2013 IEEE 27th International Conference on Advanced Information Networking and Applications

AAGA: Affinity-Aware Grouping for Allocation of Virtual Machines

Jianhai Chen[1], Kevin Chiew[2], Deshi Ye[1], Liangwei Zhu[1], Wenzhi Chen[1]

[1] *College of Computer Science*
Zhejiang University
Hangzhou, P.R. China 310027
{chenjh919,yedeshi,zhulw,chenwz}@zju.edu.cn

[2] *School of Engineering*
Tan Tao University
Duc Hoa District, Long An, Vietnam
kevin.chiew@ttu.edu.vn

***Abstract*—Virtualization technology enables various application services to be distributed and encapsulated within virtual machines (VMs), which are dynamically allocated to physical machines (PMs) in cloud computing environments. However, in many existing virtualized systems, the limited network bandwidth often becomes a bottleneck resource, leading to the intensification of network competition and the performance degradation for communication or data intensive applications. Aiming at reducing communication overheads and improving the application performance, in this paper, we propose an Affinity-Aware Grouping method for Allocation of VMs (AAGA). Firstly, we identity and model the problem of affinity-aware grouping-based allocation for virtual machines, and propose a detailed grouping method based on which a heuristic bin packing algorithm is used to deploy VM groups into PMs. In order to demonstrate the effectiveness of AAGA, we create multiple real virtual clusters (multi-VCs) with 56 VMs running multi-VM applications and compare application performance with Non-Affinity-aware Grouping-based Allocation methods (NAGA). Experimental results show that AAGA achieves better performance than NAGA.**

***Keywords*-Virtualization; Affinity Grouping; Resource Allocation; VM Packing; Virtual Cluster; Cloud Computing**

I. Introduction

Virtualization has become a crucial technology in cloud computing, in which applications, such as parallel computing applications [1] and multi-tier e-business web applications [2], are encapsulated within multiple virtual machines (VMs), and dynamically assigned to a pool of physical machines (PMs) for provisioning cloud services [3], [4]. The execution of application jobs inside VMs generates a large amount of communications or data exchanges across these VMs [5]. With the increasingly growing demands of handling cloud service provision tasks, the network, as a key infrastructure in a cloud datacenter, is sustaining a tremendous pressure. Due to the poor efficiency of network virtualization and resource allocation, the network bandwidth becomes a bottleneck in many existing virtualized datacenters, leading to the intensification of network congestion and performance degradation for communication or data intensive applications [6].

To run communication intensive applications in a cloud datacenter, allocating VMs to various PMs in the same or different racks leads in both distinct performance for applications and obvious impacts on the network performance. Actually, frequent communications of VMs imply a large amount of network traffic, blocking the network and affecting the availability of a cloud datacenter when VMs are decentralizedly placed [7] onto different PMs or racks underlying various network topologies, such as Tree, Fat-Tree, and VL2 [8]. Indeed, virtualization owns a mechanism of Inter Domain Communication (IDC) [9], where two VMs co-located in the same PM process communications without getting through physical network interface card (NIC) but via memory sharing ways. This mechanism indicates that colocating communicating-VMs can reduce the communication overhead and improve the performance for communication-intensive applications, and implies that co-locating such VMs during VMs allocation is meaningful. Besides, co-locating VMs onto the same PM can also offer benefit of saving memory while sharing memory pages [10]. However, consolidating over-loaded VMs to the same PM can bring with resource contention of CPUs and reduce application performance.

Unfortunately, *dependency across VMs*, such as communication-awareness, is rarely considered in the existing research on the resource allocation based on VMs [3], [11]–[14]. These allocation solutions, without considering dependency across VMs, adopt virtual machine packing (VMP) techniques, a type of vector bin packing method [12], [15], in which the VMs and PMs are multi-dimensional vectors including dimensions like CPU, RAM, disk and network I/O, etc., aiming at assigning a given number of VMs (items) with distinct resource requirement to a minimal number of PMs (bins) with a fixed resource capacity one-by-one according to the VM size, and matching that the total size of VMs in each PM is less than PM capacity.

In our work, we address a resource allocation problem in which VMs have resource demands and *dependency across VMs* is identified as *affinity relationship*. We present case study to find affinity of VMs and group affine VMs as a whole unit for allocation of VMs. The observations motivate us to allocate affine VMs so that they are co-located so as to improve system performance. Then we model the **A**ffinity-aware **G**rouping for **A**llocation of VMs Problem (AGAP). Further, we propose a solution that includes a tool which can automatically obtain affinity of VMs, and an **A**ffinity-**A**ware **G**rouping-based **A**llocation method (AAGA).

The major contribution we made in this work is the method to identify the affinity relationship among VMs together with an algorithm for grouping affine VMs for PM resource allocation, by which we can significantly improve the system performance measured by running communication/data intensive applications such as HPCC [1] on a real cloud environment.

The remaining sections are organized as follows. We give the background and motivation in Section II and present the problem and solutions in Sections III & IV, followed by experiments in Section V and related work in Section VI before concluding the paper in Section VII .

II. Background and Motivation

In this section, we present several case studies on measuring traffics and performance derived from several typical cloud benchmark applications running amongst multiple VMs. The observations motivate us to identify affinity relationships between VMs and engage in affinity-aware grouping VMs for resource allocation.

A. Measuring Traffic Dependency Between VMs

1) Traffic Dependency: In cloud datacenters, applications are hosted among one or more VMs, thus they can be classified into two classes, namely single- and multi- VM applications.

The running of many multi-VM applications will bring with communications between VMs in realtime, generating communication dependency and showing up traffic flows passing through the shared network, which naturally is a traffic dependency between VMs. We assume that a VM hosting a single-VM application has not traffic dependency with other VMs except a $domain$ within a Hypervisor, such as $domain0$ in Xen [9]. Moreover, application running in different time window will show both distinct resource usage and dynamic change of traffic volume between VM pairs. Besides, let a and b denote two VMs, the traffics have a bidirectional transmission feature because traffic volume from a to b is different from that from b to a within a time window or total application runtime.

2) Traffic Measuring Case Study: We are interested in finding traffic dependency between VMs and provide a case study, in which we measure traffic between VM pairs in the process of running typical cloud applications in datacenter network. Generally, we can benefit from the prior knowledge of traffic dependency between VMs while allocating VMs to PMs for minimizing network communication overhead. To do this, a crucial step is to capture and analyze the mass traffic fingerprinting.

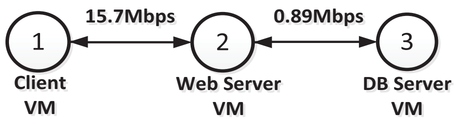


Figure 1. Traffic rate amongst 3 VMs (1–3) running RUBiS benchmark

Based on tcpdump [16] method we implement a software tool to monitor traffic among physical network automatically and capture traffic fingerprinting between VM pairs. In every time interval (e.g., second or minute) during the application running, a total bytes of all the transferred packets between VM pairs are summed up as traffic volume. Actually a pair of VMs (a, b) will have a different traffic volume in the two transmission directions. For simplicity, we assume they are identical otherwise we use average of them. Therefore we give a metric of average bandwidth (AVG_BW): bps (bytes per second) or pps (packet-amounts per second) to denote the communication dependency between a VM pair. We can conclude this metric by dividing the total traffic volume to a total runtime like total seconds consumed in running application, as shown in the following formula. Obviously it also denotes an average traffic rate between a VM pair.

$$AVG\_BW = Total\_traffic\_volume / Total\_time(sec.).$$

Three typical cloud applications are chosen in this case study. The first is RUBiS [17], a multi-tier emulation of e-bay web application. It simulates various tasks done by various clients, including user & item registration, browsing items per category & per region, bidding for or buying items and so on. 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 of RUBiS , the number of clients, a primary parameter determining the workload size, is set to 1400, and other parameters are set default. We run five times, each run lasts twenty minutes and get the average of total traffic volumes, then concludes the traffic rate of all VM pairs. Figure 1 shows the traffic rate among three VMs. We observe that different VM pair exists distinct traffic rate, i.e., the traffic rate between client and web server VM performs 17.6 times of the one between web and DB server VM.

The second cloud application is Hadoop [18], which 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. 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 select one typical workload: wordcount which is used to count all words inside a certain word file under a distributed multi-nodes platform. The word file size, a chief parameter, is set to 100MB. We capture traffic fingerprinting

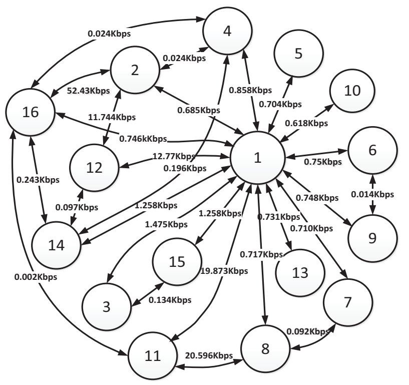


Figure 2. Traffic amongst 16 VMs (1–16) running a Hadoop workload: wordcount

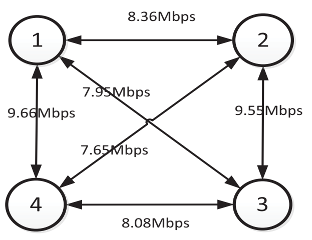


Figure 3. Traffic amongst 4 VMs (1–4) running HPCC benchmark

between all VM pairs and conclude the traffic rate as shown in Figure 2.

The third cloud application is the HPC Challenge benchmark (HPCC) [1], which is used to measure the performance of parallel computing in a cluster with several nodes. In this case study a virtual cluster (VC) is set up with four VMs. There are four main parameters relating 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. We set matrix A to 1000×1000 and others to be default. We run three times the test and get the average of the results as shown in Figure 3.

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 the counter-productive, the traffic rates between VM pairs are very small and much less than the one derived from RUBiS. The large amount of traffic indicates that the VMs hosting HPCC benchmark application require much more network bandwidth than others.

In conclusion, the case study tells us that the traffic dependency between VM pairs reveals a great difference among distinct multi-VM applications. It is natural that colocation-placing VMs with the larger or heavier traffic dependency will offer us much better benefit for application and physical network performance. This motivates us to try another case study in the next subsection.

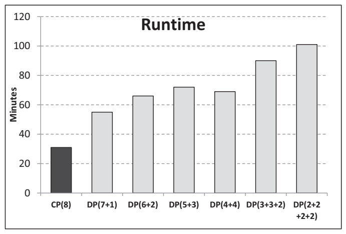


Figure 4. The performance evaluation of running efficiency for HPCC

B. Collocation of VMs for More Performance Gains

In the next, we provide a case study to prove that colocation of VMs hosting communication intensive applications can offer much more benefit for performance. We choose the HPCC benchmark for our case study and consider the application performance underlying different schemes for allocation of VMs to PMs.

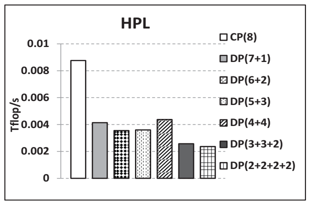
For simplicity, we identify two deployment patterns as follows. (1) Colocation-placement, denoted by CP($\cdot$), for example, CP(8) denotes 8 VMs as one group placed onto one PM whilst idling the other PM; and (2) disperse-placement, denoted by DP($\cdot$), for example, DP($7 + 1$) denotes eight VMs are divided into two groups with 7 VMs as a group placed onto a PM and 1 VM as a group place onto the other PM. Given eight VMs and two PMs, there are 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, concerning that the number of PM 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 other four metrics from four communication-aware benchmarks, namely, HPL, PTRANS, FFT and Avgpingpong from HPCC benchmark.

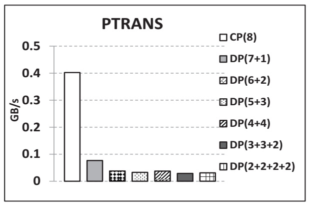
We first compares the runtime of all schemes as shown in Figure 4. We can see that CP(8) needs the least runtime as compared with others, meaning that all eight VMs placed onto one PM can finish running with the runtime less than half of that of DP($4 + 4$) or one fourth of that of DP($2 + 2 + 2 + 2$).

Figure 5 shows the performance comparison in terms of the different metrics from communication/data-dependence benchmarks under different VM deployment schemes. It tells that CP(8) outperforms all other schemes for the four metrics; especially, for metric PTRANS, CP(8) outperforms five times than DP($7 + 1$) or 10 times than others.

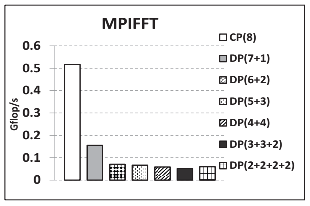
From the above observations, we conclude that for heavy communication intensive applications, colocation placing VMs can obtain much better performance, and the



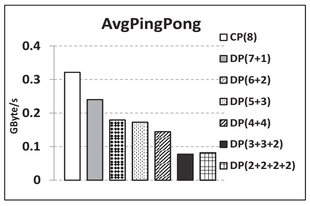
(a) Floating point rate of execution HPL



(b) Total communication capacity of the system interconnect



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



(d) Communication bandwidth

Figure 5. The HPCC benchmark performance evaluation amongst different metrics in distinct CP and DP schemes

colocation-placing VMs with communication dependency can reduce the communication overhead of physical network but not sure of improving the performance.

C. Motivation of Affinity Grouping

The above observations imply that there is a certain relationship such as communication dependency derived from network traffic amongst VMs during application running process, such that if some of VMs with communication dependency are bundled as whole unit and allocated to a PM, then we can minimize the communication overheads across physical network and enhance application performance. We refer to the dependency between the VM pair as affinity of VMs (to be elaborated later). These findings motivate us to further investigate how to group an array of VMs given that there exists affinity relationship across some of them, and how to allocate and place these groups of VMs to PMs so as to reduce the communication overhead and improve performance as required in real applications.

III. Problem Statement

In this section, we first provide two definitions: VM affinity relation and VM-affinity group, respectively, and several rules for grouping VMs based upon affinity, followed by a statement of problem of AAGA: **A**ffinity-**A**ware **G**rouping based **A**llocation of VMs problem.

A. Affinity of VMs

Definition 1: **Affinity of VMs**. The dependency between VMs is defined as affinity of VMs.

According to Definition 1, in cloud computing we can also have many types of affinity, such as communication affinity (CA), data affinity (DA), memory affinity (MA), and user-defined affinity (UA) as all others.

CA is derived from the communication dependency between VMs running communication intensive applications. While MA is derived from memory dependency between VMs when two VMs colocation benefits for sharing memory [10]. DA is induced from the dependency between VMs running data-intensive applications. Besides, a cloud customer requires his VMs be tackled with colocation. In this regard, these VMs with affinity are derived from user-aware dependency, named as user-defined affinity.

Regardless of various affinity types, our work denotes affinity of VMs as a key factor for allocation of VMs onto PMs, and takes communication affinity as an example.

B. Rules of Affinity-Aware Grouping VMs

In the next, we detail some rules for performing affinity-aware grouping VMs and provide mathematically formalized definitions and claims based on set relation theory.

Definition 2: **Affinity Relation**. Two VMs having affinity defines an *affinity relation*.

Mathematically, affinity relation is a binary relation. Given a set V of VMs, $V \times V$ be the product set of V and V, $\mathcal{AR}$ be an affinity relation set, and $\mathcal{AR} \subset V \times V$. If $\forall x, y \in V$, $(x, y) \in \mathcal{AR}$, then we call x and y have an affinity relation $\mathcal{AR}$, denoted by $x \sim y$.

In addition, according to the directionality of affinity, we can have two other classes of affinity relations: *direct affinity relation* and *indirect affinity relation*. A VM pair (x, y) with affinity forms a *direct affinity relation*, while if three VMs x, y and z, matching $x \sim y$ and $y \sim z$, then we say x and z have an *indirect affinity relation*.

Definition 3: **Affinity VM Group (AVMG)**. Given a set V of VMs, let x be a VM, an affinity VM group $\mathcal{AG}$ is defined as a non-empty subset of V, $\forall x, y \in \mathcal{AG}$, x and y have a direct affinity relation or indirect affinity relation. Let $x = y$, then one single VM forms an affinity VM group.

Besides, we also can use a graph to signify an affinity VM group based on graph theory. In an affinity VM group, all VMs form a set of vertices and the affinity relations can be viewed as edges. Every two VMs with direct or indirect affinity relation equal to a direct or indirect path from one VM vertex to another VM vertex. From this point of view, one affinity VM group constructs a complete graph.

We provide the affinity VM group to decide allocation of VMs to PMs and to guarantee the VMs with affinity as being closely placed as possible.

From these definitions, we have the following claims as rules to generate affinity VM groups as follows.

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

Claim 1: A single VM forms an affinity VM group.

Proof: A VM and itself have affinity relationship, because it will be obviously allocated to the same PM. ■

Claim 2: A set of VMs with two VMs, which have affinity relationship form an 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 VM-affinity group on x and y. Due to x and y having affinity relation, x and y are allocated to the same PM, namely, all VMs in $\mathcal{AG}$ are allocated to one PM. ■

Claim 3: The union of two affinity VM groups forms an affinity VM group, i.e., 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: If $\mathcal{AG}_1$ and $\mathcal{AG}_2$ are not allocated to the same PM, then x and y are not 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, the affinity between x and y is guaranteed. ■

C. Affinity Grouping-based Allocation Problem

We consider a basic scenario that comprises a large number of VMs and PMs in a cloud datacenter. The VMs are deployed to run a variety of application services with resource demands. The PM has a resource capacity in each dimension corresponding to the dimension of VM resource vector. There are traffic dependency between a part of VMs. We aim at minimizing the number of PMs for high consolidation-based energy-saving in a cloud datacenter.

In this scenario, we describe an Affinity Grouping-based Allocation (AGAP) problem as follows. Given n VMs and m PMs, some affinity relationships across VMs, finding an optimal grouping and allocation scheme, such that the VMs are grouped into a set of affinity groups which are allocated into the minimal number of PMs.

This problem includes two parts, namely (1) affinity grouping; (2) affinity group allocation. We accordingly can have the following two assumptions, namely (1) an optimal grouping enables each VM group to have the maximal number of VMs and each VM has affinity relationship with at least one VM in the same group and has no affinity relationship with any VM outside this group; (2) the total resource demand of each affinity VM group is less than the resource capacity of one PM.

IV. Methodologies

In this section we present some methodologies to solve our problem. We first introduce a tool to find affinity of VMs, and then propose a grouping algorithm to group VMs into affinity groups as the basic units for allocation, and give several heuristic bin packing techniques to make decision of allocating all these groups of VMs onto PMs.

A. Finding Affinity of VMs

To find affinity of VMs, we implement an affinity prober tool based on tcpdump and analyze the traffic fingerprinting results to extract affinity of VMs. The tool is deployed onto each PM hosting a Hypervisor, such as Xen's $domain0$, and runs in realtime, and captures network traffics over the NIC. Each traffic record signifies a network packet transferred from a source server to a destination server. It is a triple like <sourceID, destinationID, volume>, in which the sourceID and destinationID denote the source VM and distinction VM IP address, respectively, and the volume denotes the total summed packet size (bytes) of traffic captured within a period of monitoring time.

B. Affinity-Aware Grouping Algorithm

The claims in Section III are used as rules to group the VMs into VM-affinity groups. According to the rules we provide an affinity-aware grouping algorithm to do grouping for affinity-aware resource allocation.

Given a set V of VMs and a set $\mathcal{AR}$ of VM-affinity relation between VMs, the VM-affinity grouping algorithm aims to group the VMs into a set of disjoint subsets of V each of which is a maximized VM-affinity group. A maximized VM-affinity group is such a group inside which any VM has VM-affinity relation with at least one VM in the group and does not have VM-affinity relation with any other VM outside the group. We denote our VM-Affinity grouping algorithm as Max-VA-Grouping (MVAG).

The detailed steps of MVAG algorithm is as follows.

Step 1. Input a VM set V with the number N, and a VM-affinity relation set VR with the number E.

Step 2. Initially each VM of V is built as a VM-affinity group set (singleton set).

Step 3. For each VM in a VM-affinity relation of AR, find the VM-affinity group set which contains the VM, and get two VM-affinity group sets.

Step 4. Apply union operation to the two VM-affinity group sets to get a new VM-affinity group set.

Step 5. If there are any other VM-affinity relations, then goto Step 3; otherwise goto Step 6.

Step 6. Output all the VM-affinity group sets as final grouping result.

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 tree-based data structure and path compression to minimize the computation complexity. Similar as disjoint set algorithm, the time complexity of MVAG algorithm is O(N+E), where N is the number of VMs and E is the number of edges.

C. Allocation of VM-affinity Groups

After affinity grouping, the next step is to allocate these affine groups onto a minimal number of PMs.

Bin packing in VM placement is called VM packing (VMP), which is also a type of vector bin packing (VBP) [19]. In our affinity group allocation problem, we target to allocate affinity VM groups to PMs. This is an optimization problem as follows. Given a list of VM affinity groups (items), an affinity group set $\mathcal{AG}$ of k d-dimensional vectors $g_1, g_2, ..., g_k$ from $[0,1]^d$, find a packing scheme of $\mathcal{AG}$ into $B_1, B_2, ..., B_m$ such that $\sum_{g \in B_i} g^h \leqslant 1, \forall i, h$ (g^h denotes the projection of the hth dimension of vector g). The objective is to minimize the number of PMs.

Many greedy algorithms, such as heuristic bin packing, can be used to solve the affinity group allocation problem. These heuristics include the First Fit (FF), Best Fit (BF), Nest Fit (NF), First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD) for one dimensional item, and many variants of FFD, such as FFDProd, FFDAvgSum etc., for multi-dimensional items according to a measure used to decompose a vector item as a numerical value used for sorting items in a decreasing order [19].

D. Discussion

It is reasonable to suppose that each affine-VM-group can be allocated onto one PM, namely no group resource demand exceeds the limited PM capacity. Firstly, the increasingly growing hardware technology brings one PM with increasing size of resource capacity, such as CPU and RAM. Then, we can use a virtual-large-PM, which is a cluster of several PMs deployed onto one or more racks connected with infiband. Further, cloud tenants apply for several VMs to run their cloud services. We 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 number of VMs with affinity generating one large affinity group with total resource requests overstepping the PM resource limit. If there are no PMs can take in all VMs in one large affinity group, then a mini-cut method [7] can be used to divide the large group into several small groups, across which having minimized communication, such that each group resource request is less than one PM capacity and the network communication overhead is minimized. This is another interesting problem of our future work.

V. Experimental Results and Analysis

In this section, we provide experiments to demonstrate the effectiveness of the AAGA method by comparing it to the NAGA method.

A. Experimental Schemes

Given many VMs and PMs, we present two schemes based on two VM deployment schemes corresponding to AAGA and NAGA method, respectively. In each scheme we run the same multi-VMs application and compare the application performance results. Here multi-VMs application is defined by an application hosted on multiple VMs to run.

B. Experimental Environment

A cloud datacenter generally runs many kinds of applications hosted amongst multiple VMs, and we denote a group of the VMs for running one application as a VC. Hence a cloud datacenter comprises several VCs. Different VCs may run different applications or the same application with different input workloads or data scales. We focus on the case that one application is hosted with many VMs. For simplicity, we only choose one multi-VM application with different size of parameter input for each VC.

Table I
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

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 @ 2.40GHz, each of which has totally 16 Cores and 32GB 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.

We choose HPCC benchmark generating several VCs in according the different number of nodes and application scale parameters. We construct 6 distinct VCs with total 56 VMs as nodes by given different configurations. Each VC is configured with distinct number of VM nodes with different total CPU and memory. The VMs of a VC are set with identical amount of resource RAM.

The configuration of VC1–VC6 is listed in Table I. The VMs in one PM share the CPU limited to 32 VMs, and the RAM capacity is limited to 12GB (total 32GB) considering the resource reservation [20]. As shown in Table I, we limit each affinity-group with a total resource (RAM) demands less than the PM capacity, and are deployed in one PM.

C. Deployment Schemes

According to the two experimental schemes we implement two specific deployment schemes. In each scheme the deployment pattern is determined by a certain placement strategy. The placement solution is concluded by a given allocation strategy, denoted as a table, in which the row denotes PM, the column denotes VC and a numerical value in a cell (i, j) denotes the VM number of VC_j allocated to PM_i. One VC runs a multi-VM application and the VMs exist traffic dependency, generating one affinity group.

The first scheme adopts the AAGA method, denoted by AAGA. The VMs are firstly identified with affinity relationships and grouped into several affinity groups via

Table II
FFD-BASED AFFINITY GROUP VM PLACEMENT

PM—VC	VC1	VC2	VC3	VC4	VC5	VC6
PM1	16	0	0	0	0	0
PM2	0	8	0	0	10	0
PM3	0	0	6	12	0	4

Table III
FFD-BASED VM PLACEMENT

PM—VC	VC1	VC2	VC3	VC4	VC5	VC6
PM1	0	8	4	0	0	4
PM2	16	0	2	0	5	0
PM3	0	0	0	12	5	0

Max-VA-Grouping. Then, we arbitrarily use a bin packing method FFD to allocate these VM-affinity groups onto PMs and the detailed deployment is listed in Table II.

The other scheme employs the VM packing method without considering affinity of VMs, which is denoted by NAGA. The VMs are deployed onto PMs one by one using the same bin packing method as AAGA, i.e., FFD. The detailed deployment is listed in Table III, from which we can see that the VMs of VC3 and VC5 are deployed amongst two PMs, i.e., VC3 as $(4 + 2 + 0)$ and VC5 as $(0 + 5 + 5)$, while the VMs of other VCs are still placed centrally on one PM, e.g., VC1 as $(0 + 16 + 0)$.

D. Running and Results Analysis

The HPCC multi-VM applications over the six VCs in each scheme run concurrently three times and result in several files each time. To evaluate the performance, 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, Avg-PingPongBandwidth_GBytes for PingPong benchmark of b_eff, MPIFFT_Gflops for FFT. Table IV gives the average result for each metric value.

From Table IV, we can observe that all performance of VC3 and VC5 generated from scheme AAGA are better than those generated by scheme NAGA. Particularly, the performance of MPIFFT benchmark in VC5 derived from AAGA scheme outperforms 23.5 times of the one from NAGA scheme, while VC3 outperforms 5 times. For other VCs, AAGA scheme has better performance than NAGA scheme. However, because of the impact of resource contention derived from VMs sharing CPU and memory, a few cases do not show expected performance, such as MPIFFT in VC6, AvgPingPong in VC2 and VC6, VC6 under AAGA and NAGA scheme.

In addition, in our experiments, we take the summation of metric value for each benchmark from these VCs to compare the overall performance of the simulated cloud multi-VCs system, because all VCs run the same benchmark within AAGA and NAGA deployment schemes. Figure 6 shows the overall performance results which demonstrate

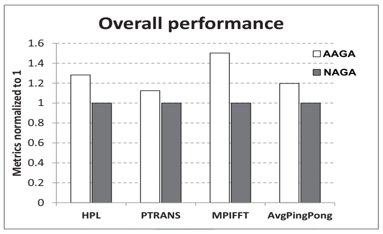


Figure 6. The overall system performance evaluation for each benchmark under AAGA and NAGA

the effectiveness of the AAGA method, because AAGA outperforms NAGA in all four performance metrics, i.e., HPL improves 28.3%, PTRANS 16.3%, MPIFFT 87.6%, and AvgPingPong 19.7%, respectively.

VI. RELATED WORK

The related work is categorized into two parts: affinity and non-affinity.

A. Affinity-aware Studies in Virtualized Systems

In virtualized systems, a few study have been done on affinity. First, Chen and Li [21] employ affinity to implement a new schedule strategy to improve the efficiency of virtualized resource scheduling. The proposed affinity is used to identify the relation between a virtual CPU and a CPU in VMM or Hypervisor. Second, Sonnek *et al.* [6] presented an affinity-aware VM migration technique to minimize the communication overhead on a virtualized platform. The affinity identified a policy or a technique of VM migration for a dynamic resource allocation. Moreover, Meng *et al.* [7] proposed a traffic-aware VM placement to improve the network scalability.

Recently Sudevalayam *et al.* [5] attempted to evaluate performance of virtualized applications hosted among two VMs with colocation affinity. It focuses on performance evaluation but not resource allocation. Besides, VMWare [22] use affinity to signify the relationship between VMs which are kept together as one unit in VM placement. This proves the practicality of our study on affinity-aware resource allocation.

B. Non-affinity Resource Allocation In Virtualized Systems

Virtualization enables easily resource allocation in datacenters and cloud computing. Many efficient solutions have been proposed for VM resource management, together with some resource allocation methods [11], [13], [14]. As a resource allocation problem, researchers have proposed various algorithms such as bin packing and optimization methods to VM initial placement problem. Wilcox *et al.* [15] solved the VM placement as vector packing with Grouping Genetic Algorithm. Panigrahy *et al.* [19] addressed the heuristics vector bin packing based on FFD approximate approaches to the vector backing problem. Their work is

Table IV
THE METRIC RESULTS OF BENCHMARKS IN ALL VCS UNDER AAGA AND NAGA SCHEME

Benchmark (Metric)	Deployment Scheme	VC1	VC2	**VC3**	VC4	**VC5**	VC6	Total
HPL(Tflop/s)	AAGA	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)	AAGA	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)	AAGA	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)	AAGA	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

non-affinity-awareness and the techniques can be helpful to our affinity-aware work.

VII. CONCLUSION

In this paper, we have studied the problem of affinity aware grouping for allocation of virtual machines. The contribution we made is three-fold. (1) We have identified the relationship of affinity across VMs from real application cases. (2) We have proposed an affinity grouping algorithm to group the VMs which are known with affinity relationships across VMs. (3) We have conducted comprehensive experiments with different allocation methods to show that affinity grouping can help us improve system performance. For future study, we will investigate some techniques to identify affinity relationships from a list of VMs, and study how to handle the case when the capacity of a PM is less than the resource demand of a group of VMs with affinity.

ACKNOWLEDGMENT

This research is supported in part by the National Key Technology Research and Development Programm of China under Grant No. 2011BAD21B02, MOE-Intel IT Research Fund of China under Grant No. MOE-INTEL-11-06, the key Science and Technology Innovation Team Fund of Zhejiang under Grant No. 2010R50041-12, and NSF of China under grant Nos. 60970125 and 61272303.

REFERENCES

[1] P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the hpc challenge benchmark suite," 2005.
[2] X. Wang, Z. Du, Y. Chen, and S. Li, "Virtualization-based autonomic resource management for multi-tier web applications in shared data center," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1591–1608, 2008.
[3] G. Soundararajan, D. Lupei, S. Ghanbari, A. Popescu, J. Chen, and C. Amza, "Dynamic resource allocation for database servers running on virtual storage," in *Proccedings of the 7th conference on File and storage technologies (FST 2009)*, 2009, pp. 71–84.
[4] J. Sonnek and A. Chandra, "Virtual putty: Reshaping the physical footprint of virtual machines," in *Proceedings of Workshop on Hot Topics in Cloud Computing (HotCloud 2009)*, 2009.
[5] S. Sudevalayam and 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.
[6] J. Sonnek, J. Greensky, R. Reutiman, and 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.
[7] X. Meng, V. Pappas, and 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.
[8] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware vm placement for cloud systems," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 498–506.
[9] 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," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.
[10] M. Sindelar, R. Sitaraman, and P. Shenoy, "Sharing-aware algorithms for virtual machine colocation," in *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*, San Jose, CA, 2011.
[11] F. Hermenier, X. Lorca, J. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments(VEE 2009)*, 2009, pp. 41–50.
[12] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.
[13] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923 – 2938, 2009.
[14] N. Vasic, D. Novakovic, S. Miucin, D. Kostic, and 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)*, vol. 12, 2012.
[15] D. Wilcox, A. McNabb, and 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.
[16] "tcpdump." [Online]. Available: http://linux.die.net/man/8/tcpdump
[17] "Rubis." [Online]. Available: http://rubis.ow2.org/
[18] "Apache hadoop." [Online]. Available: http://hadoop.apache.org/
[19] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research.microsoft.com*, 2011.
[20] K. Ye, X. Jiang, D. Huang, J. Chen, and 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.
[21] H. Chen, H. Jin, and 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, Nov. 1–5, 2010, pp. 75–80.
[22] V. Infrastructure, "Resource management with vmware drs," *VMware Whitepaper*, 2006.