Modelling and Optimizing Bandwidth Provision for Interacting Cloud Services

Chao Chen[1], Ligang He[1,2(✉)], Bo Gao[1], Cheng Chang[2], Kenli Li[2], and Keqin Li[2,3]

[1] Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
{chao,liganghe,bogao}@dcs.warwick.ac.uk

[2] School of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
{chengchang,lkl}@hnu.edu.cn

[3] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
lik@newpaltz.edu

Abstract. Non-deterministic communication patterns among interacting Cloud services impose a challenge in determining appropriate bandwidth provision to satisfy the communication demands. This paper aims to address this challenge and develops a Communication Input-Output (CIO) model to capture data communication produced by Cloud services. The proposed model borrows the ideas from the Leontief's Input-Output Model in economy. Based on the model, this paper develops a method to determine the bandwidth provision for individual VMs that host a service. We further develop a Communication-oriented Simulated Annealing (CSA) algorithm, which takes an initial VM-to-PM mapping as input and finds the mapping with the minimal bandwidth provision and without increasing the PM usage in the initial mapping. Experiments have been conducted to evaluate the effectiveness and efficiency of the CIO model and the CSA algorithm.

1 Introduction

Services deployed in a Cloud are often hosted in a number of virtual machines (VMs), which are then placed on Physical Machines (PMs). In a Cloud environment, when services are invoked, the service invocations are often not isolated. An invocation to a service may spawn further invocations to other services. Moreover, service invocation and consequently data communication among them may not be static, but depend on the dynamic system information or service input. Consider the following example. NASDAQ QMX, the largest stock exchange company in the world, has been developing their data analysis services on Amazon Web Services (AWS) [1]. The data analysis process may involve a collection of interacting services, which are implemented through the standard services

This research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Numbers: 61133005, 61432005).

A. Barros et al. (Eds.): ICSOC 2015, LNCS 9435, pp. 305–315, 2015.
DOI: 10.1007/978-3-662-48616-0_19

provided in AWS, such as S3, VPC, EC2, etc. Which services are involved in the data analysis workflow and their invocation order are not static, but depend on dynamic system information at runtime, such as the initial data submitted by the clients, performance or security needs of the clients, and so on.

This brings the challenge to determine the bandwidth provision for these services and more specifically for the VMs that host the services. Solving the problem of VM bandwidth provision can help the tenants equip the VMs with proper communication capacity. In EC2, different types of VM instances have different communication capacity and consequently different price rates. Moreover, the data transfer between VMs is also charged in AWS. An exemplar application of this work is that when an enterprise tenant purchases the VMs in EC2 to build a business Cloud platform, offering to its users a rich set of interacting services, this work can help the enterprise decide which type of VM instance is most appropriate for each service, so that the VMs are able to fulfil the communication requirement inherent in the business Cloud while the enterprise does not pay unnecessary extra bills for VMs with higher bandwidth.

This paper aims to address this challenge by developing a Communication Input-Output (CIO) model for data communication among services. It borrows the idea from Leontief's Input-Output model in Economy and captures the interaction relation and impact among services. The data communication performed by each service can be calculated from the model. Knowing data communication performed by a service does not necessarily mean that the solution is apparent to the problem of bandwidth provision for the service's VMs. This is because if two VMs of two communicating services are consolidated into the same PM, the data transmission between these two VMs does not consume their bandwidth. Generally, even if the bandwidth provision for the services is determined, the bandwidth provision for each individual VM still depends on the specific VM-to-PM mapping. A lot of existing work has investigated the methods to find the VM-to-PM mapping with the minimal number of PMs. However, previous work does not take into account the non-deterministic nature of service interaction when they design their consolidation strategies. Our studies found that even if the VM-to-PM mapping has the minimal number of PMs, there is still room to further reduce the communication cost in the mapping while maintaining the minimal number of PMs. This paper designs and implements a Communication-oriented Simulated-Annealing (CSA) algorithm to reduce the total bandwidth provision of all VMs in a set of interacting services. The CSA algorithm takes as input the VM-to-PM mapping with the minimal number of PMs that is generated by the existing strategies. The CSA gradually adjusts the initial VM mapping to generate new mappings with reduced bandwidth provision. The adjustment of VM mappings is designed in the way that it does not increase the number of used PMs.

2 Background and Related Work

Background of the IO Model. Leontief's input-output (IO) model [2] divides an economy into sectors (e.g. agriculture, manufacturing, etc.). Goods produced by a sector are consumed by the consumer market and other sectors. The consumer market is referred to as the *open sector*. Demands generated from the open sector are referred to as *external demands*. Goods that are exchanged between sectors is referred to as *internal demands* of the economy. Let column vectors A and X denote the external demands and the total demands of all sectors in the economy respectively, and C denote the internal consumption matrix of the economy in which c_{ij} represents the amount of goods that need to be consumed by sector i to produce one unit of goods in sector j. Leontief's IO model can be expressed by Eq. 1, which can determine the total demand vector X.

$$X = (I - C)^{-1}A. \tag{1}$$

Bandwidth Provision in Cloud. The work in [3–7] implements the techniques to enforce the minimum bandwidth allocation for each VM that is used to host specific services. However, these studies do not consider the policies to determine the appropriate bandwidth capacity for each service and its constituent VMs from a holistic perspective.

The methods proposed in the literature [3,4,8] are mostly job-oriented (or tenants-oriented), i.e., to calculate the resource allocation given the specific tasks submitted by the tenants. However, as we discussed in Sect. 1, service invocations in a service workflow may vary according to the dynamic system information, and therefore it may be difficult to know beforehand the exact execution paths of the workflows in the Cloud. The bandwidth allocation policies developed in this paper are service-oriented, which do not focus on allocating the resources for a set of specific tasks, but aim to allocate the resources for each service based on the interaction patterns among the services.

VMs-to-PMs Placement. Various methods have been developed to address the VM-to-PM mapping problem, including knapsack modelling [9], the mixed integer programming [10], genetic algorithms [11], and heuristic methods [12]. However, the work is used to tackle the placement of independent VMs (i.e., there are no communications among VMs), aiming to minimize the usage of physical machines. In this paper, we investigate the placement method for interacting VMs.

Our previous work in [13] conducted the research in the same problem domain. The work in [13] and this work are in the big scope of the same project on investigating resource management for interacting services in Clouds. Nevertheless, they focus on completely different aspects of the project. The work in [13] focuses on computing resources demanded by services, while this work focuses on the demand for communication resources. The technical contributions in [13] are not attributed to the current work in terms of both developed IO models and VM placement algorithms.

3 Modelling Bandwidth Provision

3.1 The Communication Input-Output Model

We consider a cloud system as an economy, and each service hosted on the cloud as a sector of this economy. Instead of producing goods, cloud services (sectors) produce and exchange/communicate data over the network. Whereas goods in a real economy are measured by a common currency that is recognised across different sectors, data produced by services is measured in units of bandwidth across the network infrastructure. Similar to the production of goods in an economy as described by Leontief's model, the cause for the production of data by services is also classifiable as internal and external demands.

Internal demand is the data produced by a service as a consequence of a call from another service. Given two services s_i and s_j from service economy S, we define a *consumption coefficient* c_{ij} as Eq. 2, where d_i and d_j denote the average data size produced by s_i and s_j respectively, and p_{ij} denotes the probability that one invocation of s_j causes one invocation of s_i. To understand Eq. 2, suppose s_j is able to produce one unit of data per unit of time (e.g. it is allocated with one unit of bandwidth). Since an invocation of s_j produces d_j amount of data on average, s_j can be invoked $1/d_j$ times in a unit of time, so that the allocated bandwidth (one unit) of s_j is able to transfer the amount of data produced by s_j. As a consequence, the number of invocations to s_i is then given by $(1/d_j)p_{ij}$. Therefore, the total amount of data produced by s_i can be obtained by Eq. 2. As defined by Eq. 2, c_{ij} represents the amount of data produced by service (sector) s_i for each unit of data produced by s_j in a time unit. This is in line with the definition of c_{ij} used in Leontief's model.

$$c_{ij} = \frac{1}{d_j} p_{ji} d_i \tag{2}$$

In contrast, external demand in a cloud economy is the data produced by a service due to the invocation requested by external clients. When a service s_i is at the head of a service workflow (e.g., a login service at the start of a workflow), then the number of times s_i is invoked by the clients in a time unit (which we call the arrival rate of external requests for service s_i and is denoted by λ_i), together with the average amount of data that an invocation of s_i produces (i.e., d_i), determines the amount of data that will be produced by s_i in a time unit due to the external demand. Therefore, the external data demand for s_i, denoted by a_i, can be calculated by

$$a_i = \lambda_i d_i. \tag{3}$$

This definition is also in line with the definition of external demand as defined by Leontief's model. The end clients of the cloud system who trigger service workflows can be regarded as the open sector of the cloud economy which demands data production from the services.

From these derivations, we can see that a cloud economy shares many similar properties to that of a real economy. By Eqs. 2 and 3, we are able to apply the philosophy of Leontief's IO model to a cloud setting as follows.

We denote x_i^{out} as size of data produced by s_i in a time unit in order to meet both internal and external demand (we use "out" to indicate that these are the data that need to be sent out from s_i). We can establish the relation shown in Eq. 4, where X^{out} and A are vectors of dimension $|S|$ holding the data production (x_i^{out}) and external data demands (a_i in Eq. 3) of the Cloud economy, respectively, and C is the matrix of c_{ij}. Equation 4 establishes the interdependencies within the Cloud economy in terms of data production. x_i^{out} represents the amount of data that may be transmitted over the *uplink* network interface of the PMs that service s_i is hosted in. Note that if s_i and the destination service of some data sent by s_i are located in the same PM, no uplink bandwidth of the PM needs to be consumed for transferring this part of data. In Subsect. 3.2, we will present how to handle this situation and determine the bandwidth allocation for individual VMs that collectively host service s_i.

$$X^{out} = CX^{out} + A \tag{4}$$

In addition to the economy described by Leontief's model, which only considers the amount of goods produced by each sector, we need to calculate the amount of data received by each service in our data demand IO model. This is because Leontief's model does not consider the additional cost associated with a service receiving the data through its host PM's *downlink* network interface.

Among x_i^{out} of data sent by s_i, the amount of $x_i^{out}p_{ij}$ will be sent to s_{ij}. Let c'_{ij} denote the probability that a unit of data produced by s_i is to be received by s_j. Then c'_{ij} can be calculated as $\frac{x_i^{out}p_{ij}}{x_i^{out}} = p_{ij}$. We denote x_{ij}^{out} as the size of data transmitted from s_i to s_j in a time unit and x_{ji}^{in} as the size of data received by s_j from s_i in a time unit, then we have

$$x_{ji}^{in} = x_{ij}^{out} = c'_{ij}x_i^{out}. \tag{5}$$

Additionally, we denote x_i^{in} as the size of data consumed by s_i (i.e., received from all services) in a time unit. x_i^{in} can then be calculated by Eq. 6, where X^{in} is the vector of x_i^{in} and C' is the matrix of c'_{ij}. Equation 6 establishes the relationship between data production (out) and consumption (in).

$$X^{in} = C'X^{out}. \tag{6}$$

3.2 Bandwidth Provision for VMs

From the CIO model, we can derive the amount of data that are communicated by each service. In this section, our objective is to translate this quantity into actual bandwidth provision for individual VMs hosting a service. In a cloud system, each service is hosted by a collection of VMs. We assume that the service is the only service hosted in each of the VMs. This assumption is reasonable since it is a typical setting in Clouds to host different Cloud services in different VMs so as to provide the isolated service environments.

When two VMs of a pair of services are located on the same PM, data may be transmitted locally and thus does not consume the VMs' physical bandwidth. However, in order to take advantage of this local data transmission channel, the local ratio between the numbers of VMs of two service needs to match their global ratio. This is explained in detail below.

Given a pair of services s_i and s_j from S, V_i and V_j denote the total number of VMs in the cloud for hosting these two services, respectively. Consequently, the amount of data sent from a VM^i (VM^i denotes a VM that hosts service s_i) to service j can be calculated by $\frac{x_{ij}^{out}}{V_i}$, where x_{ij}^{out} is the data sent by service i to j in a time unit, which is calculated by Eq. 4. Given a PM PM_k, v_{ik} and v_{jk} denote the number of VM^i and VM^j in PM_k, respectively. Then in PM_k, the amount of data that are communicated by VM^is to service j is $v_{ik}\frac{x_{ij}}{V_i}$. If $\frac{v_{ik}}{v_{jk}}$ (i.e., the local ratio of the number of VM^i to the number of VM^j in PM_k) is no greater than $\frac{V_i}{V_j}$ (i.e., the global ratio of the number of VM^i to the number of VM^j in the cloud), all data sent by VM^is in PM_k (the VMs that host service i in PM_k) to service j can be handled by VM^js in PM_k. Therefore, there is no need to consume the bandwidth of VM^i (or VM^j) for sending (or receiving) these data. For example, assume V_i and V_j are 20 and 50, respectively. If in PM_k, v_{ik} is 2 and v_{jk} is 6, then there are more than fair share of VM^j (which is 5) in PM_k to handle the data sent by VM^i in the same machine (since $2/6 < 20/50$).

On the contrary, if the local ratio is greater than the global ratio, which means that there are not adequate VM^j in PM_k to handle the data sent by VM^i in PM_k. The portion of data that cannot be handled by VM^j in PM_k, denoted by y_{ijk}, have to be sent by VM^i to VM^j in another PM, PM_l, and therefore consume the uplink bandwidth of VM^i and the downlink bandwidth of VM^j. y_{ijk} can be calculated by Eq. 7. Equation 7 essentially compares whether the local ratio is no greater than the global ratio. If so, y_{ijk} is 0. Otherwise, Eq. 7 calculates the data that s_i has to send out after deducting the portion of data that can be handled by VM^j in the same machine.

Since y_{ijk} is the data communicated in a time unit, y_{ijk} is essentially the bandwidth that has to be allocated to the VM^is in PM_k for sending data to service s_j. Therefore, $\frac{y_{ijk}}{v_{ik}}$ is the uplink bandwidth that has to be allocated to each VM^i in PM_k for sending the data to s_j, while $\frac{y_{ijk}}{v_{jl}}$ is the downlink bandwidth allocated to each VM^j in PM_l for receiving y_{ijk}. The total uplink bandwidth that needs to be allocated to VM^i in PM_k can be calculated by $\sum_{s_j \in PM_k} y_{ijk}$.

$$y_{ijk} = \max\{v_{ik}\frac{x_{ij}}{V_i}(1 - \frac{v_{jk}(V_i/V_j)}{v_{ik}}), 0\} \tag{7}$$

Given a VM-to-PM mapping, denoted by $\mathcal{M}$, the total uplink communication bandwidth generated by $\mathcal{M}$ can be calculated by Eq. 8, where y_{ijk} is the amount of data that are sent from VM^i (hosting service i) in PM_k (consuming the uplink bandwidth of PM_k) to VM^j (hosting service j) in other PMs. The total downlink bandwidth generated by a VM-to-PM mapping can be calculated in a similar way.

$$\mathcal{C}(\mathcal{M}) = \sum_k \sum_j \sum_i y_{ijk}. \tag{8}$$

4 The Communication-Oriented Simulated Annealing Algorithm

In the classical SA approach, an initial solution is first generated (a solution is encoded) and the neighbourhood searching routine is then applied to generate new suitable candidate solutions. A cost function and the metropolis criterion [14], which models the transition of a thermodynamic system, are used to determine the quality of the solutions and guide the searching direction so that better solutions can be gradually generated until the stopping criterion is met.

In this section, we design a Communication-oriented SA (CSA) algorithm that aims to find the VM-to-PM mapping with the minimal bandwidth provision for all VMs. In the CSA algorithm, the initial solution is set as the VM-to-PM mapping that is generated by the MinPM algorithm [9] (i.e., the algorithm that produce the VM-to-PM mapping that uses the minimal number of PMs to host VMs). The amount of bandwidth provision calculated in Eq. 8 is used as the cost function for the CSA algorithm. The CSA algorithm adjusts the VM-to-PM mapping, aiming to reduce the bandwidth provision without increasing the number of PMs. This section presents the encoding of the solution, the neighbourhood searching routine and the flow of CSA algorithm in this paper.

Encoding the Solution. In the SA algorithm, a solution is encoded as a two-dimensional array, A, in which an element $a[i][j]$ represents how many VMs of Service s_j there are in PM_i. Note that this encoding method does not differentiate the VMs for the same service. This way, the number of VMs does not affect the complexity of the algorithm. Consequently, the proposed SA algorithm can find the good VM-to-PM mappings efficiently.

Neighbourhood Searching. In SA, the design of neighbourhood searching routine is critical for generating good solutions with good efficiency. This subsection presents the method to conduct the neighbourhood searching. Two probabilities, p_p and p_s, are set to represent the possibility that the VM mapping of a service in a PM is adjusted. To improve the efficiency, the following design is adopted for the neighbourhood searching. The neighbourhood searching routine randomly selects $N \times p_p$ PMs (N is the total number of PMs) to adjust the VM mappings of some services in these PMs. For a selected PM, the routine further randomly selects $M \times p_s$ services (assume M is the number of services in the PM) and the VM mappings of these services will be adjusted. For service s_i in PM_j, its VM mapping is adjusted in the following way. First, the neighbourhood searching routine randomly selects another PM, PM_k, and then randomly selects a service, s_l ($l \neq i$), in PM_k. The routine then tries to swap the VMs between s_i and s_l. In order to render a valid swap, the routine calculates the maximum number of VMs that can be swapped between the two services, which can be calculated using Algorithm 1, where f_k and f_l are the spare resource capacity in PM_k and PM_l, respectively, v_{ik} is the number of VM^i in PM_k, $swap_{ik}$ is the maximum number of VM^i that can be swapped in PM_k. A valid swap is one after which the total capacity of every type of resource (the resource types of CPU utilization, memory and bandwidth are considered in this work) allocated to the VMs in either PM does not exceed the total physical resource capacity of the PM. This validity rule guarantees that the number of required PMs does not increase. The neighbourhood searching is presented in Algorithm 2.

As discussed above, the neighbourhood searching routine randomly selects $N \times p_p$ PMs (N is the total number of PMs) and in each selected PM, the routine

Algorithm 1. Calculating maximum number of VMs that can be swapped

```
1: if $VM_k^i \times v_{ik} < VM_l^j \times v_{jl}$ then
2:     $Swap_{ik} = v_{ik}$
3:     $Swap_{jl} = [\frac{VM_k^i \times v_{ik} + f_k}{VM_l^j}]$
4: else if $VM_k^i \times v_{ik} > VM_l^j \times v_{jl}$ then
5:     $Swap_{jl} = v_{jl}$
6:     $Swap_{jk} = [\frac{VM_l^j \times v_{jl} + f_l}{VM_k^i}]$
7: else
8:     $Swap_{ik} = v_{ik}$
9:     $Swap_{jl} = v_{jl}$
```

Algorithm 2. Neighbourhood searching

```
1: Randomly select $\lfloor p_p \times N \rfloor$ PMs
2: for each of these PM do
3:     Randomly select $p_s \times S\ |_k$ services in $PM_k$
4:     for each of services do
5:         Randomly select a PM, $PM_l (l \neq k)$ and a service $j$ $(j \neq i)$ in $PM_l$
6:         Call Algorithm 1 to calculate maximum number of VMs in $VM^i$ and $VM^j$ that can form a valid swap
7:         Swap calculated number of VMs between $s_i$ in $PM_k$ and $s_j$ in $PM_c$
8: Return new VM-to-PM mapping, $\mathcal{M}'$
```

further selects $M \times p_s$ services to adjust their VM mappings. Therefore, the time complexity of Algorithm 2 is $O(p_p \times N \times p_s \times M)$.

Simulated Annealing. Algorithm 3 outlines the entire SA process aiming to find the optimal VM-to-PM allocation. In the algorithm, T is the initial temperature of the SA process, which is typically set as 1000. $factor$ is the cool-down factor of the SA process, which is typically set as 0.85. In each iteration, $\mathcal{M}$ is the current VM-to-PM mapping. Algorithm 2 is called to generate a new candidate VM-to-PM mapping, $\mathcal{M}'$ (Line 4). Equation 8 is then applied to calculate the communication cost ($\mathcal{C}'(\mathcal{M}')$) of the new mapping $\mathcal{M}'$ (line 5). If $\mathcal{C}'(\mathcal{M}')$ is better(smaller) than that of the current mapping, the algorithm accepts the new mapping and the new mapping becomes the current mapping (Line 6–8). Otherwise, the metropolis criterion, calculated by $\exp(\frac{-\Delta\mathcal{C}(\mathcal{M})}{T})$, is used to decide whether this new but worse VM mapping should be accepted. If the calculated metropolis criterion is greater than a float number randomly generated between 0 and 1 (Line 7), $\mathcal{M}'$ is accepted. Otherwise, the current mapping remains intact. The iteration repeats until the current mapping stays unchanged for a certain number of consecutive iterations (counted by j) or the number of iterations (counted by i) reaches a pre-set number, k_{max1} and k_{max2} in the CSA (Line 2).

There are at most k_{max2} iterations in the "while" loop in Algorithm 3. In each iteration, calling Algorithm 2 dominates the time spent in an iteration. Therefore, the time complexity of Algorithm 3 is $O(k_{max2} p_p N p_s M)$.

5 Performance Evaluation

We have conducted the simulation experiments to evaluate the effectiveness of the CIO model and the CSA algorithm developed in this work.

The synthetic trace is generated in the simulation experiments. A set of 500 services are generated. A service is defined as the start service, from which all workflows in the trace start. Another service is defined as the end service, which

Algorithm 3. The CSA Algorithm

```
Require: $\mathcal{M}$
 1: $i = 0$, $j = 0$
 2: while $j \leq k_{max1}$ or $i \leq k_{max2}$ do
 3:     $T \leftarrow T \times factor$
 4:     $\mathcal{M}' \leftarrow$ Call Algorithm 2
 5:     $C'(\mathcal{M}') \leftarrow$ Call Eq. 8
 6:     $\Delta C(\mathcal{M}) \leftarrow C'(\mathcal{M}') - C(\mathcal{M})$
 7:     if $\Delta \mathcal{C}(\mathcal{M}) < 0$ or $\exp(\frac{-\Delta \mathcal{C}(\mathcal{M})}{T}) > \boldsymbol{R}(0,1)$ then
 8:         $\mathcal{M} \leftarrow \mathcal{M}'$
 9:         $j = 0$
10:     else
11:         $j = j + 1$
12:     $i = i + 1$
```

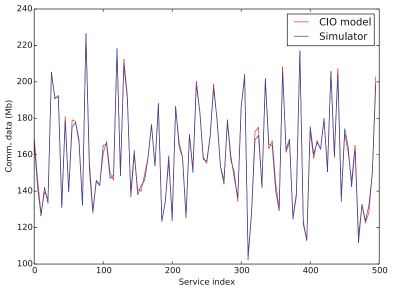

Fig. 1. Accuracy of the CIO model using synthetic traces

means that when the workflow reaches to this service, it will not invoke further services. The degree of parallelism (denoted by DP) is set, which is 3 by default, when generating the workflow instances for the synthetic trace. For all services except the end service, after a service (e.g., s_i) invoked by a task is completed, it further randomly invokes DP (e.g., 3) services. The roulette wheel method is used to randomly determine which DP services are selected based on p_{ij}. In the synthetic trace, the value of p_{ij} is randomly set from the range of [0.001, 0.003] with the average of 0.002 (i.e., 1/500, where 500 is the number of services generated in the trace). The workflow instance stops growing when all branches in the workflow reach the end service. The technique presented in [13] is used to calculate the number of VMs for each service. The strategy presented in [9] is used to generate the initial VM-to-PM mapping with the minimal number of PMs.

5.1 Accuracy of the CIO Model

It is straightforward to determine p_{ij} for the synthetic trace since a service randomly invokes another service. With p_{ij}, we apply the bandwidth IO model to calculate the bandwidth allocated for each service. In the simulator developed in this work, we allocate the calculated bandwidth to the services and then run the simulation experiments. We record the amount of data that are communicated by each service. If the proposed bandwidth IO model is effective, then the amount of data that are communicated by each service in a time unit in the simulation experiment should equal to the bandwidth allocated to each service. The results are shown in Fig. 1. The average percentage of discrepancy between the CIO model and simulation experiments is 1.3 %, which suggests that the CIO model is able to capture the bandwidth demands accurately.

5.2 The Effectiveness of CSA

The experiments in this subsection investigate the effectiveness of the CSA algorithm. In the experiments, we first used the methods proposed in [9], which we call the MinPM algorithm in this paper, to obtain the VM-to-PM mapping that uses the minimal number of PMs to host the VMs. We then apply the proposed SA algorithm to further adjust the VM-to-PM mapping in order to reduce the communication cost without increasing the number of PMs. We also used the greedy method presented in [15] to perform the VM-to-PM mapping and compared the results against those generated by the proposed SA. In the greedy algorithm, all services are ranked in the decreasing order of their communication intensity (i.e., the data that have to be communicated by a service in this paper). The greedy algorithm first place the VMs of the first service (i.e., the one with most communication intensity) on PMs, with each PM having the same number of VMs or having at most ±1 difference if it can not be evenly divided). Then the greedy algorithm selects the next service, s_2, and tries to place its VMs to PMs so that the local ratio of the number of VMs of s_1 to that of s_2 in a PM equal (or is the closest) to the global ratio of the total number of VMs of s_1 to that of s_2. The procedure repeats until all VMs are mapped.

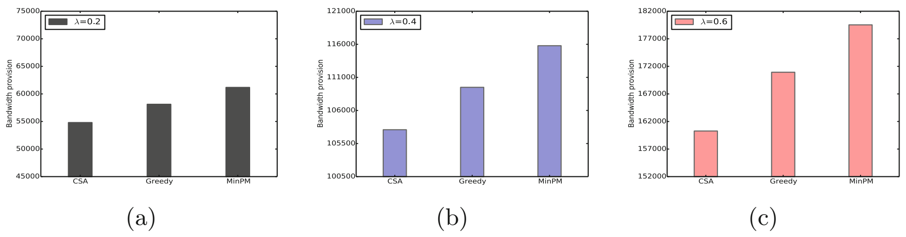

Fig. 2. Comparing CSA with other existing algorithms using synthetic trace

We increase the arrival rate of the generated workflows and use the technique presented in [13] to calculate the number of VMs for each service under different arrival rates. The experimental results are shown in Fig. 2(a, b and c). It can be seen that CSA outperforms other two algorithms in all cases.

References

1. Amazon case study: Nasaq OMX. http://goo.gl/28wfGV
2. Leontief, W.: Input-output analysis. New Palgrave Dictionary of Economics (1987)
3. Jalaparti, V., Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Bridging the tenant-provider gap in cloud services. In: ACM SOCC (2012)
4. Meng, X., Pappas, V., Zhang, L.: Improving the scalability of data center networks with traffic-aware virtual machine placement. In: IEEE INFOCOM (2010)
5. Popa, L., Kumar, G., Chowdhury, M., Krishnam. A., Ratnas, S., Stoica, I.: Faircloud: sharing the network in cloud computing. In: ACM SIGCOMM (2012)

6. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. In: ACM SIGCOMM (2011)
7. Ballani, H., Jang, K., et al.: Chatty tenants and the cloud network sharing problem. In: Proceedings of NSDI2013 (2013)
8. Jiang, J.W., Lan, T., et al.: Joint VM placement and routing for data center traffic engineering. In: IEEE INFOCOM (2012)
9. Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: 2009 ACM SIGPLAN/SIGOPS (2009)
10. Petrucci, V., et al.: A dynamic optimization model for power and performance management of virtualized clusters. In: Proceedings of e-Energy 2010 (2010)
11. He, L., Zou, D., et al.: Developing resource consolidation frameworks for moldable virtual machines in clouds. Future Gener. Comput. Syst. **32**(1), 69–81 (2013)
12. Hu, L., Jin, H., Liao, X., Xiong, X., Liu, H.: Magnet: a novel scheduling policy for power reduction in cluster with virtual machines. In: Cluster (2008)
13. Chen, C., He, L., Chen, H., Sun, J., Gao, B., Jarvis, S.: Developing communication-aware service placement frameworks in the cloud economy. In: Cluster (2013)
14. Van Laarhoven, P.J., Aarts, E.H.: Simulated Annealing. Springer, Heidelberg (1987)
15. He, L., Jarvis, S.A., Spooner, D.P., Jiang, H., Dill, D.N., Nudd, G.R.: Allocating non-real-time and soft real-time jobs in multiclusters. In: TPDS (2006)