Hosting Virtual Machines on a Cloud Datacenter: A Matching Theoretic Approach

Chuan Pham*, Nguyen H. Tran*, Minh N.H. Nguyen*, Shaolei Ren†, Walid Saad*‡, Choong Seon Hong*.
*Department of Computer Science and Engineering, Kyung Hee University, Korea.
†Department of Electrical & Computer Engineering, University of California at Riverside, USA.
‡Bradley Department of Electrical and Computer Engineering, Virginia Tech, USA.

***Abstract*—In this paper, the problem of resource allocation in cloud datacenters, that own highly complex and heterogeneous tasks and servers, is considered. To address this problem, a novel framework, dubbed joint operation cost and network traffic cost (JOT) framework, is proposed. This framework combines notions from Gibbs sampling and matching theory to find an efficient solution addressing the NP-hard problem JOT. The proposed model is shown to be capable of controlling the active server set, in a coordinated manner while allocating VMs in order to reduce both operation cost and network traffic cost of the cloud datacenter. We also conduct a case-study to validate our proposed algorithm and the results show that JOT can reduce the total incurred cost by up to 19% compared to the existing non-coordinated approach.**

Keywords*—Datacenter, Virtual Machine, Heterogeneous Tasks, Heterogeneous Server, Resource Allocation.

I. Introduction

The use of cloud datacenters witnessed an unprecedented growth in the past few years, owing the flexibility and agility of the platform which has motivated many business to rely on resources hosted on a cloud datacenter [1]. Deploying applications into cloud-hosted virtual machines (VMs), will reduce the complexity of the physical infrastructure in datacenters. For example, using such virtualization, system administrators can easily perform many task such as create connections between VMs, migrating a database VM, or duplicating a VM, among others.

Considering the resource management policies of the datacenters, it is challenging to develop an efficient resource allocation solution that can satisfy all the complexity constraints of cloud providers and requirements of heterogeneous user demands [2]. One basic challenge for VM resource allocation is the traditional problem of VM placement. VM placement often consists of a procedure that places VMs into servers in order to improve the utilization of physical resources and reduce the operation cost in terms of controlling number of active server [3], and [4]. The current works [3]–[6] on this area primarily consider only server-side constraints such as CPU, memory, storage capacity, etc., that improve the efficiency of resource usage on the datacenter. However, those works often ignore the relationship between VMs. Such relationship is central to the design of VM demands where in practical users usually submit bundles of VMs to run their applications. These VMs are rarely independent. In contrast, they typically require significant inter-communications among them [7].

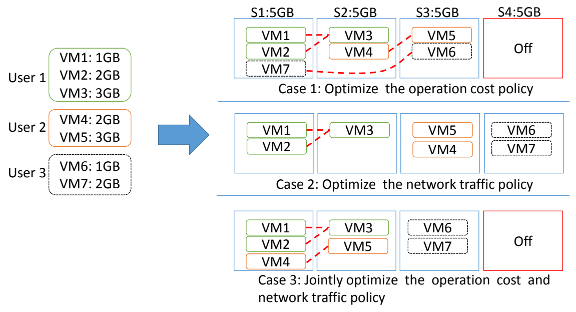

S1:5GB
S2:5GB
S3:5GB
S4:5GB
User 1
VM1: 1GB
VM2: 2GB
VM3: 3GB
User 2
VM4: 2GB
VM5: 3GB
User 3
VM6: 1GB
VM7: 2GB
VM1
VM2
VM7
VM3
VM4
VM5
VM6
Off
Case 1: Optimize the operation cost policy
Case 2: Optimize the network traffic policy
Case 3: Jointly optimize the operation cost and network traffic policy

Fig. 1: Example of resource allocation with different policies.

Due to the importance of such VM interdependence, some existing works such as [4] and [8] have studied these relationships. For example, the authors in [8] focused on reducing the network traffic in migration while the work in [4] designed a two-tier approximate algorithm to reduce the traffic load inside datacenters. Although these works propose novel approaches to reduce the network traffic, they often relax the complexity of the possibly heterogeneous resource allocation in the datacenters.

We now illustrate the benefit of allocating VMs in a coordinated manner so as to reduce both operation cost and network traffic cost. This is in contrast to the conventional non-coordinated VM allocation, which implements, in isolation, the operation cost policy or the network traffic policy. In this scenario, there are three users, who submit the VM demands to the cloud datacenter. With heterogeneous requests, the cloud provider needs to allocate VMs appropriately. Case 1 represents a resource allocation scenario in which the cloud provider tries to minimize the operation cost (e.g., total power consumption of all active servers). This method is also adopted in Sandpiper [9] as well as the consolidation framework of OpenStack [10]. On the other hand, Case 2 considers only the internal network traffic among VMs, so the datacenter consumes a lot of active servers to satisfy user demands. By joint operation cost and network traffic cost, the optimal plan of allocation is shown in the Case 3 of the Fig. 1. Assuming in 1 hour, the cost to operate one server is \$10 and the network delay cost is \$1/link. The datacenter will be charged \$33 to host all VMs in the Case 1 (i.e., three active servers and three links to connect among VMs of User 1 and among VMs of User 3), and similarly, \$42 in the Case 2, and \$32 in Case 3. In this simple example, the VMs of each users should be located close to others within their group.

This example shows the effect of the joint operation cost and network traffic cost on the total cost incurred by datacenters. Even though some recent works, such as [7] and [8], attempt to address this problem, however, these works ignore the heterogeneous demands of users and heterogeneous server in datacenters. Developing a practical solution in the heterogeneous environment of datacenters is therefore a significant and necessary issue.

The main contribution of this paper is to develop a novel framework for resource allocation in a cloud datacenter. We first formulate a combinatorial optimization problem that characterizes the joint operation and network traffic cost (JOT) in the datacenter. Then, we combine Gibbs sampling and matching theory from economics to solve JOT problem. Finally, we design an effective algorithm for resource allocation that coordinately controls the active server set and allocates VMs in order to reduce both operation cost and network traffic cost in cloud datacenters.

The rest of the paper is organized as follows. Section II presents the system model and problem statement. In Section III, we design an optimization algorithm that combines Gibbs sampling and matching method to solve the optimization JOT problem. We then simulate and evaluate our work in the Section IV. Finally, we present conclusion in Section V.

II. System model and problem formulation

A. System overview

To better study the VM allocation scenario on the cloud datacenter, we consider the problem in Software-defined Network (SDN) context, since it provides flexible control on scheduling. This system has been studied in [11]. Wang, *et al.* [11] represented an architecture in cloud datacenter where all networking resources are under the control of SDN controller [12], such as OpenFlow. Devices in the network, switches or routers, are controlled by SDN controller to communicate and gather link-state information (i.e., SDN controller can use a standardized protocol, OpenFlow). SDN controller can measure the network traffic, network topology and traffic matrix of the datacenter that are necessary to compute a VM hosting plan on the datacenter.

On the other hand, to manage the computing resource, OpenStack protocol can be implemented on the cloud controller to gather all information of servers, such as available memory, size of VMs, residual CPU resource of physical hosts. Besides, OpentStack also provides some general functions, such as allocating VMs, migrating, etc., [12].

Based on the system model above, we explain the process of VM allocation and propose an efficient method in assignment for this model. The process of VM allocation is described as follows. First, the user requests are sent to the controller. Based on the collected information from OpenStack and SDN components, the controller makes a plan that allocates appropriately VMs to servers depend on the resource demand and bandwidth requirements. Then, after calculating the optimal allocation plan, OpenStack will carry out the plan at the corresponding time.

B. Problem formulation

We assume that time is slotted and we study the system for one time period. We consider a cloud provider having M servers and in each time slot, it runs cloud services on a set $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ of $m \leq M$ active servers. In this system, a set $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ of n VMs is submitted to the controller by a set $\mathcal{T} = \{t_1, t_2, ..., t_k\}$ of k user tasks. Each user requires an amount of resource (called a task) which is, in fact, a subset of VM, where one VM belongs to exactly one task.

Next, we consider p resource type such as CPU, memory, storage, etc., and denote r_i^p as resource type p load requirement of VM i. Further, r_j^p represents as the available resource type p of server j. For ease of notation, we use r_i and r_j to denote a vector resource of VM i and server j, respectively. To measure the traffic cost of VM i from server j to VM i' located on server j', we use the network latency, $D_{jj'}$, between server j to server j'. This matrix can be obtained by SDN component, as Section II-A, which manages all network devices and network traffic in datacenter. We let X_{ij} be a binary variable that indicates whether VM i is located on serer j ($X_{ij} = 1$) or not ($X_{ij} = 0$).

Next, we define the key constraints of our system. First, the total resource allocation on any server is lower than or equal to its available capacity

$$\sum_{i \in \mathcal{V}} r_i^p \times X_{ij} \leq r_j^p, \forall j \in \mathcal{S}, \forall p. \tag{1}$$

Second, each VM can be located on only one active server as follow:

$$\sum_{j \in \mathcal{S}} X_{ij} = 1, \forall i \in \mathcal{V}. \tag{2}$$

Operation cost. As shown in [13] for VM placement, turning on all servers can have negative effects on the incurred cost of datacenters. Therefore, the operation cost in datacenters depends on the number of active servers and, thus, the cost is often formulated as the power consumption of active servers. We now define the operation cost of the datacenter as follows:

$$E(\mathcal{S}) = \sum_{j \in \mathcal{S}} \alpha Q_j, \tag{3}$$

where α is the price that converts the power to a monetary term, Q_j is the active power of server j.

Traffic cost. Considering the internal traffic that circulates between VMs, we measure the network traffic in the datacenter based on the parameter $D_{jj'}$

$$\begin{aligned} G(X) = \beta D_{jj'} X_{ij} X_{i'j'}, \\ \forall i, i' \in \mathcal{V}, \forall j, j' \in \mathcal{S}, \end{aligned} \tag{4}$$

where β is the price that converts the network delay to a monetary term, and $G(.)$ is the traffic cost function between VM i in server j to VM i' in server j'.

Combining the two cost models above gives the following total operation cost $C(\mathcal{S}, X)$ of the datacenter:

$$C(\mathcal{S}, X) = E(\mathcal{S}) + \sum_{i \in \mathcal{V}} G(X). \tag{5}$$

To combine two cost models with different units, we convert them to the monetary by parameter α and β, which can easily adjust to adapt with a specific system.

JOT: Joint operation cost and network traffic problem.

The joint operation cost and network traffic problem (JOT) can be formulated as follows.

$$\begin{aligned}
\textbf{JOT}: \min \quad & C(\mathcal{S}, X), \\
\text{s.t.} \quad & \sum_{i \in \mathcal{V}} r_i^p \cdot X_{ij} \leq r_j^p, \forall j \in \mathcal{S}, \forall p, \\
& \sum_{j \in \mathcal{S}} X_{ij} = 1, \forall i \in \mathcal{V}, \\
& X_{ij} = \{0,1\}, \forall i \in \mathcal{V}, \forall j \in \mathcal{S}, \\
& |\mathcal{S}| \leq M, \\
\text{var.} \quad & \{\mathcal{S}, X\}.
\end{aligned} \tag{6}$$

However, the problem above cannot be found in the polynomial time, since it is NP-hard. Many works in this area reduce the complexity of this problem by relaxing the traffic cost in the datacenter, such as [10] and [14], meanwhile some other papers only consider to reduce the network traffic and ignore the operation cost in the datacenter [7] and [8]. Among the many choices of optimization methods, the one we advocate below has the advantage in solving the high complexity of the combinatorial optimization problem.

III. Gibbs sampling based for JOT problem

A. JOT algorithm

Basically, JOT algorithm follows the phases of Gibbs sampling method. Gibbs sampling is a stochastic optimization method that can achieve the global optimal solution by probabilistically transitioning among possible states in the solution space [15].

- *Calculate the current state*. In the initialization step, the controller randomly chooses a subset $\mathcal{S}_0$ of active servers, and a feasible X_0 that satisfies the constraint (1) and (2) and sets $m^* \leftarrow |\mathcal{S}_0|, X^* \leftarrow X_0$ to compute the optimal value C^* of JOT.
- *Calculate the new state*. To move on the next state, JOT algorithm randomly chooses the new subset $\mathcal{S}'$, satisfying (1). Next, it sets $m' \leftarrow \mathcal{S}'$, then solves JOT to obtain the new X' and optimal value C' (i.e., line 2 in Algorithm 1).
- *Move to the new state with probability* $\boldsymbol{p}$. Based on the transition state of Gibbs-sampling method, JOT moves to the new state with probability p (i.e., lines 3, 4). The parameter $\delta > 0$, referred to as the tunable smoothing parameter, is used to control exploration versus exploitation (i.e., the degree of randomness). As δ increases, JOT algorithm becomes more greedy and chooses a new solution with a greater probability if it is better than the current solution (i.e., $C^* \leq C'$) [15].
- *Keep the current state with probability* $1 - \boldsymbol{p}$. If the new state gives a worse optimal value than the current C^*, the controller will keep the current state (i.e., line 4 in Algorithm 1).
- *Convergence*. The transition from one active server set combination to another depends only on the current state and is irrelevant to previous states. Thus, it converges to a steady state (i.e., obtaining the minimum value in JOT) after a finite number of transitions.

To obtain an optimal solution in one snapshot, JOT algorithm has to solve the subproblem **P1** several times. This is still NP-hard and there is no computationally-efficient solution to solve. However, JOT problem now is reduced the complexity due to removing the variable m in the objective function. The subproblem **P1** can be cast as a the generalized assignment problem [16], which can be solved using the mathematical framework of matching theory [17].

Algorithm 1: JOT algorithm.

1. Initialization: Choose randomly a sub set $\mathcal{S}_0$ of active servers and a feasible X_0 based on (1) and (2). Then, set $m^* \leftarrow |\mathcal{S}_0|$ and $X^* \leftarrow X_0$ and compute the optimal value C^* of JOT problem.
2. Obtain X' and the corresponding optimal value $C' = C(X', m')$ by solving the following sub-problem, named as problem **P1**:

$$\begin{aligned}
\textbf{P1}: \text{minimize} \quad & \sum_{i \in \mathcal{V}} G(X), \\
\text{subject to} \quad & \sum_{i \in \mathcal{V}} r_i^p \times X_{ij} \leq r_j^p, \forall j \in \mathcal{S}', \forall p, \\
& \sum_{j \in S'} X_{ij} = 1, \forall i \in \mathcal{V}, \\
& X_{ij} = \{0,1\}, \forall i \in \mathcal{V}, \forall j \in \mathcal{S}', \\
\text{var.} \quad & \{X\}.
\end{aligned} \tag{7}$$

3. Compute the transition probability [15]

$$\mathbf{p} = \frac{\exp(\delta C')}{\exp(\delta C') + \exp(\delta C^*)}. \tag{8}$$

4. With probability p, the controller sets $m^* \leftarrow m'$, $X^* \leftarrow X'$, and $C^* \leftarrow C'$. With probability $1 - p$, the controller keeps the current state.
5. Choose a new active set $\mathcal{S}'$ that satisfies (1) and set $m' \leftarrow |\mathcal{S}'|$.
6. Return to Step 2 until the stopping criteria is met.

B. VM allocation for subproblem *P1* as a many-to-one matching game

To model the subproblem **P1** as a many-to-one matching game [17], we consider two sets $\mathcal{V}$ of VMs and $\mathcal{S}$ of servers as two teams of players. The *matching* is defined as an assignment of VMs in $\mathcal{V}$ to servers in $\mathcal{S}$. VMs decide to locate on servers that can reduce inter-communication between them. Meanwhile, servers choose VMs depending on their capacity, including many dimensions such as CPU, memory, storage, etc. In term of reducing the redundant resources, a server s_j would like to host VMs as much as possible based on its quota. Actually, real values of resource types in datacenter are heterogeneous, therefore, we define a scale method to formulate again these values.

1) Dominant resource in a task and a server: Based on the analysis in [4], a server should host VMs, which have the same dominant resource type with itself to reduce the redundant resource in allocation. For example, a VM with dominant in memory should be allocated into a server with dominant memory. We now define a scaling vector $\gamma = (\gamma_1, \gamma_2, ..., \gamma_p)$ to normalize the real size of servers and tasks. For example, considering with two types of resource, such as CPU and memory and a scaling vector $(\gamma_{CPU} = 1, \gamma_{Mem} = 2)$, one VM has a configuration, such as 10 CPU and 15 Mem, it is equivalent to 10 CPU units and 30 Mem units. Based on this scaling, we can compare which resource type is dominant in each task and each server.

2) Matching concepts: The assignment of VMs in $\mathcal{V}$ to servers in $\mathcal{S}$ can be considered as an outcome of a many-to-one matching.

Definition 1. The outcome of a VM allocation problem is a matching μ. Formally, a matching is a function $\mu : \mathcal{V} \cup \mathcal{S} \to 2^{\mathcal{V} \cup \mathcal{S}}$such that:

- $\mu(j) \subseteq \mathcal{V}$ such that $|\mu(j)| \leq r_j, \forall j \in \mathcal{S}$, where $|\mu(j)|$ is the amount of aggregation resource of all VMs that are matched to j,
- $\mu(i) \subseteq \mathcal{S}$ such that $|\mu(j)| = r_i$ or $|\mu(j)| = 0, \forall i \in \mathcal{V}, j \in \mathcal{S}$, where $|\mu(j)|$ is the amount of resource of one server that is matched to i,
- $i \in \mu(j)$ if and only if $\mu(i) = j, \forall i \in \mathcal{V}, j \in \mathcal{S}$.

The definition states that a matching is a many-to-one relation in the sense that each server is matched to a subset of VMs. Before setting an assignment of VMs to servers, each player needs to specify its preferences over the opposite set depending on its goal in the network.

Task's preference list. Each task $r, r \in \mathcal{T}$ has strict, transitive and complete preference relation $P(r)$ over the server set $\mathcal{S}$. We denote $j \succ_r j'$ as the task r prefers server j to server j' and if r prefers to remain unmatched instead of being matched to server j, i.e., $\emptyset \succ_r j, j$ is said to be unacceptable to r. One task includes a list of VMs that all of them have the same preference as its task, since all VMs in each task want to locate together in the same server.

Server's preference list. Similarly to the traditional many-to-one matching theory, we consider each server has a preference list over all VMs. This consideration is proper on the allocation scenario of the datacenter, where the controller has enough information of available resource on each server to determine which VM should be located on which server, as discussed in Section II-A. Therefore, we consider each server $j \in \mathcal{S}$ has a strictly, transitive and complete preference relation $P(j)$ over the VM set $\mathcal{V}$. We denote $i \succ_j i'$ as the server j prefers VM i to VM i' and if j prefers to remain unmatched instead of being matched to VM i, i.e., $\emptyset \succ_j i, i$ is said to be unacceptable to j. Based on [17], we design a VM allocation scenario with the definitions as follows.

In order to address the matching game, we have to find a stable solution, where there does not exist any player that is not matched to another but they are prefer to be partners.

Definition 2. A matching μ is blocked by a pair of agents (i, j) if there exists a pair (i, j) with $i \notin \mu(j)$ and $j \notin \mu(i)$ such that $i \succ_j \mu(j)$ and $j \succ_i \mu(i)$. Such a pair is called a blocking pair in general.

Definition 3. A task $r, r \in \mathcal{T}$ is saturated if all VMs in task r are assigned. Similarly, a server is saturated if all its capacity is utilized.

In our model, to satisfy the requirement from users, all demands have to serve at servers. Therefore, all stable assignments do not exist any unsaturated task.

Definition 4. A matching is stable if (i) there is no blocking pair (ii) all VMs are assigned to servers (or all tasks are saturated).

In studied system, constructing the preference list for players is an important step, because it results the matching outcome. Now, we consider concretely how to calculate the preference list of server and task that can determine the way to optimize both of them. This is also an issue in many current works [18], [19] that often relax the calculation method to build the preference lists in matching model.

3) Task's preference list: For VMs, they always prefer locating on the server which has the most available resource. Also, all VMs in a task are always interested locating in the same servers to reduce the inter-communication between them. Thus, considering servers as vertexes and links between servers as edges, we choose the largest server (i.e., it has the most available resource) depending on the dominant resource type of the task, then apply Nearest Neighbor algorithm [20] to find a set of active servers that can host all VMs of the task.

4) Server's preference list: On the server side, to improve the resource utilization on each server, the controller (i.e., OpenStack in Section II-A) only considers the configuration vector of each VM r_i and the available resource of server r_j. And one of the famous methods to pack VMs into servers is Best Fit Decreasing algorithm [21], which chooses the list of VMs to pack into server such that the remaining resource is smallest. Hence, we apply Best Fit Decreasing algorithm [21] to build the preference list for each server. It means that a server prefers to match with a VM that is the best fit with its available resources.

Considering on server j, the controller chooses a list of VMs that server j has enough resource to host. The controller then ranks all chosen VMs in decreasing order based on the dominant available resource type of server j. VMs in the ranking list will be added into the preference list of server j following the order of ranking.

*C. Multi-dimensions matching algorithm for subproblem **P1***

After formulating the VM allocation problem as a many-to-one matching game, we propose an extension of Deferred acceptance Algorithm [17] to our model with multi-dimensions of resources in datacenters. Difference from the deferred acceptance algorithm for many-to-one, our algorithm has to compare the quota of servers with the VM demands in p resource types in every phase of matching algorithm.

Algorithm 2: Multi-dimensions matching algorithm for subproblem **P1**

```
Input: $\mathcal{V}, \mathcal{S}$.
Output: Allocate all VMs in $\mathcal{V}$ to servers in $\mathcal{S}$.
while $\exists$ task $r$, who is not saturated do
    while $\exists$ VM $i, i \in \mathcal{V}$ is unassigned do
        $j = i$'s highest ranked server in $P(i)$;
        if $r_j \geq r_i$ then
            Match $j$ and $i$;
            $r_j = r_j - r_i$;
        end
        else
            Find all $i'$ matched to $j$ so far such that
            $i \succ_j i'$;
            Reject all $i'$ and set $i'$ as unassigned,
            $r(j) = r(j) + r(i)'$;
            Remove $i'$ out of $P(j)$;
            Remove $j$ out of $P(i')$;
        end
    end
end
```

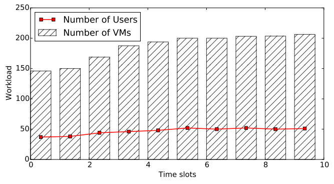
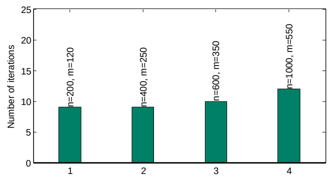


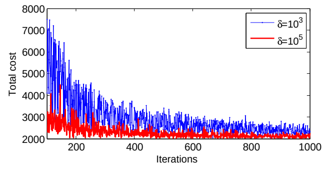
Fig. 2: Trace of VM request in 10 time slots.

Following the many-to-one matching algorithm [17], we designed the matching algorithm to iteratively find the stable matching with respect to optimize tasks. First, all VMs in each task apply to servers following their shared preference lists. Servers then accept VMs based on their preference lists and reject VMs if their quotas come to the limitation. Finally, all unassigned VMs propose to the lower ranking servers based on their preference lists. One server can reject accepted VMs, then add new VMs if new VMs have high ranking in the server's preference list. As same as Deferred acceptance Algorithm [17], when the server rejects one VM, all the accepted VMs with lower ranking are also rejected.

A pseudo-code implementation of our model is shown in the Algorithm 2. The algorithm finds an optimal solution for tasks, in the sense that every task is designed to be saturated and locate on its best server possible in his preference list. Moreover, the Algorithm 2 basically follows all steps of Deferred acceptance Algorithm in [17] that is already proven the existing of stable matching in the result.



(a) Evaluation of fast convergence of Algorithm 2 when solving sub-problem **P1** in different number of servers and VMs.



(b) Evaluation of convergence of JOT Algorithm in different values of δ.

Fig. 3: Evaluation of convergence.

IV. Simulation and Numerical Results

A. Settings

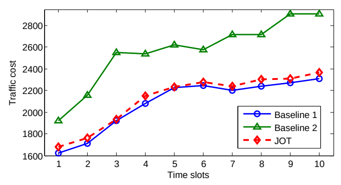
For datacenters, we consider five types of VMs whose configurations are set randomly in range from 1 to 10 for CPU unit and 5 to 20 for memory. In terms of the monetary weights, α and β are set to 0.1. The power of active server Q_i is set in range from 200W to 400W. Finally, we create randomly the distance matrix D with the value in range from 1 to 5. Furthermore, in order to illustrate the efficiency of JOT, we use the workload of one cloud system [22] in 10 time slots with heterogeneous tasks, as shown in the Fig. 2.

B. Results

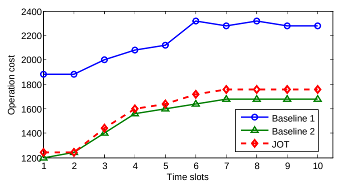
Convergence. According to settings above, we execute Algorithm 2 and Algorithm 1 to evaluate the convergence of matching algorithm and JOT convergence. The result in Fig. 3a matches with the characteristics of the matching theory, where the stable allocation can be achieved after 12 iterations for 1000 VMs and 550 servers. The advantage of stable matching algorithm also is proven by increasing the number of VMs and servers. The number of iteration does not quite sensitive with values of n and m. In practice, the number of servers and VMs in datacenters can be thousands, which require an efficient solution with low computation cost and fast convergence. Bin-packing algorithms [21] are difficult to satisfy these things. Hence, matching approach is a suitable solution that can be adapted to the practical model of heterogeneous servers and user requests in datacenters.

About JOT algorithm, we compare the convergence with different δ. As the proof in [15], JOT algorithm converges better when increasing δ, $\delta \to \infty$. In practice, this is impossible, thus, we show the effect of parameter δ by conducting in our experiment. As shown in Fig. 3b, the total cost decreases when the value of δ increases. The minimum value of total cost can be obtained around 1000 iterations with $\delta = 10^5$.

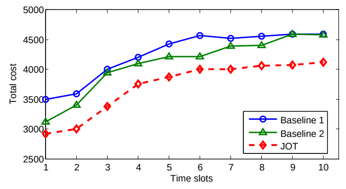
Total incurred cost. We next compare JOT algorithm with two approaches, ones is applied the bin-packing algorithm [21] to optimize only the operation cost (Baseline 1 in Fig. 4) and another one is used TSP algorithm [20] to optimize the network traffic (Baseline 2 in Fig. 4). From time slot 1 to 10, the number of users and the number of VMs are increasing. For each user, we create randomly a list of VMs demand in range 3 to 10. Optimizing the traffic cost, all VMs of each user in baseline 1 aim to occupy in the one servers or a group servers that has minimum traffic cost, as shown in Fig. 4a. Therefore, this policy spends a lot of servers, since the resource utilities of these servers (CPU, memory) are not optimized. Meanwhile, using bin-packing algorithm in baseline 2, all servers are consolidated to maximize the resource utility (CPU, memory). But baseline 1 policy ignores the inter-connection of VMs in their groups, as shown in Fig. 4b. Considering only traffic cost or operation cost, JOT cannot obtain the optimal in each case but nearly tags after the minimum value at each time slot, as shown in Fig.4a and Fig. 4b. By taking the average of 10 time slots, JOT algorithm reduces 19.1% and 9.28% of the total cost compared with the Baseline 1 and 2, respectively,



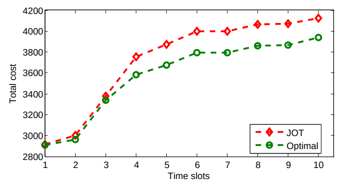
(a) Evaluation of Traffic cost



(b) Evaluation of Operation cost.



(c) Evaluation of Total cost.



(d) Comparison of Total cost between JOT and the optimal solution.

Fig. 4: Comparison between only using operation cost policy (Baseline 1), using traffic cost policy (Baseline 2), the optimal solution and JOT.

as represented in Fig. 4c. It also demonstrates that jointly controlling the operation cost and the network traffic is an efficient resource allocation method in the datacenter.

V. Conclusion

In this paper, we studied the joint operation cost and network traffic policy, JOT, which can minimize the total cost in the datacenter. We formulate JOT as the NP-hard problem and design a method, which combine the Gibbs sampling method and the many-to-one matching theory to find an optimal solution, where dynamically optimize the resource allocation in cloud datacenters. Comparison with the methods that only consider traffic cost or operation cost, our method shows the efficiency in reducing both operation and network traffic cost. To our knowledge, our proposed system can be practical and applied to the scheduling function of datacenters.

Acknowledgment

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-(H8501-15-1015) supervised by the IITP (Institute for Information & communications Technology Promotion). *Dr. CS Hong is the corresponding author.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] F. Ma, F. Liu, and Z. Liu, "Multi-objective optimization for initial virtual machine placement in cloud data center," *J. Infor. and Computational Science*, vol. 9, no. 16, 2012.

[3] S. K. Mandal and P. M. Khilar, "Efficient virtual machine placement for on-demand access to infrastructure resources in cloud computing."

[4] M. Mishra and A. Sahoo, "On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 275–282.

[5] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 41–50.

[6] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," 2013.

[7] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[8] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 66–70.

[9] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration."

[10] A. Beloglazov and R. Buyya, "Openstack neat: A framework for dynamic consolidation of virtual machines in openstack clouds–a blueprint," *Cloud Computing and Distributed Systems (CLOUDS) Laboratory*, 2012.

[11] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *arXiv preprint arXiv:1412.4980*, 2014.

[12] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "Openflow supporting inter-domain virtual machine migration," in *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*. IEEE, 2011, pp. 1–7.

[13] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[14] A. Murtazaev, S. Oh *et al.*, "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing," *IETE Technical Review*, vol. 28, no. 3, p. 212, 2011.

[15] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[16] M. Baiou and M. Balinski, "Erratum: The stable allocation (or ordinal transportation) problem," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 662–680, 2002.

[17] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American mathematical monthly*, pp. 9–15, 1962.

[18] H. Xu and B. Li, "Egalitarian stable matching for vm migration in cloud computing," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE, 2011, pp. 631–636.

[19] B. H.Xu, "Anchor: A versatile and efficient framework for resource management in the cloud," vol. 24, no. 6. IEEE, 2013, pp. 1066–1076.

[20] G. Gutin, A. Yeo, and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp," *Discrete Applied Mathematics*, vol. 117, no. 1, pp. 81–86, 2002.

[21] J. Kleinberg and É. Tardos, *Algorithm design*. Pearson Education India, 2006.

[22] Facebook dashboard. [Online]. Available: https://PrinevilleDataCenter/app399244020173259