lacement (GP) problem* enables (1) some VMs to be co-located onto the same PM while the VMs are required to be placed onto distinct PMs; and (2) some VMs to be dispersedly placed onto distinct PMs while the VMs are required to be co-located onto the same PM, leading to a serious performance degradation of application running over multiple VMs in cloud computing. In this work we study an ***A**ffinity **A**ware VM **P**lacement (AAP)* problem and propose a ***J**oint **A**ffinity **A**ware**G**rouping and **B**in **P**acking (JAGBP)* method to remedy the deficiency of the NAGA method. We firstly introduce affinity of VMs to identify affinity relationships to VMs which are required to be placed with a special VM placement pattern, such as colocation or disperse placement, and formulate the AAP problem. Then, we propose an affinity aware resource scheduling framework, and provide methods to obtain and identify the affinity relationships between VMs, and the JAGBP method. Lastly, we present holistic evaluation experiments to validate the feasibility and evaluate the performance of the proposed methods. The results demonstrate the significance of introduced affinity and the effectiveness of JAGBP method.

1. Introduction

Cloud computing has been a paradigm for delivering computing services to users with an abstraction of scalable, unlimited computing resources on a pay-as-you-go basis and it runs applications to provide services for users on remote datacenters over the Internet [9]. Nowadays many large companies, such as Amazon, Google, Microsoft and Alibaba, have built public clouds successfully. More and more enterprises set up private clouds managed by frameworks such as VMware vCloud [1], OpenStack [5], and CloudStack [18], paving a new commercial business model with migration of business applications to clouds. Due to a recent survey of IT decision makers of large companies, 68% of the respondents estimate that more than 50% of their companies' IT services will be transmitted to cloud platforms by the end of 2014 [26].

The success of cloud computing partly ascribes to virtualization. Virtualization has become a crucial technology of cloud computing and built the essential infrastructure for clouds. It enables server consolidation and live migration, and brings the dynamical resource scheduling and allocation into reality for clouds. Various applications, such as High Performance Computing (HPC) applications [21], multi-tiers web applications [17], parallel applications [14], and big data processing applications [2], are encapsulated into multiple virtual machines (VMs) which are dynamically allocated to a large pool of physical machines (PMs) [8,32,44].

With the growing number of users using clouds, the VM placement resource scheduling and allocation has become an increasingly important problem of current datacenter [20]. Many cloud datacenters currently are still at a very low resource utilization and need efficient resource allocation methods [6]. The goal of resource allocation is commonly to maximize datacenter revenues or the ratio between performance and cost. It involves performance efficiency as well as energy efficiency. The performance efficiency requires the allocation to minimize application performance cost using virtualization. The energy efficiency requires the allocation to maximize resource utilization, namely, minimize the number of PMs.

Besides, as virtualization brings itself performance cost, some VMs running applications are required to be placed with a specific placement pattern, i.e., VMs colocation or disperse placement [15,30]. For examples, the running of many communication-/data- intensive applications inside multiple VMs generates net-

* Corresponding author.
E-mail addresses: chenjh919@zju.edu.cn (J. Chen), hqm@zju.edu.cn (Q. He), yedeshi@zju.edu.cn (D. Ye), chenwz@zju.edu.cn (W. Chen), yang@deakin.edu.au (Y. Xiang), kchiew@handalindah.com.my (K. Chiew), zhulw@zju.edu.cn (L. Zhu).

http://dx.doi.org/10.1016/j.micpro.2016.12.006
0141-9331/

work communication traffics or data traffics amongst VMs [25]. Colocating VMs with traffics can reduce network communication overhead and improve application performance. While co-locating the VMs onto the same PM can cause competition of shared computing resources, such as CPU, memory, disk and network I/O bandwidth, etc. The special demand of VM placement pattern brings a dependency or relationship for VMs running applications.

Nevertheless, considering the dependency between VMs, the VM placement resource scheduling and allocation bring forward two major challenges:

(1) Obtaining the dependency of VMs with an effective placement pattern for running applications efficiently is not a trivial task. It requires a detailed analysis of application program behaviours or features and a holistic performance evaluation for running application under all kinds of distinct VM placement schemes.
(2) Solving the problem of VM placement resource scheduling and allocation is always an optimization problem, such as the bin packing problem. Due to bin packing is an NP-hard problem, making decision of an optimal VM placement resource allocation solution for both minimizing the number of PM to guarantee energy efficiency and guaranteeing the dependency between VMs is a big challenge.

There have been some research works addressing the challenges and proposed some methods on resource allocation [10,34,39–42]. However, current methods rarely consider the dependency of VMs and lack a general approach to generalize the dependency of VMs for VM placement. Further, there still are no general efficient methods to obtain and identify relationships between VMs in VM placement. Although some methods solve the general VM placement resource allocation as a bin packing problem, without consideration of the dependency or relationship between VMs, the bin packing methods will enable the VMs which are required to be colocation placed onto the same PM but not, and the VMs which are required to be dispersedly placed onto distinct PMs but not, causing a certain level of application performance degradation [46].

In this paper, we target to remedy the deficiency of the general resource allocation methods like bin packing and focus on both minimizing the application performance cost and maximizing the resource utilization. At first, by conducting experiments in a testbed with several typical cloud applications running between multiple VMs, we do performance evaluation by running applications under two different VM placement patterns, namely, VMs colocation placement and dispersedly placement with all kinds of combination of VMs and PMs. Motivated by the observation results, we introduce affinity to denote the dependency between VMs and define affinity relationships between VMs for VM placement and state an affinity aware VM placement (AAP) problem. Then, we propose a *Joint Affinity Aware Grouping and Bin Packing* (JAGBP) method to solve the AAP problem, including an affinity grouping algorithm and several heuristic bin packing algorithms. At last, we present three experiments to validate efficiency of the proposed methods, including (1) experiments on obtaining and identifying the affinity relationships between VMs, (2) a simulation experiment on verifying the run efficiency of algorithms, and (3) a real cloud environment experiment to illustrate the effectiveness of the JAGBP method through constructing seven virtual clusters (VCs) configured with 59 VMs for running typical cloud applications, including the HPCC and RUBiS benchmarks. By comparing to the Non-affinity aware grouping allocation (NAGA) method, the JAGBP method significantly improve the application performance better than the NAGA method.

In brief, our contributions are as follows.

(1) We model affinity between VMs for VM placement. In the model, we give the methods to find and identify affinity relationships between VMs, involving the performance measurement, analysis and evaluation of typical cloud virtualization applications.
(2) We present an affinity aware VM placement (AAP) problem and the JAGBP method. To the best of our knowledge, we are the first to present AAP problem and propose JAGBP method.
(3) We provide an affinity aware VM placement framework for cloud computing system resource scheduling. It integrates affinity obtaining methods and resource scheduling algorithms which can be efficiently used to advance the practical cloud computing platforms.
(4) We conduct overall experiments to demonstrate the effectiveness of our proposed methods.

The remaining sections are organized as follows. We present the related work in Section 2, the background and motivation of our work in Section 3. Next, we model the affinity and affinity relationship, and state the affinity aware placement problem in Section 4 and we propose solutions to solve the problem in Section 5, followed by experiments in Section 6. Finally we give conclusion in Section 7.

2. Related work

This paper presents an approach for VM placement resource allocation in cloud computing systems, considering both optimization of application performance and resource utilization to provide highly performance cost ratio for cloud. Here, we discuss related work in the literature related to similar issues.

2.1. Affinity and virtualization performance studies

Many research on affinity and virtualization performance has been conducted in cloud computing.

Chen and Li et al. proposed affinity as a property of VMs to identify the relation between a virtual CPU and CPU core in VMM or Hypervisor to implement a new schedule strategy to improve the efficiency of virtualized resource scheduling under a single virtualized system in cloud computing [4]. Sonnek et al. presented an affinity-aware VM migration technique to minimize the communication overhead on a virtualized platform [33]. The affinity is identified as a policy or a technique of VM migration for a dynamic resource allocation. Yan, Cairong et al. discussed an affinity aware virtual cluster optimization method for Mapreduce applications placement [43]. The affinity is defined as relationships between virtual clusters and obtained by measuring the latency distance between network virtual machines. The proposed affinity in this paper denotes relationships between VMs associated with special VM placement patterns, colocation placement and no-colocation placement.

Sudevalayam et al. put forward an affinity aware modeling of CPU usage method to evaluate performance of virtualized applications hosted onto two VMs with colocation placement or disperse placement [35]. They focus on CPU usage-based application performance evaluation under VM colocation or disperse location scenarios but not resource allocation. Two years later, they described an affinity aware modeling of CPU usage with communicating virtual machines and develop pair-wise affinity-aware models to predict expected CPU resource requirements [36]. Their proposed performance evaluation methods paved a good way to obtain affinity of VMs for our work. Our work presents a general affinity of VMs and affinity relation model for VM placement resource allocation. Meng et al. introduced a traffic-aware VM placement to improve the network scalability [25]. The network traffic generates communications. The authors did not address affinity for VM placement but the traffic property defines a kind of affinity for our work. Re-

cently, Zhang, Wei et al. provided an interference minimization optimization method for Hadoop virtual machines placement. The interference between VMs running Hadoop application also denotes a kind of affinity for our work [47].

Besides, some Enterprise virtualization products such as VMWare addressed a simple resource management system which employed affinity to signify a set of VMs which are required to be kept together as one unit to do colocation in VM placement [12]. It proves the practicality of our study on affinity-aware grouping resource allocation.

2.2. Resource allocation, scheduling for virtual machine placement

Virtualization reduces complexity of resource scheduling allocation and eases resource management in VM placement. In large scale datacenter or cloud computing systems, the network performance optimization is a significant issue. Yet many solutions directly improve the network performance by changing network architecture and routing protocols in modern datacenters, but rarely by optimizing the VM placement [11,27].

While in virtualized datacenters, more and more researches have developed a lot of efficient methods to improve performance by designing efficient algorithms to optimize VM placement. Stillwell, M. et al. proposed various algorithms for the VM placement resource allocation problem [34]. Wilcox et al. solved the VM placement as vector packing with a grouping genetic algorithm [40]. Panigrahy et al. addressed the heuristics vector bin packing based on FFD approximate approaches to the vector packing problem [29]. Tordsson, J. et al. proposed cloud brokering mechanisms for optimized placement of virtual machines across multiple providers [38]. Jay Smitha et al. investigated a robust static resource allocation problem for distributed computing systems operating under imposed Quality of Service (QoS) constraints and used a unique application of both path relinking and local search within a Genetic Algorithm [31].

Recently, Depoorter et al. provided a resource management system to advance reservation, co-allocation and pricing of network and computational resources in grids [7]. Xiaoling Li et al. study a resource allocation with multi-factor node ranking in data center networks [19]. These VM allocation solutions adopt VM bin packing techniques, in which the VMs and PMs are multi-dimensional vectors including dimensions like CPU, RAM, disk and network I/O, etc. [40]. We present an affinity aware grouping resource allocation method to minimize system overhead. In other words, these works are non-affinity-awareness but the proposed techniques are helpful to our affinity-aware grouping resource allocation research works.

3. Background and motivation

We are interested in recognizing the correlation between application performance and VM placement patterns. In this section, we firstly present a case study on application performance evaluation as a background, resulting in motivation of our further affinity aware VM placement work.

3.1. VM placement pattern

In VM placement one VM is allocated onto a PM, which means one VM is mapped to a fixed PM, or VM has a fixed PM location. Some VMs cannot only be colocatedly placed onto one PM but also dispersedly placed onto distinct PMs. Certainly we can also choose a specific PM to deploy some VMs. We introduce a notation *VM placement pattern* to denote some VMs placed onto PMs according to how to deploy VMs onto PMs. As far as the number of VMs or PMs is concerned, we use another notation *VM placement scheme* to denote a specific combination of given some VMs and PMs for VM placement corresponding to the VM placement pattern.

Moreover, for any one or two VMs placed onto PMs, we address three general simple VM placement patterns as shown in Fig. 1. At first, if we need to place some VMs into a fixed PM, then we have a *fixed placement* pattern (FP) that is one VM is placed onto a fixed PM. While for any two VMs, there are two patterns as shown in (1) and (2) of Fig. 1. One is (1)*colocation placement* (CP) pattern, that is two or more VMs are *colocation placed* onto the same PM, and the other is (2)*no-colocation placement* (NCP) or *dispersedly placement* (DP) pattern that is two VMs are dispersedly placed onto distinct PMs.

For the simplicity of description in the latter section of this paper, we use a function form CP($\cdot$) or CP(n) (n is the number of given VMs) to denote the *colocation placement* scheme in which considering the number of VMs and the VMs are viewed as a whole group and placed onto the same PM or location. For example, CP(8) denotes 8 VMs as one group placed onto one PM whilst idling the other PM. Furthermore, we use a function form DP($\cdot$) to denote *no-colocation placement* (NCP) or *dispersedly placement* (DP) pattern in which VMs are distributed among multiple PMs or different PM locations. For example, DP($7+1$) denotes eight VMs are divided into two groups with 7 VMs as a group placed onto one PM and 1 VM as a group placed onto the other PM.

3.2. Benchmarking cloud application

We choose a typical cloud application, the *HPC Challenge benchmark* (HPCC) [22]. As a case study, we evaluate HPCC application performance under distinct VM placement patterns and placement schemes. HPCC comprises of seven benchmark applications. It is used to measure the performance of parallel computing in a cluster with several nodes configured with VMs. We choose seven typical metrics for our study, including four metrics from four communication-aware benchmarks, namely, HPL, PTRANS, FFT and Avgpingpong from HPCC benchmark and three memory-intensive benchmarks, namely, STREAM, RandomAccess and DGEMM.

The testbed system involves four homogeneous PMs. All PMs are Dell PowerEdge T710 servers each with dule Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz, totally 16 cores and 16GB RAM running Xen-3.3.1 virtualization technology. We deploy 4, 8 and 16 VMs respectively onto PMs to run the HPCC benchmark applications to measure its performance under all kinds of placement schemes. HPCC application has four main parameters related to workload size, i.e., A, NB, P and Q, in which A denotes the order of the coefficient matrix, NB the partitioning blocking factor, P the number of process rows, and Q the number of process columns. In the case study, Matrix A is set to 1000×1000 and others to be default.

3.2.1. Traffics between VMs on running HPCC

At first, without consideration of the performance, we focus on testing the dependency between VMs derived from traffics between VMs. We create a simple virtual cluster (VC) with four VMs and deploy VMs onto one PM. We run the test three times and obtain the average of the traffic results as shown in Fig. 2. We find that traffics between HPCC VMs are very heavy or frequent.

3.2.2. HPCC performance

Further, we counter-intuitively think that the traffics can incur performance influence for the HPCC application. Then, we create another two VCs with 8 VMs and 16 VMs to run the whole HPCC application many times, summarize the performance metric values and conclude the average value as final results for our performance analysis. According to the number of VM and PM, we separate VMs and PMs into several combination forms for all possible VM placement schemes. For example, for given eight VMs and

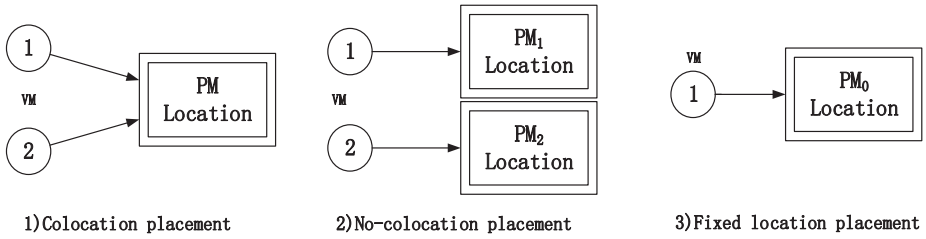


Fig. 1. Three basic placement patterns.

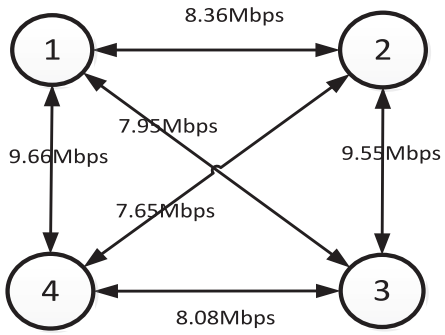


Fig. 2. Traffic amongst 4 VMs (1–4) running HPCC benchmark. The values in the edge between VMs are average traffic rate, concluded by a ratio between total traffic volume in all time intervals and the total runtime, detail in Section 4.

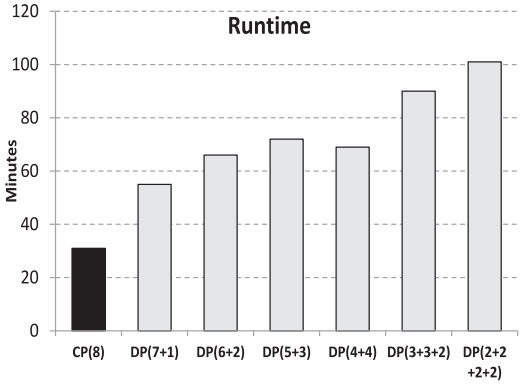


Fig. 3. The runtime performance of HPCC.

two PMs, we can have totally five combinations to allocate these VMs to two PMs, namely CP(8), DP(7 + 1), DP(6 + 2), DP(5 + 3), and DP(4 + 4). Moreover, considering that the number of PMs is three or four, for eight VMs, we also try other two special scenarios: a scheme DP(3 + 3 + 2) where 8 VMs deployed among three PMs, and a scheme DP(2 + 2 + 2 + 2) deployed onto four PMs each with two VMs, respectively. After running HPCC under all schemes we obtain the runtime metric and seven key metrics corresponding to all benchmarks of HPCC for evaluation. At last, for 16 VMs, we only consider three general placement schemes, namely, CP(16), DP(8 + 8), and DP(4 + 4 + 4 + 4) and obtain the HPL benchmark metric for evaluation.

The results are described as follows.

(1) We first compare the runtime performance metric as shown in Fig. 3. We see that CP(8) requires the least runtime as compared with others, meaning that all eight VMs placed onto one PM can complete running with the runtime less than half of that of DP(4 + 4) or one fourth of that of DP(2 + 2 + 2 + 2).

Table 1
HPL comparison of three schemes.

	CP(16)	DP(8+8)	DP(4+4+4+4)
HPL_Tflops	0.0061	0.0018	0.0017

(2) We then compare the performance in terms of the other four different metrics from communication/data- intensive benchmarks under different VM deployment schemes. The results is shown in Fig. 4. It tells that CP(8) outperforms all other schemes for the four metrics; especially, for metric PTRANS, CP(8) is five times better than DP(7 + 1) or 10 times better than others.

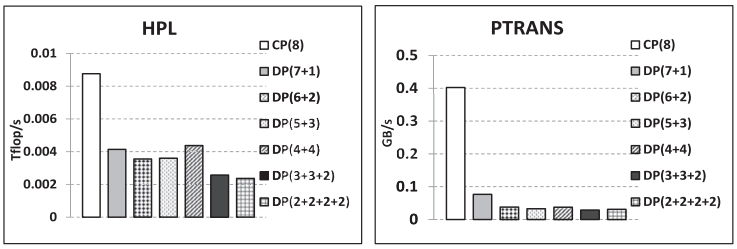
(3) We further compare the performance derived from three non-communication intensive metrics of HPCC. These metrics are memory intensive. From the results shown in Fig. 5, we find that all memory intensive metrics shows little difference under distinct VMs placement schemes.

(4) Table 1 illustrates the comparison of HPL metric for these schemes for 16 VMs used to run HPCC application, from which we confirm that CP(16) outperforms nearly four times than other two schemes so that colocation-placing VMs can obtain much more benefit.

3.3. Motivation

From the above observations of performance evaluation case study, we conclude that for heavy communication intensive applications, VMs colocation placement scheme can obtain much better application performance according to specific concerned metrics. But certainly this may not be true because the performance of the CPU-intensive or memory-intensive applications are restricted by the limit of PM resource capacity because of contention of sharing resource. In other word, with consideration of limited PM resource capacity, in colocation-placement the overloaded allocation VMs onto PMs cannot be sure to improve application performance due to contention of sharing resource between colocated VMs.

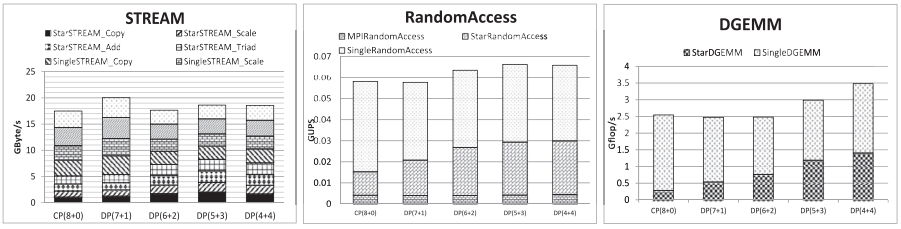
Therefore, in view of application performance, it is meaningful to do co-locating such VMs to run communication intensive applications during the VM placement resource allocation. Then, if the benefit of performance under distinct placement patterns is known before VM allocation decision, it is necessary to identify the VMs with a dependency or relationship. Motivated by this, in the latter section of the paper, we further introduce affinity to denote the dependency or relationship between some VMs running a specific application and in practical VM placement we do placing the VMs according to the affinity rules. When the VMs are identified with affinity relationships, we can also offer benefit to do affinity aware grouping and placing VM groups with affinity as a whole unit to PMs, so as to guarantee the affinity.



(a) Floating point rate of execution HPL (b) Total communication capacity of the system interconnect

(c) Floating point rate of running DFT based on MPI (d) Communication bandwidth

Fig. 4. Communication-intensive benchmark metrics of the HPCC benchmark performance evaluation amongst different metrics in distinct CP and DP schemes.



(a) Memory sustainable bandwidth (b) Updating integer rate to random memory locations (c) Floating point rate of executing matrix-matrix multiplication

Fig. 5. Non-communication intensive benchmarks of the HPCC performance evaluation against different metrics.

4. Problem statement

In this section, we firstly provide some definitions for modelling affinity of VMs and then state the Affinity Aware VM Placement (AAP) problem.

4.1. Definition

Definition 1. Affinity. The affinity is defined as a dependency between one or more VMs with a special placement pattern running a special application workload.

The affinity can be derived from the communication or data transmission traffics between the two VMs generated during the running of application. Generally, to reduce performance cost of virtualization and offer better application performance in VM resource allocation, the two VMs are required to be colocation placed onto one PM with enough CPU, MEM or placed onto two PMs with enough network bandwidth and located locally. We further extend the meaning of affinity conception to a general conception that we can identify a dependency to VMs if the VMs are required to have a special placement pattern.

The affinity of VMs means a relationship between VMs. According to relation theory, we define affinity relation as follows [28].

Definition 2. Affinity Relation. The affinity relation is defined as a relationship between two VMs with affinity property.

According to the basic VM placement patterns, we define three affinity types and affinity relations, namely, *colocation placement affinity/relation* (co-affinity/relation), *no-colocation placement affinity/relation* (noco-affinity/relation) and *fixed placement affinity/relation*(fixed-affinity/relation).

4.2. Affinity aware VM placement problem

Without consideration of affinity, we call the VM placement as the **G**eneral VM **P**lacement (GP) problem. In the following, we state the affinity aware placement (AAP) problem by adding affinity as new rules to the GP problem. Specifically, we put forward definitions and integer programming models for both the GP and AAP problem.

4.2.1. The GP problem and IP-GP model

As a resource allocation problem, a VM can be viewed as an item and a PM as a bin. We formalize the GP problem as a bin packing or VM packing (VMP) problem which is described detail as follows [40].

Definition 3. General VM placement problem (GP). Given a set $V = \{v_1, v_2, \ldots, v_n\}$ with n VMs and a set $P = \{p_1, p_2, \ldots, p_m\}$ with m homogeneous PMs, each VM item has a **d**-dimensional resource demand size and PM has a **d**-dimensional resource capacity, finding an optimal placement scheme, subject to matching the PM resource capacity limit (RLC) constraint that for each resource dimension the total size of VMs in each PM cannot excess the capacity size of the PM, such that the number of PM is minimized.

The GP problem is an optimization problem. We describe the Integer Programming model for GP problem (IP-GP) in the following.

Definition 4. IP-GP model. Given n VMs and m homogeneous PMs, we suppose that a PM is a bin and a VM an item. The PM bin and VM item are both d-dimensional resource vectors. The VM items are allocated and packed onto PM bins. For each bin $j \in \{1, \ldots, m\}$ we introduce a binary variable y_j which we set to 1 if bin j is used in the packing, and 0 otherwise. For each VM item $i \in \{1, \ldots, n\}$ and each bin j, we introduce a binary variable x_{ij} which we set to 1 if item i is packed into bin j, and 0 otherwise. Each item i is a resource vector $s_i = (s_i^1, s_i^2, \ldots, s_i^d)$, and each dimension $k \in \{1, \ldots, d\}$ denotes a resource demand size s_i^k which is normalized to 0..1. In each dimension, the capacity of each PM bin is also normalized to 1.

Specifically, the IP-GP model is described as follows.

$$min \sum_{j=1}^{m} y_j \tag{1}$$

subject to

$$\sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1, \ldots, n\}, \tag{2}$$

$$\sum_{i=1}^{n} s_i^k \cdot x_{ij} \leq 1, \forall k \in \{1, \ldots, d\}, j \in \{1, \ldots, m\} \tag{3}$$

$$x_{ij} \leq y_j \tag{4}$$

$$x_{ij} \in \{0, 1\}, y_j \in \{0, 1\} \tag{5}$$

The bin packing of GP problem is NP-hard and we cannot get the optimal VM placement solutions in polynomial time unless P = NP [16]. In next section, we will describe some bin packing heuristic methods to give approximate solutions [29].

4.2.2. The AAP problem and IP-AAP model

Next, we state the affinity aware VM placement problem as follows.

Definition 5. Affinity Aware VM placement Problem AAP. Given a set V of n VMs, a set P of m homogeneous PMs, a set E with e affinity relations between VMs, find an optimal VM placement solution to allocate the VMs onto PMs, subject to matching the **RLC** constraint, such that both the affinity relationships are satisfied and the number of PM is minimized.

Like GP problem we further consider an *Integer Programming* model of the AAP problem (IP-AAP). The IP-AAP model is derived from the IP-GP model considering three affinity relationships. Specifically, corresponding to the basic three affinity relationships, we address three equations as rules and add to the IP-GP model. The equations are illustrated in Eqs. (6)–(8).

Moreover, for the simplicity of description, we introduce an affinity relation matrix $A = (a_{ij})_{i,j \in \{1,\ldots,n\}}$ to denote all affinity relations between VM items. For $\forall i, j \in \{1, \ldots, n\}$, the value of a_{ij} denote a weight value of affinity relation and is set to 0 as default. We distinguish three affinity relation types by setting the elements a_{ij} to distinct values.

(1) At first, we set the diagonal element of A, namely, $a_{ii} (i \in \{1, \ldots, n\})$, a value ($\geq$0) to denote one VM with fixed-affinity or without fixed-affinity. The value set to 0 denotes VM item i has no fixed-affinity, otherwise a positive integer number $k (k \in \{1, \ldots, m\})$ denotes item i has a fixed-affinity and need to be placed onto a fixed PM bin k.

(2) Then, we set other elements $a_{ij} (i \neq j), i, j \in \{1, \ldots, n\}$, to a value to denote the co-affinity and noco-affinity between item i and j. The value set to 0 denotes item i and j have no *colocation/no-colocation affinity*, a positive number ($a_{ij} > 0$) denotes item i and j have a *colocation affinity* relation, and a negative number ($a_{ij} < 0$) denotes item i and j have a *no-colocation affinity*.

The size of value represents a real weight of affinity (affinity degree) which is associated with the value of some application performance metrics, such as the *traffic rate* between VMs or performance revenue value.

Each affinity relation constructs a rule in IP-AAP model. We add the rules to the IP-AAP model for every affinity relation as follows.

- **Colocation affinity rule (Co-affinity)**: let item i and j have a colocation affinity and are required to be colocation placed onto the same bin $k \in \{1, \ldots, m\}$, then item i and item j can use equal binary variable, namely, there exists bin k, $k \in \{1..m\}$, $x_{jk} = x_{ik} = 1$ and $x_{jl} = x_{il} = 0$, where $l \neq k$, and l for all others. In other words, we can combine item i and j to one group item as a new item, and for each dimension d, the size of group item v is denoted by the sum of item i and j, i.e., $s_v^d = s_i^d + s_j^d$. In fact, we use the same binary variable for these two items. For all colocation affinity relations, we add the following constraint (6) to ILP.

$$x_{jk} = x_{ik}, \forall k, \text{ if } a_{ij} > 0 \tag{6}$$

- **No-colocation affinity rule (Noco-affinity)**: we assume item i and j have a no-colocation affinity and are required to be deployed onto two distinct bins k and l ($k \neq l$).
In this case, we add the following constraints (7) to ILP-AAP.

$$x_{ik} + x_{jk} \leq 1, \forall k, \text{ if } a_{ij} < 0 \tag{7}$$

- **Fixed PM affinity rule (Fixed-affinity)**: let item i have a fixed PM affinity associated with a fixed PM bin a_{ii}, namely, is placed onto a fixed PM bin $u = a_{ii}$. Then we have the following corresponding equation.

$$x_{iu} = 1, \forall i, \text{ if } u = a_{ii} \geq 1 \tag{8}$$

The IP-AAP model also can be solved by a faster optimizer of mathematics, such as CPLEX [3] and Gurobi [13]. We use it to conclude a solution. However it lasts even longer time than that of IP-GP model. So it is unrealistic for practical applications when the size of items is increased.

5. Solve the problem

In this section we propose an overall solution to solve the AAP problem with an affinity aware resource scheduling framework and a joint affinity aware grouping and placement method.

5.1. Scheduling framework

We design an *affinity aware resource scheduling* (AARS) framework for cloud system as shown in Fig. 6. We consider a general cloud application scenario in the framework. At first, cloud users or tenants send requests for running applications by renting VMs in cloud computing. Each application runs inside one or more VMs, for example, the cloud App.1 with 2 VMs, App.2 with 3 VMs, and so on. Then the cloud controller receives a list of VMs request from cloud users and execute a resource scheduling or VM resource allocation, such as initial placement, load balancing at every moment. For the affinity aware resource scheduling, we design three modules, i.e., affinity generator, affinity scheduler and schedule decider.

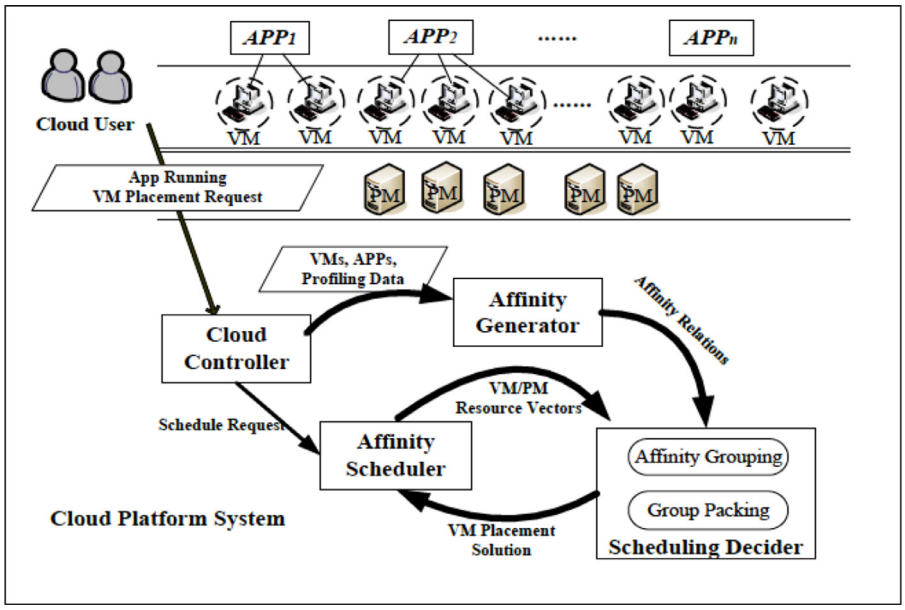


Fig. 6. Affinity aware resource scheduling framework.

- The *affinity generator* is in charge of generating affinity relationships for VMs as resource scheduling rules. During the running of application between VMs, the affinity generator generates the affinity relations between VMs. It receives application workload profiling data, VMs, PMs resource data from cloud controller. It needs many affinity monitor tools responsible for monitoring, obtaining and recording the detailed affinity information of VMs timely. Eventually, the monitoring result information data is summarized to do affinity analysis and conclude whether VMs have affinity or not.
- The *affinity scheduler* is responsible for the control and management of VM affinity aware resource scheduling. It termly checks scheduling request from cloud controller, and determines which type of scheduling should be done, such as initial placement or load balancing. When a schedule request is coming, it immediately sends the request to the scheduling decider to complete scheduling decision.
- The *scheduling decider* is responsible for making resource scheduling decision. The decision results are made up of a list of specific scheduling operations, such as VM migrations, placement of VMs onto PMs, VM boot or shutdown operation. The scheduling decision is realized with some decision algorithms including affinity grouping and affinity group packing algorithms.

5.2. Obtaining and identifying affinity

We can firstly obtain affinity of VMs and identify affinity relationships for the VMs in VM placement. Below we give some principles and methods for obtaining and identification of affinity between VMs.

5.2.1. General principles and methods

We assume in a real cloud scheduling some VMs are known to run specific applications. The applications always have specific program behaviours or features. Then, we identify the affinity to VMs. If we do not know the application performance features, then we do some performance evaluations similar as what we have done in the case study in Section 3. Certainly performance evaluation maybe bring much additional cost. Especially, there are no need to do it for the short-run application programs. But it is useful for clouds to do it for the longrun ones. Therefore, we have a principle with a focus on doing some performance evaluations for the longrun applications to obtain affinity relationships between the multiple VMs.

After all, we obtain affinity for VMs based on the placement demand of a special VM placement pattern. The general principles and methods are as follows. (1)During the performance evaluation, if colocating VMs onto one PM can achieve better performance or lower cost, then identify the VMs with colocation placement affinity. (2)On the contrary, we identify some of two VMs with no-colocation placement affinity. (3)Moreover, if a cloud user has a preference with a demand of placing his VMs onto a fixed PM, then we identify the VMs with a fixed PM placement affinity.

In addition, in order to distinguish the affinity relationship can be weighed with a value as affinity ratio. The value of affinity degree is mapping to a performance metric value.

5.2.2. Colocation placement affinity

We take an example for identifying VMs with colocation placement affinity for VMs running communication intensive applications. We obtain the affinity between VMs by detecting traffics between VM pairs. Actually, the traffic can be easily captured by some traffic monitoring tools such as tcpdump [24]. Based on tcpdump, we implement a traffic collection tool to automatically capture traffic fingerprinting between all VM pairs in every time interval during the whole running process of applications.

In every time interval (e.g., second or minute), we sum up a total bytes of all the transferred packets between VM pairs as traffic volume. Specifically, each traffic record signifies a network packet transferred from a source server to a destination server. We express it as a triple like $<$ sourceid, destinationid, volume $>$, in which sourceid and destinationid denote the source and distinction server IP address, respectively, and the volume denotes the packet size (bytes) of traffic being captured within a length of monitoring time. In addition, a pair of VMs (x, y) will have distinct value of traffic volumes in the two transmission directions. For simplicity, we assume they are identical otherwise use average of them. As a metric, we use *average traffic rate* (ATR): bps,mbps (bytes/mbytes per second) or pps (packet-amounts per second) to denote the communication traffic dependency between a VM pair. In addition, we conclude the metric of ATR between a VM pair,

by dividing the total traffic volume by the total runtime like total seconds consumed in running applications as in the following formula.

$$ATR = Total\ traffic\ volume/Total\ time(sec.). \quad (9)$$

VMs with traffics indicates the VMs be colocation placed onto the same PM and we identify the VMs with *colocation placement affinity*. For each VM pair with affinity we map the metric ATR as affinity degree.

Besides, when a set of VMs runs CPU-/memory- intensive applications in one PM, we can capture the PM resource usage including CPU, RAM, disk I/O and network, and check whether there are heavy resource contention or not between VMs. And then we evaluate the application performance by comparing performance results under distinct VM placement patterns. If there are heavy resource contention between VMs, then we identify no-colocation placement affinity for the VMs.

5.3. Joint affinity aware grouping and placement

We adopt the approximation methods to solve the AAP problem because the AAP problem is NP-hard. We propose a joint affinity aware grouping and placement method. In the method, the AAP problem is divided into two subproblems, (1) affinity aware grouping problem; (2) affinity-VM-group placement problem.

5.3.1. Affinity aware grouping

The affinity aware grouping problem is generated from the co-affinity equivalence class partition features derived from the colocation affinity equivalence relation. Specifically, note that VMs with colocation affinity relationships are required to be colocation placed, so two VMs with colocation affinity are required to be colocation placed onto one PM, which indicates the two VMs can be combined together as one big VM unit to be placed onto one PM. Further, in Section 4, we have proved that the colocation affinity relation is an equivalence relation. Hence, given a number of VMs with colocation affinity relations, we can do equivalence partition, i.e., grouping for these VMs. We call it as affinity aware grouping, which targets to partition the VMs with colocation affinity relations into a set of disjoint VM sets. That is, the VMs with colocation affinity are firstly partitioned into a set of disjoint equivalence classes. Each equivalence class includes a set of VMs, which constructs a VM group.

We introduce and define *colocation affinity VM group* as follows.

Definition 6. Colocation affinity VM Group. The colocation affinity VM group is defined as a set of VMs in an equivalence class based on colocation affinity relation. In VM placement, all VMs of this group are viewed as a whole unit required to be colocation placed onto one PM.

We create rules to generate colocation affinity VM groups as follows according to the relation theory and co-affinity property.

Given a finite set V of VMs, let $x, y \in V$ be two VMs.

Obviously, a set of one VM forms a colocation affinity VM group because a VM item and itself will be obviously allocated to the same PM and have colocation affinity relationship.

Claim 1. *A set of two VMs having a co-affinity relationship forms one affinity VM group.*

Proof. Given two VMs x and y, $x \sim y$, $\mathcal{AG}$ is the union of set $\{x\}$ and $\{y\}$, i.e., $\mathcal{AG} = \{x\} \cup \{y\} = \{x, y\}$, a subset of V, generates a co-affinity group on x and y. Due to x and y having co-affinity relation, x and y are allocated to the same PM, namely, all VMs in $\mathcal{AG}$ are allocated to one PM. □

Claim 2. *The union of two co-affinity VM groups forms an co-affinity VM group, if two VMs between the two different groups exist one co-affinity relation. Equivalently, let $\mathcal{AG}_1$ and $\mathcal{AG}_2$ be two affinity VM groups, and $\mathcal{AG}_1 \cap \mathcal{AG}_2 = \varnothing$, if $\exists x \in \mathcal{AG}_1, y \in \mathcal{AG}_2$, and $x \sim y$, then the union of $\mathcal{AG} = \mathcal{AG}_1 \cup \mathcal{AG}_2$ is a union affinity VM group for two disjoint affinity VM groups.*

Proof by contradiction. if $\mathcal{AG}_1$ and $\mathcal{AG}_2$ cannot be allocated to the same PM, then x and y cannot be allocated to the same PM, which means x and y are not grouped into the same group, or the affinity between x and y is broken. So if and only if $\mathcal{AG}_1$ and $\mathcal{AG}_2$ are allocated to the same PM, or distinct PMs, the affinity-aware VM placement between x and y can be guaranteed. □

5.3.2. Affinity VM group placement

After affinity aware grouping, the VMs are grouped into a set of co-affinity VM groups which are uniformly processed in VM placement.

We consider three cases as follows:

(1) The first case is called *only-co-affinity placement*, denoted by *COAP*, in which all the affinity of VMs are co-affinity and there are no large co-affinity groups. We do not consider noco-affinity, then the VMs can be grouped of a set of co-affinity VM groups, and each co-affinity VM group can be combined and viewed as one big VM. The VM placement turns into a group placement problem.
(2) The second case is called *co- and noco-affinity placement*, denoted by *CN-COAP*, in which some VMs have co-affinity and some VMs have noco-affinity. In this case, we firstly partition VMs into groups according to co-affinity relations and then make decision of allocation the VM groups into PMs with consideration of the noco-affinity one by one.
(3) The third case is called *big co-affinity placement* denoted by *BIG-COAP* in which the total resource demand size of all VMs in one *co-affinity group is bigger than all the PM resource capacity*. In this case, we cannot allocate these co-affinity group VMs into one PM, so we allow the co-affinity group VMs to be dispersedly placed onto distinct PMs. But yet, some co-affinity relations are not satisfied, which generates a special system overhead.

5.4. Scheduling algorithms

In this section, we propose the joint affinity aware grouping and placement method with affinity aware resource scheduling algorithms to solve the AAP problem. It includes a grouping algorithm for VMs with co-affinity relations and bin packing heuristics algorithms for resource scheduling with consideration of all other affinity relations except co-affinity. The scheduling includes many types, such as the initial placement and load balancing. We focus on solving the initial VM placement problem.

5.4.1. Affinity aware grouping

Given a set V of VMs and a set $\mathcal{AR}$ of VM-affinity relation between VMs, the affinity aware grouping algorithm aims to group the VMs into a set of disjoint subsets of V, each of which is a *maximized co-affinity VM group* and in each group none of VM pairs has noco-affinity relationships. The *maximized co-affinity VM group* is such a group inside which any VM has co-affinity relation with at least one VM in the group and has no co-affinity relation with any other VM outside the group.

We accordingly have the following two assumptions, namely (1) an optimal grouping enables each VM group to have the maximal number of VM items with co-affinity and each VM has co-affinity relationship with at least one VM in the same group and has no co-affinity relationship with any VM item outside this

group; (2) for each dimension the total resource demand of any co-affinity VM group is no more than the resource capacity of PM; (3) two VMs with noco-affinity must be allocated onto two distinct co-affinity groups, namely, the affinity groups are also identified with noco-affinity relations.

We denote this co-affinity grouping algorithm as Max-VA-Grouping (MVAG) algorithm and describe the detailed steps as shown in Algorithm 1.

Algorithm 1 Algorithm MVAG : Max-VA-Grouping.

Require:
A VM set with the number N;
A VM affinity relation set VR with the number E;
Ensure:
A set of disjoint maximized co-affinity VM groups VG and a set of noco-affinity relation between co-affinity groups GR.
1: Input the resource demand value of VMs V, and affinity relations VR.
2: Build-set. Initially each VM of V is built as a co-affinity VM group set (singleton set) VG and a new noco-affinity relation set for co-affinity groups GR is set empty.
3: Find-set. For each affinity relation between a VM pair in VR, find the co-affinity VM group set which contains the two VMs, and get the two co-affinity VM group sets.
4: Union-set. If the affinity relation is co-affinity, then we apply union operation to the two co-affinity VM group sets to get a new co-affinity VM group set; otherwise, we have the affinity is noco-affinity, then we create and identify a new noco-affinity relation between the two co-affinity groups, namely, add the noco-affinity between the two co-affinity groups to GR.
5: If there are any other co-affinity relations, then goto Step 3; otherwise goto Step 6.
6: **return** VG, the result co-affinity VM group set and GR, a noco-affinity relation set for VG.

The grouping can reduce the size scale of the number of VM items in resource allocation and improve the efficiency of VM placement decision. Specifically, let NV be the number of VM items and NG be the number of group items, then we have $NG \leq NV$. The time complexity consists of the time for three operations, namely build-set (Step 2), find-set (Step 3), and union-set (Step 4) operations. We use a tree-based data structure and a path compression technique to minimize the computation complexity. Being similar as the disjoint set algorithm [37], the time complexity of MVAG algorithm is O(N+E), where N is the number of VMs and E is the number of affinity relations.

5.4.2. Affinity-VM-group packing

After affinity aware grouping, we have a set of co-affinity VM groups and a set of noco-affinity relationships between these co-affinity VM groups. Next, we firstly assume the resource size of all affinity groups is smaller than the PM resource capacity. And then, we consider an optimal resource allocation scheme to allocate these co-affinity groups onto PMs and minimize the number of PM with satisfying two constraints, namely, resource capacity limit of PM and affinity. Given that the co-affinity group can be viewed as a whole unit like one big VM, the co-affinity group allocation is similar as the single VM resource allocation and the affinity group allocation problem is NP-hard [16]. Thus, we also use bin packing methods to conclude the optimal solution. We call the VM group bin packing as affinity group packing (AGP), which is also a type of vector bin packing (VBP) [29], or multi-dimensional VM packing. In particular, during the whole process of decision making of group allocation, all affinity relations are required to be satisfied, namely, the VMs in the same group be allocated onto the same PM, two VMs with noco-affinity be disperse placed onto distinct PMs.

In the following, we depict some affinity group packing algorithms based upon bin packing heuristic methods. The main process of affinity group packing algorithm includes the following four steps.

Step 1. Input. Initially, input affinity VM group items, PM bins and no-colocation affinity relations between groups;
Step 2. Sorting groups. Use a measure to identify a value for each group item to decompose resource vectors and make sort by the measure in a decreasing order;
Step 3. Packing groups. Make decision of selecting and packing each group items into a suitable PM according to a policy of satisfying two constraints or rules: (a) resource capacity limits constraint; (b) the affinity constraints including colocation affinity and no-colocation affinity.
Step 4. Output. Output the group items placement results, a list of mapping VMs to PMs.

From the above basic process, if we do not sort the group items in a decreasing order, we have the heuristics methods, the First Fit (FF), Best Fit (BF), Next Fit (NF), etc. If we employ the sort in placement we have the heuristics including the First-Fit-Decreasing (FFD), the Best-Fit-Decreasing (BFD). We also have many variants of FFD for multi-dimensional items, such as FFD-Prod, FFDAvgSum etc. if we convert the resource vector items to a numerical value for sorting in a decreasing order. In particular, 8 variants of FFD and BFD based on eight kinds of measures are proposed in [23] and three variants of FFD, namely, FFDProd, FFDAvgSum and FFDExpSum are provided in [29].

Without loss of generality, we choose the FFD bin packing policy to make decision of the affinity group packing, and then we have a FFD-based group packing. The algorithm is denoted as FFD-AGP algorithm. We analyze the performance of the algorithm. The time complexity of FFD-AGP includes the sorting and packing operation. The sorting complexity is at least $O(K\log(K))$ because we can use quick-sort method, where K is the number of group items. The packing includes iterations of making decisions of packing K group items and testing E times of no-colocation affinity relations, so the complexity of packing is $O(K+E)$. Thus, the total complexity of FFD-AGP is $O(K\log(K))+O(KE)$. Especially, the max of E is $K(K-1)/2$ if all pairs of group items have a no-colocation affinity, which is actually impossible.

5.4.3. Discussion

We have mentioned three cases of affinity VM group placement, namely, COAP, CN-COAP and BIG-COAP. The first two cases can be solved by the above-mentioned affinity group packing algorithms. For the BIG-COAP scenario, the big affinity VM group is split into two or more groups, each of which can be packed into a PM. The split operation will generate a special system overhead. To minimize the overhead derived from the unsatisfied affinity, the best placement scheme is that the VMs in each colocation affinity VM group are allocated onto the same PM, and the PM have so large size of resource capacity that can pack all VMs in this VM group.

We can assume that for each VM affinity group, there always exists one PM that has enough resource capacity to pack VMs in this affinity group. This assumption may be reasonable because now the increasingly growing hardware technology brings one PM with increasing size of resource capacity, such as CPU and RAM. We can use a bigger location, such as rack, which denotes a cluster of several PMs deployed onto one or more racks connected with high bandwidth. We also assume that for each service most tenants use a small number of VMs across which having affinity leading to a small affinity group. Only a few services require a large

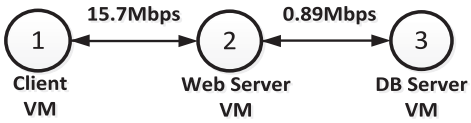


Fig. 7. Traffic rate amongst 3 VMs (1–3) running RUBiS benchmark. The values (Mbps) are average traffic ratio (ATR).

number of VMs with affinity as one large affinity group with total resource requests overstepping the PM resource capacity.

Actually, in the BIG-COAP scenario, there are no PMs that can take in all VMs in one large affinity group. We also can use a mini-cut method [25] to divide the large affinity VM group item into several small affinity VM group items, across the items having minimized overhead, such that each group resource request is less than one PM capacity and the network communication overhead is minimized.

6. Evaluation

We implement a tool based on tcpdump to automatically obatin traffics between VMs running communication-/data- intensive applications for testing the existence of affinity between VMs. Then we implement a simulation software tool for affinity resource scheduling with grouping and bin packing heuristic algorithms in C language. We create a real cloud environment to evaluate the effectiveness of the JAGBP method. The configuration of experimental environment is the same as described in our case study in Section 3.

6.1. Evaluate affinity from traffics

In Section 3, we provide a case study of obtaining traffic between VMs on running the HPCC benchmark application. In this section, we present a further experiment to evaluate the existence of affinity to generalize the method. We choose other two typical benchmark applications, namely, RUBiS and hadoop [47] for affinity evaluation. RUBiS is a multi-tier emulation of e-bay web application. It simulates various tasks done by various clients, including user and item registration, browsing items per category and per region, bidding for or buying items and so on. Hadoop provides a distributed file system by using the MapReduce paradigm to analyze and transform very large data sets. It tackles computation from files distributed amongst multiplicative nodes.

6.1.1. Traffics on running RUBiS

In this section, we evaluate the traffics between the VMs of RUBiS. RUBiS contains three modules: a web server, a database server and an emulation client, which are setup onto three VMs under one PM, respectively. In traffic measurement, the number of clients, a primary parameter determining the workload size, is set to 1400, and other parameters are set as default. We run the experiment five times and each run lasts twenty minutes. We get the average of total traffic volumes and conclude the traffic rate of all VM pairs. The final results are shown in Fig. 7. We observe that the traffic rate between different VM pairs is different, i.e., the traffic rate between client and web server VM is 16.6 times higher than that of between web and DB server VM. Hence, the web server and database server have frequent communications because all web requests from clients are processed by the web server and database server.

6.1.2. Traffics on running hadoop

In this section, we evaluate the traffic between VMs running Hadoop application. We construct a virtual cluster (VC) with 16 VMs, which are placed onto one PM and all VM image files are uniformly stored in a network file system (NFS) server. We pick one typical workload of hadoop: wordcount, which is used to count all words inside a certain word file under a distributed multi-nodes platform. The size of word file is set to 100MB. We capture traffic fingerprinting between all VM pairs and conclude the average traffic rate as affinity degree as shown in Fig. 8.

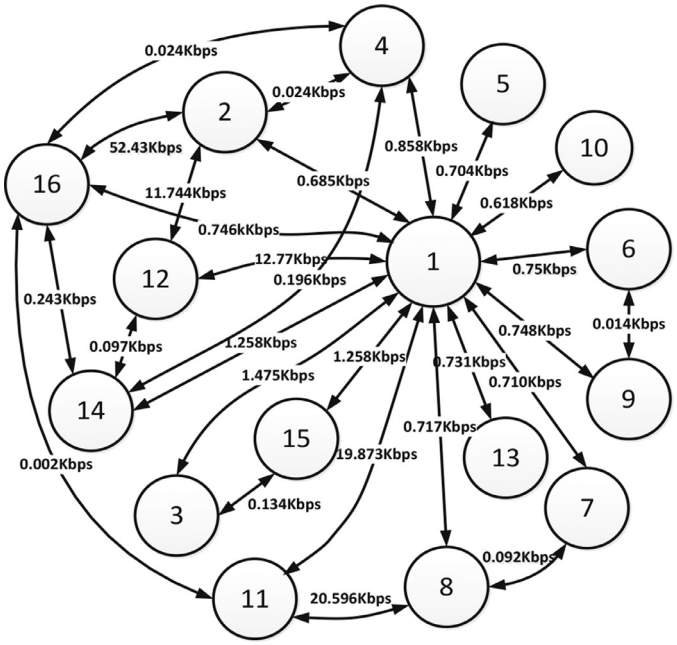


Fig. 8. Traffic amongst 16 VMs (1–16) running a Hadoop workload: wordcount. The values (Mbps) are average traffic ratio (ATR).

Fig. 8 shows that all VM pairs have small average traffic rates. The main control node is VM node 1 which has the most traffic rates. We can conclude that the traffics between hadoop VMs are less frequent.

6.1.3. Result analysis

By comparison of all traffic measurement results from the three typical cloud applications, HPCC performs the largest traffic rates, which indicates the largest dependency between the VMs. While Hadoop performs as the opposite, the traffic rates between VM pairs are very small and much less than that from RUBiS. The large amount of traffic rate indicates that the VMs hosting HPCC benchmark application require much more network bandwidth than others. In conclusion, these results tell us that not only the traffic dependency between VM pairs commonly exist amongst the VMs running cloud applications, but also it reveals a great difference among distinct cloud applications running between multiple VMs.

6.2. Evaluate algorithms in simulation

In this section, we provide a simulation experiment to verify the effectiveness of algorithms used in JAGBP method. Initially, all the algorithms, including the affinity grouping and bin packing heuristics algorithms, are implemented in a simulator tool which is used as a resource scheduler for our affinity aware resource scheduling framework. The simulator program runs on a Dell PowerEdge T710 Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz machine with 16GB of memory running Linux OS.

We generate several classes of VM workloads according to distinct scale in VM number, dimension, and affinity relation. We use a metric affinity ratio to denote the number scale of affinity relations, which is concluded by dividing the number of affinity

relationships to the number of VMs *N*. each class contains 5 different scales of VM number (N is set to $2000, 4000, 6000, 8000, 10,000$), 6 dimensions ($d$ is set from 1 to 6), and 6 kinds of affinity relation scale, which is denoted by *affinity ratio* (AR): $r = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,$ respectively.

Firstly, we randomly generate totally 8 VM workload files corresponding to 8 classes of distribution workloads. Each file denotes a class of VM workload information. To facilitate running tests, we generate two arrays with 10,000 rows and 6 columns, totally $60,000$ real numerical VM workload data in each file. However, the production of the affinity relation workloads data is directly in correlation with the VM workload and also generated random. It varies in different classes of workload distribution, the number of VM, dimension and affinity relation. Then, totally $5 \times 6 \times 6 \times 8 = 1440$ affinity workload files are produced. One simulation instance includes one VM workload file and one affinity file.

We conduct simulations for two scenarios, (1)VM packing (VMP) method, a bin packing method without considering affinity, i.e., a *non-affinity aware grouping allocation* method (NAGA) for the GP problem, and (2) JAGBP method, for our Joint-Affinity Grouping and Bin Packing method, to test and compare performance based on the runtime metric. In each round of simulation execution, the simulation tool firstly loads a VM workload file with a set of VMs and an affinity relation file. Then it invokes the Max-AV-Grouping algorithm and outputs the affinity group results. These results are a set of VM affinity groups and used as input for the bin packing algorithms. At last it implements the heuristic bin packing algorithms and outputs the placement scheme results. During the period of each round of execution, the runtime of each algorithm and the packing efficiency of relative metrics including PM number for each heuristic packing algorithm, are logged in a result file. In JAGBP, the total runtime is the sum of *Affinity Grouping* time, denoted by AGT and affinity group packing time, denoted by GPT. While in NAGA method the runtime is just only total of VM packing time. Besides, for the correctness of the result, the overall execution runs three times and the results are accumulated into several files. We generate total $1440 \times 3 \times 14 = 60,480$ amounts of data used for performance analysis.

At last, we choose FFD packing algorithm based on a descending order by total sum of all dimensions of each item and the runtime metric to show the efficiency of JAGBP with comparison to NAGA method. The results are shown in Fig. 9 which includes four subfigures. Fig. 9(a) is the total runtime results of JAGBP and NAGA amongst all workload classes in the case of 2 dimensions, 4000 VMs, and affinity ratio = 0.4; Fig. 9(b) is the total runtime results of JAGBP and NAGA amongst all distinct VM number in the case of 4 dimensions, workload class C4, and affinity ratio = 0.4; Fig. 9(c) is the total runtime results of JAGBP and NAGA amongst all distinct affinity ratios in the case of 4 dimensions, workload class C4, and 6000 VMs; and Fig. 9(d) is the total runtime results of JAGBP and NAGA amongst all distinct dimensions in the case of workload class C1, 6000 VMs, and affinity ratio = 0.5.

From the Fig. 9, in all cases the total runtime of JAGBP is less than that of NAGA method, resulting that the VM affinity grouping and packing achieve higher efficiency than the single VM packing in terms of less runtime. Actually, in a large resource scheduling scenario, the scheduling decision efficiency is important. Thus the runtime efficiency of JAGBP method will be meaningful for practical applications.

6.3. Evaluate performance of the JAGBP method

In this section, we create a real cloud application running experiment in which we combine the simulator algorithms and VM placement as a whole solution. Given many VMs and PMs, we present two schemes based on two VM deployment schemes corresponding to JAGBP and NAGA method, respectively. In each scheme we run the same cloud application running on multiple VMs and evaluate the efficiency of the JAGBP method by comparing the application performance results.

Table 2
The configuration of each virtual cluster.

VC	VM number	RAM/VM (MB)	CPU	Total RAM(GB)
VC1	16	512	shared	8
VC2	8	768	shared	6
VC3	6	640	shared	3.75
VC4	12	384	shared	4.5
VC5	10	512	shared	5
VC6	4	896	shared	3.5
VC7	3	1024	shared	3

6.3.1. Experimental environment

A cloud datacenter generally runs many kinds of applications hosted amongst multiple VMs, and we denote a group of VMs for running one type of application as a virtual cluster (VC). Hence a cloud datacenter comprises several VCs. Different VCs run different or same applications with distinct preset input workloads or data scales. We focus on the case that one application is hosted with many VMs. For simplicity, we choose HPCC and RUBiS application for our evaluation.

To construct an experimental environment imitating a real cloud datacenter, we chose four identical Dell PowerEdge T710 server machines with dual Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz, each of which has totally 16 Cores and 16GB RAM running Xen-3.3.1. The Linux VMs and domain0 run Linux Kernel-2.6.18.8-xen. The domain0 is configured with many virtual CPU cores. All VM images are stored in a Network Filesystem Server (NFS). The VMs in different VC share CPU cores, and are allocated with different size of memory.

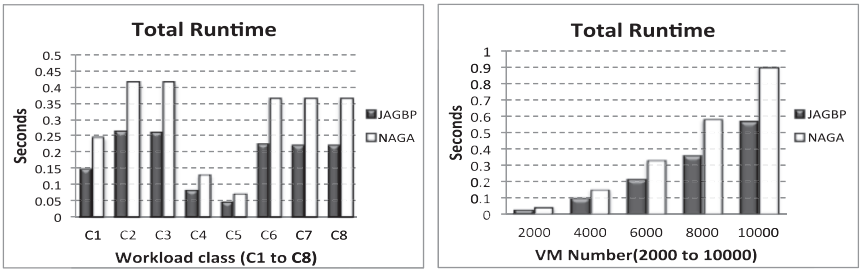
To simulate multiple applications, we totally construct 7 distinct VCs with total 59 VMs as computing nodes by given different preset configurations. 6 VCs (VC1-VC6) are generated by running HPCC benchmarks according to different number of nodes and application scale parameters. The VC7 is used for running RUBiS application with three VMs and each VM is allocated with 1 shared vCPU and 2GB memory. Each VC is made up of different number of VMs with different total vCPU and memory. Because the running of HPCC and RUBiS generate traffics between VMs, the VMs of each VC are identified with colocation placement affinity. Thus each VC is made up of a colocation affinity VM group.

The configuration of $VC1 - -VC7$ is listed in Table 2. The VMs in one PM share the CPU limited to 32 Cores, and the RAM capacity is limited to 12GB (total 16GB) with consideration of the resource reservation [45]. As shown in Table 2, we limit each affinity-group with a total resource (RAM) demands less than the PM capacity, such that it can be colocation placed onto one PM.

6.3.2. Placement schemes

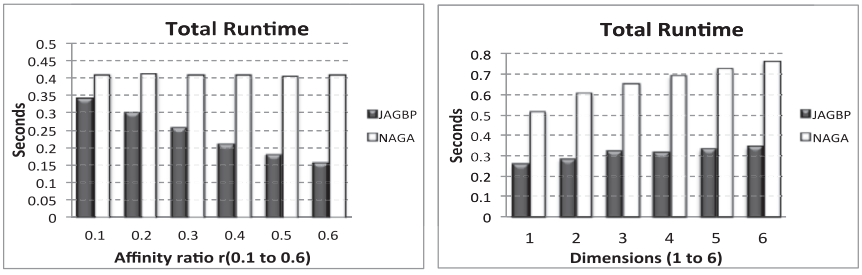
We implement two specific placement schemes. In each scheme the placement pattern is determined by a certain placement strategy. The placement solution is concluded by a provided allocation strategy, denoted as a table, in which the row represents VC, the column represents PM and a numerical value in a cell (i, j) denotes the VM number of VC_i allocated to PM_j. One VC runs a multi-VM application and the VMs construct one colocation affinity group with colocation placement affinity relationships.

The first scheme adopts the JAGBP method. The VMs are firstly identified with affinity relationships and grouped into several colocation affinity groups via Max-VA-Grouping. Then, we arbitrarily



(a) Performance comparison in workload class,$d = 2, VMs = 4000, AR = 0.4$

(b) Performance comparison in different VM number,$d = 4, Class = C4, AR = 0.4$



(c) Performance comparison in affinity ratio(AR),$d = 4, Class = C4, VMs = 6000$

(d) Performance comparison in dimension,$Class = C1, VMs = 6000, AR = 0.5$

Fig. 9. The performance comparison from JAGBP to NAGA method against total runtime.

Table 3
FFD-based affinity-VM-group placement using JAGBP.

VC/PM	PM1	PM2	PM3	PM4
VC1(HPCC)	16	0	0	0
VC2(HPCC)	0	8	0	0
VC3(HPCC)	0	0	6	0
VC4(HPCC)	0	0	12	0
VC5(HPCC)	0	10	0	0
VC6(HPCC)	0	0	4	0
VC7(RUBiS)	0	0	0	3

Table 4
FFD-based VM placement.

VC/PM	PM1	PM2	PM3	PM4
VC1(HPCC)	0	16	0	0
VC2(HPCC)	8	0	0	0
VC3(HPCC)	4	2	0	0
VC4(HPCC)	0	0	12	0
VC5(HPCC)	0	5	5	0
VC6(HPCC)	4	0	0	0
VC7(RUBiS)	1	0	1	1

use an FFD bin packing heuristic method to allocate these colocation affinity groups onto PM. In FFD bin packing the VMs are sorted by the VM memory resource dimension in a decreasing order. The detailed deployment is listed in Table 3.

The other scheme employs the *VM packing (VMP)* method without considering affinity of VMs, namely, *non-affinity grouping allocation* method, which is denoted as **NAGA**. We use FFD bin packing heuristic method in which VMs are firstly sorted in a decreasing order based on memory dimension, and then deployed onto PMs one by one. For VC7 we deploy the 3 VMs dispersedly onto three PMs to run RUBiS with a workload size 2000 (client number) in order to do a comparison with JAGBP method. Table 4 lists the detail of deployment information. Except VC7, the VMs of VC3 and VC5 are deployed amongst two PMs, i.e., VC3 as $(4+2+0+0)$ and VC5 as $(0+5+5+0)$, while the VMs of other VCs are still colocation placed onto one PM, e.g., VC1 as $(0+16+0+0)$.

6.3.3. Running and results analysis

We firstly run the HPCC applications over the VC1-VC6 and the RUBiS application of VC7 under both JAGBP scheme and NAGA scheme respectively. All schemes run concurrently three times and generate several files each time. And then we extract from the result files and pick four metrics from four communication-intensive benchmarks of HPCC, i.e., HPL_Tflops for HPL, PTRANS_GBs for PTRANS, AvgPingPongBandwidth_GBytes for PingPong benchmark of b_eff, MPIFFT_Gflops for FFT, and choose the metric average throughput from the RUBiS application result. Table 5 gives the average result for each metric value.

From Table 5, we can observe that all performance metrics of VC3 and VC5 generated from JAGBP scheme are better than those generated from NAGA scheme. Especially, the performance of MPIFFT benchmark in VC5 derived from JAGBP scheme performs 23.5 times better than that from NAGA scheme, while VC3 performs 5 times better. For other VCs, JAGBP scheme performs better performance results than NAGA scheme. However, a few cases do not show expected performance, such as MPIFFT in VC6, AvgPingPong in VC2 and VC6, VC6 under JAGBP and NAGA scheme, because of the resource contentions derived from VMs sharing CPU and memory bring impact on it.

In addition, in our experiments, due to VC1-VC6 run the same HPCC benchmark, we sum the metric value to compare the overall performance of the simulated cloud multi-VCs system under JAGBP and NAGA deployment schemes. All metric value of VCs under JAGBP and NAGA deployment schemes are normalized to 1 for performance comparison. Fig. 10 shows the overall performance results. We conclude that JAGBP outperforms NAGA the HPCC application performance metrics, i.e., HPL improves 28.3%, PTRANS 16.3%, MPIFFT 87.6%, and AvgPingPong 19.7%, respectively, and Average Throughput metric of RUBiS benchmark application improves 25.2%. These results demonstrate the effectiveness of the JAGBP method.

7. Conclusion and future work

In this paper, we have studied the affinity aware VM placement problem. The contribution we made is as follows. (1) We

Table 5
The metric results of benchmarks in all VCs under JAGBP and NAGA scheme.

Benchmark (Metric)	Deploy. Scheme	VC1	VC2	VC3	VC4	VC5	VC6	VC7	Total
HPL(Tflop/s)	JAGBP	0.0087	0.0094	**0.0026**	0.0046	**0.0045**	0.0060		0.0358
	NAGA	0.0078	0.0059	**0.0010**	0.0054	**0.0017**	0.0061		0.0279
PTRANS(GB/s)	JAGBP	0.3648	0.3711	**0.0497**	0.1621	**0.0594**	0.4659		1.4731
	NAGA	0.3959	0.2454	**0.0100**	0.2793	**0.0135**	0.3662		1.3103
MPIFFT(Gflop/s)	JAGBP	0.5606	0.4468	**0.3198**	0.5565	**0.5859**	0.1478		2.6174
	NAGA	0.3944	0.4014	**0.0623**	0.5363	**0.0249**	0.3226		1.7419
AvgPingPong(Gbyte/s)	JAGBP	0.3208	0.1679	**0.2673**	0.2931	**0.3826**	0.1316		1.5634
	NAGA	0.2166	0.2682	**0.1763**	0.2999	**0.1048**	0.2399		1.3056
AverageThroughput(req/s)	JAGBP							273	273
	NAGA							218	218

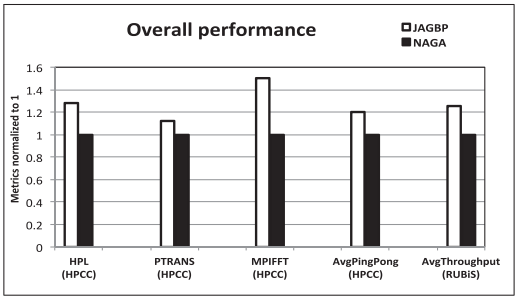


Fig. 10. The overall system performance for each benchmark under JAGBP and NAGA method.

have made a case study of evaluating application performance under distinct VM placement patterns and the results give us our motivation. (2) We have introduced affinity of VMs, affinity relations between VMs, associated the VM placement patterns to minimize the application performance reduction, and identified the affinity relationships across VMs from real application cases. (3) We have proposed the JAGBP method to solve the AAP problem. The JAGBP method includes an affinity grouping algorithm to group the VMs given that there are known affinity relationships across VMs and bin packing heuristic algorithms. The algorithms are integrated into an affinity aware resource scheduling framework for cloud system. (4) We have conducted comprehensive experiments to demonstrate the effectiveness and efficiency of the JAGBP method.

For our next study, a closely related direction is to investigate many other techniques to find, quantify and identify the affinity relationships from a list of VMs running practical cloud computing applications. Another direction is to study how to use the affinity degree value to advance the VM placement based resource allocation and handle the case when the capacity of a PM is less than the resource demand of an co-affinity VM group.

Acknowledgments

The authors thank anonymous referees for helpful comments and suggestions to improve the presentation of this paper. This research is supported partly by the National Science and technology support program of China under Grant (No. 2012BAH94F00), and NSF of China under grant (No. 61472359,11671355).

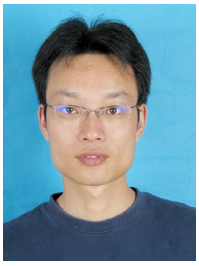
References

[1] J. Y. Arrasjid, B. Lin, R. Veeramraju, S. Kaplan, D. Epping, M. Haines, Cloud computing with vmware vcloud director(2011).
[2] M.D. Assunção, R.N. Calheiros, S. Bianchi, M.A. Netto, R. Buyya, Big data computing and clouds: trends and future directions, J. Parallel Distrib. Comput. 79 (2015) 3–15.
[3] D. Carrera, M. Steinder, I. Whalley, J. Torres, E. Ayguadé, Utility-based placement of dynamic web applications with fairness goals, in: Proceedings of 2008 IEEE Network Operations and Management Symposium (NOMS 2008), IEEE, 2008, pp. 9–16.
[4] H. Chen, H. Jin, K. Hu, Affinity-aware proportional share scheduling for virtual machine system, in: Proceedings of the 9th International Conference on Grid and Cooperative Computing (GCC 2010), Nanjing, China, 2010, pp. 75–80.
[5] A. Corradi, M. Fanelli, L. Foschini, Vm consolidation: a real case based on openstack cloud, Future Gener. Comput. Syst. 32 (2014) 118–127.
[6] C. Delimitrou, C. Kozyrakis, Quasar: resource-efficient and qos-aware cluster management, ACM SIGPLAN Not. 49 (4) (2014) 127–144.
[7] W. Depoorter, K. Vanmechelen, J. Broeckhove, Advance reservation, co-allocation and pricing of network and computational resources in grids, Future Gener. Comput. Syst. 41 (2014) 1–15.
[8] M. García-Valls, T. Cucinotta, C. Lu, Challenges in real-time virtualization and predictable cloud computing, J. Syst. Archit. 60 (9) (2014) 726–740.
[9] S.K. Garg, A.N. Toosi, S.K. Gopalaiyengar, R. Buyya, Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter, J. Netw. Comput. Appl. 45 (2014) 108–120.
[10] G. Gonçalves, P. Endo, T. Damasceno, A. Cordeiro, D. Sadok, J. Kelner, B. Melander, J. Mångs, Resource allocation in clouds: concepts, tools and research challenges, XXIX SBRC-Gramado-RS (2011).
[11] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Vl2: a scalable and flexible data center network, in: ACM SIGCOMM Computer Communication Review, 39, ACM, 2009, pp. 51–62.
[12] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, X. Zhu, Vmware distributed resource management: design, implementation, and lessons learned, VMware Tech. J. 1 (1) (2012) 45–64.
[13] I. Gurobi Optimization, Gurobi optimizer reference manual2014, URL http://www.gurobi.com(2014).
[14] C.-H. Hong, Y.-P. Kim, H. Park, C. Yoo, Synchronization support for parallel applications in virtualized clouds, J. Supercomput. 72 (9) (2016) 3348–3365.
[15] V. Ishakian, A. Bestavros, Morphosys: Efficient Colocation of qos-Constrained Workloads in the Cloud, Tech. Rep. BUCS-TR-2011-002, CS Dept., Boston University, 2011.
[16] D. Johnson, M. Garey, Computers and Intractability: A Guide to the Theory of np-Completeness, Freeman&Co, San Francisco, 1979.
[17] C. Krintz, The appscale cloud platform: enabling portable, scalable web application deployment, IEEE Internet Comput. 17 (2) (2013) 72–75.
[18] R. Kumar, K. Jain, H. Maharwal, N. Jain, A. Dadhich, Apache cloudstack: open source infrastructure as a service cloud computing platform, Int. J.Adv.Eng.Technol. Manage. Appl. Sci. 1 (2) (2014) 111–116.
[19] X. Li, H. Wang, B. Ding, X. Li, D. Feng, Resource allocation with multi-factor node ranking in data center networks, Future Gener. Comput. Syst. 32 (2014) 1–12.
[20] Y. Li, X. Tang, W. Cai, Dynamic bin packing for on-demand cloud resource allocation, Parallel Distrib. Syst. IEEE Trans. 27 (1) (2016) 157–170.
[21] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, D. Takahashi, The hpc challenge (hpcc) benchmark suite, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC 2006), Citeseer, 2006, pp. 11–17.
[22] P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, D. Takahashi, Introduction to the hpc challenge benchmark suite(2005).
[23] K. Maruyama, S. Chang, D. Tang, A general packing algorithm for multidimensional resource requirements, Int. J. Parallel Program. 6 (2) (1977) 131–149.
[24] S. McCanne, C. Leres, V. Jacobson, Tcpdump and libpcap, 2012.
[25] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of IEEE INFOCOM (INFOCOM 2010), 2010, pp. 1–9.
[26] B. Narasimhan, R. Nichols, State of cloud applications and platforms: the cloud adopters' view, Computer 44 (3) (2011) 24–28.
[27] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in: ACM SIGCOMM Computer Communication Review, 39, ACM, 2009, pp. 39–50.
[28] O. Ore, Theory of equivalence relations, Duke Math. J. 9 (3) (1942) 573–627.
[29] R. Panigrahy, K. Talwar, L. Uyeda, U. Wieder, Heuristics for vector bin packing, research. microsoft. com (2011).

[30] M. Sindelar, R. Sitaraman, P. Shenoy, Sharing-aware algorithms for virtual machine colocation, in: Proceedings of the 23th ACM Symposium on Parallelism in Algorithms and Architectures. San Jose, California, USA, (SPAA 2011), 2011.
[31] J. Smith, A.A. Maciejewski, H.J. Siegel, Maximizing stochastic robustness of static resource allocations in a periodic sensor driven cluster, Future Gener. Comput. Syst. 33 (2014) 1–10.
[32] J. Sonnek, A. Chandra, Virtual putty: Reshaping the physical footprint of virtual machines, in: Proceedings of Workshop on Hot Topics in Cloud Computing (HotCloud 2009), 2009.
[33] J. Sonnek, J. Greensky, R. Reutiman, A. Chandra, Starling: minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration, in: Proceedings of the 39th IEEE International Conference on Parallel Processing (ICPP 2010), IEEE, 2010, pp. 228–237.
[34] M. Stillwell, D. Schanzenbach, F. Vivien, H. Casanova, Resource allocation algorithms for virtualized service hosting platforms, J. Parallel Distrib. Comput. 70 (9) (2010) 962–974.
[35] S. Sudevalayam, P. Kulkarni, Affinity-aware modeling of cpu usage for provisioning virtualized applications, in: Proceedings of the IEEE International Conference on Cloud Computing (CLOUD 2011), IEEE, 2011, pp. 139–146.
[36] S. Sudevalayam, P. Kulkarni, Affinity-aware modeling of cpu usage with communicating virtual machines, J. Syst. Softw. 86 (10) (2013) 2627–2638.
[37] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press and McGraw-Hill, 2001.
[38] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Gener. Comput. Syst. 28 (2) (2012) 358–367.
[39] N. Vasic, D. Novakovic, S. Miucin, D. Kostic, R. Bianchini, Dejavu: accelerating resource allocation in virtualized environments, in: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012), 12, 2012.
[40] D. Wilcox, A. McNabb, K. Seppi, Solving virtual machine packing with a reordering grouping genetic algorithm, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2011)„ IEEE, 2011, pp. 362–369.
[41] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Sandpiper: black-box and gray-box resource management for virtual machines, Comput. Netw. 53 (17) (2009) 2923–2938.
[42] J. Xu, J. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proceedings of the IEEE/ACM Int'l Conference on & Int'l Conference on Green Computing and Communications (GreenCom 2010), 2010, pp. 179–188.
[43] C. Yan, M. Zhu, X. Yang, Z. Yu, M. Li, Y. Shi, X. Li, Affinity-aware virtual cluster optimization for mapreduce applications, in: Proceedings of 2012 IEEE International Conference on Cluster Computing (CLUSTER 2012), IEEE, 2012, pp. 63–71.
[44] D. Ye, J. Chen, Non-cooperative games on multidimensional resource allocation, Future Gener. Comput. Syst. 29 (6) (2013) 1345–1352.
[45] K. Ye, X. Jiang, D. Huang, J. Chen, B. Wang, Live migration of multiple virtual machines with resource reservation in cloud computing environments, in: Proceedings of the IEEE International Conference on Cloud Computing (CLOUD 2011), 2011, pp. 267–274.
[46] K. Ye, Z. Wu, C. Wang, B.B. Zhou, W. Si, X. Jiang, A.Y. Zomaya, Profiling-based workload consolidation and migration in virtualized data centers, IEEE Trans. Parallel Distrib. Syst. 26 (3) (2015) 878–890.
[47] W. Zhang, S. Rajasekaran, S. Duan, T. Wood, M. Zhuy, Minimizing interference and maximizing progress for hadoop virtual machines, ACM SIGMETRICS Perform. Eval. Rev. 42 (4) (2015) 62–71.

Jianhai Chen is currently a Lecturer and an on-the-job Ph.D. candidate of college of Computer Science, Zhejiang University. He got his B.S. degree in Applied Mathematics from Hunan University in 1997 and M.E. degree in Computer Science and Technology from Zhejiang University in 2006. His research interests cover virtualization, cloud computing, performance evaluation & modeling,design and analysis of algorithms, scheduling and bin packing, and network communication. He is a student member of the IEEE and the ACM.

Qinming He is currently a Professor in College of Computer Science & Technology at Zhejiang University, P. R. China. He received his B.S., MS and Ph.D. degree in Computer Science from Zhejiang University, P. R. China in 1985, 1988 and 2000 respectively. His research interests include data mining and computing virtualization.



Deshi Ye is currently an Associate Professor in College of Computer Science at Zhejiang University. He got his B.S. degree and Ph.D. degree in Mathematics from Zhejiang University, China, in 1999 and 2005, respectively. His research interests include design and analysis of algorithms, algorithmic game theory, wireless network and mobile computing, scheduling and bin packing, and performance evaluation.

Wenzhi Chen received the B.S., MS, and Ph.D. degrees in computer science and technology from Zhejiang University, Hangzhou, China. He is currently a Professor at the School of Computer Science and Technology at Zhejiang University. His areas of research include computer architecture, system software, embedded system, and network security. He is a member of the IEEE and the ACM.

Yang Xiang received the Ph.D. degree in computer science from Deakin University, Australia. He is currently a Full Professor at School of Information Technology, Deakin University. He is the director of the Network Security and Computing Lab (NSCLab). His research interests include network and system security, distributed systems, and networking. In particular, he is currently leading his team developing active defense systems against large-scale distributed network attacks. He is the chief investigator of several projects in network and system security, funded by the Australian Research Council (ARC). He has published more than 170 research papers in many international journals and conferences, such as IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Information Security and Forensics, and IEEE Journal on Selected Areas in Communications. He has served as the program/general chair for many international conferences such as ICA3PP 12/11, IEEE/IFIP EUC 11, IEEE TrustCom 13/11, IEEE HPCC 10/09, IEEE ICPADS 08, NSS 11/10/09/08/07. He has been the PC member for more than 60 international conferences in distributed systems, networking,and security. He serves as the associate editor of the IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems,Security and Communication Networks (Wiley), and the editor of Journal of Network and Computer Applications. He is the coordinator, Asia for IEEE Computer Society Technical Committee on Distributed Processing (TCDP). He is a senior member of the IEEE.

Kiew Chiew Holding a Ph.D. degree awarded by Nanyang Technological University in Singapore, currently Kevin Chiew is R&D Lead with Handal Indah Company in Singapore, working on R&D projects related to big data analysis. Prior to this, he had been an Assistant and Associate Professor with Asian and Australian universities. He has conducted research in various areas such as data mining, information systems and security, and autonomous and intelligent systems, and has got nearly 60 papers published by internationally reputable journals (e.g., IEEE Trans., KAIS, and ORIJ) and conferences (e.g., ACM SIGIR, ICDE, SDM, and PAKDD).

Liangwei Zhu is currently a software engineer in Ali cloud computing Co., Ltd., working on the researches on cloud computing and virtualization technology and development. He got his B.S. degree in Computer Science from Nanjing University Of Information Science & technology in 2011 and M.E. degree in Computer Science and Technology from Zhejiang University in 2014. His research interests cover virtualization and cloud computing, performance evaluation, scheduling algorithms.