# Developing Communication-aware Service Placement Frameworks in the Cloud Economy

Chao Chen[1], Ligang He[1], Hao Chen[2], Jianhua Sun[2], Bo Gao[1], Stephen A. Jarvis[1]
1. Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom
2. School of Information Science and Engineering, Hunan University, Changsha, 410082, China
Email: liganghe@dcs.warwick.ac.uk

***Abstract*—In a Cloud system, a number of services are often deployed with each service being hosted by a collection of Virtual Machines (VM). The services may interact with each other and the interaction patterns may be dynamic, varying according to the system information at runtime. These impose a challenge in determining the amount of resources required to deliver a desired level of QoS for each service. In this paper, we present a method to determine the sufficient number of VMs for the interacting Cloud services. The proposed method borrows the ideas from the Leontief Open Production Model in economy. Further, this paper develops a communication-aware strategy to place the VMs to Physical Machines (PM), aiming to minimize the communication costs incurred by the service interactions. The developed communication-aware placement strategy is formalized in a way that it does not need to the specific communication pattern between individual VMs. A genetic algorithm is developed to find a VM-to-PM placement with low communication costs. Simulation experiments have been conducted to evaluate the performance of the developed communication-aware placement framework. The results show that compared with the placement framework aiming to use the minimal number of PMs to host VMs, the proposed communication-aware framework is able to reduce the communication cost significantly with only a very little increase in the PM usage.**

## I. Introduction

Numerous Cloud services may be deployed in a Cloud system. These services are often not isolated. After a client invokes a service (which is the external requests), the service may request further actions from other services in the Cloud system (which is the internal requests) during or after its execution. The interactions among the services, which are demanded by the mix of the external and internal requests, are not static and may vary according to dynamic system information at runtime. These impose a challenge in determining the amount of resources required for each of these services, in order to deliver a desired level of Quality-of-Service (QoS).

The services in a Cloud system are typically hosted in VMs. Therefore, determining the suitable resource quantity for the services comes down to determining the resource capacities allocated to the VMs that host the services. There are some existing works building the performance model for the processing capacity of a VM , i.e., establishing the relation between a VM's processing capability and the amount of the resource capacities (such as CPU percentage, memory size, network bandwidth) allocated to the VM [1] [2]. For example, Amazon EC2 offers small, medium, large and extra large VMs [3]. The performance model established in the existing work can calculate the processing capability of these different types of VMs for a type of tasks.

The work in this paper makes use of the performance model of a VM established in the literature. Assuming that the processing capability of one VM is known, this paper presents a method to determine the sufficient number of VMs for interacting services in a Cloud system. The proposed method borrows the ideas from the Leontief Input-Output Model in economy (called the IO model in this paper). The input-output model conducts the input-output analysis for different industry sectors in an economy. It is able to capture the consumption relations among different sectors and calculate the equilibrium level of production for each sector, so as to satisfy both external demands from the open sector (e.g., people) and internal demands due to the consumptions relations among individual sectors in an economy. Moreover, the IO model is able to analyze the impact of the increase in the external demand for a particular sector on the production of all sectors in the economy.

The behaviors of the interacting services in a Cloud system bear the similarity with the behaviours of different industry sectors in an economy. A service supplies resources, which are consumed by clients and also by other services due to service interactions. To the best of our knowledge, this paper is the first one in literature that applies the IO model in economy to formalize and solve the resource demand problem in Clouds.

Further, when the services interact with each other, data may be communicated between them. If the VMs that host the services with frequent communications among themselves can be placed to the same Physical Machine (PM), the communication cost could be significantly reduced.

There is the existing work in literature investigating the VM-to-PM placement problem [4] [5] [6] [7] [8]. However,the existing work either focuses on consolidating the independent VMs (i.e., there are no interaction between VMs) into resources, i.e., finding a VM-to-PM placement that can minimize the number of PMs used to host the VMs, or requires to know the specific communication patterns between individual VMs. Different methods have been developed to model such a VM-to-PM placement problem. For example, the placement problem has also been modelled as a knapsack problem [5], an ant colony optimization problem [7], a mixed integer

∗ Dr. Ligang He is the correspondence author

978-1-4799-0898-1/13/$31.00 ©2013 IEEE

programming problem [9] and a genetic algorithm [6]. Then the existing solvers and the bespoke methods were developed to solve the objective functions for the optimized placement solutions. Heuristic algorithms have also been developed to find the placement solutions [10].

This paper develops a communication-aware strategy to place the VMs that host the interacting services on physical machines, aiming to minimize the communication costs incurred by the service interactions. A genetic algorithm is then developed to find a VM-to-PM placement with significantly reduced communication costs. In a nutshell, the main differences between our work and the work in literature are that 1) this work aims to find a placement solution to minimize the communication costs among services and 2) the approach adopted in this work does not need to know the specific communication pattern between individual VMs. The more detailed differences are discussed in II-B.

We have also conducted experiments to compare the framework proposed in this paper with two existing placement methods: one striving to use the minimal number of PMs to host VMs, and the other applying the heuristic approach to placing VMs. Our experimental results show that the proposed communication-aware framework significantly outperforms the heuristic approach in terms of both communication cost and the number of used PMs, and that comparing with the method aiming to achieve the minimal number of used PMs, our communication-aware approach is able to significantly reduce the communication overhead in the Cloud with only a tiny fraction of increase in resource usage.

The remainder of this paper is organized as follows. Section II-B briefly introduces the background knowledge of the input-output model, and then presents the related work. Section III presents the workload and system models. Section IV proposes the method of modelling resource demands of services in a Cloud economy. The communication-aware VM placement framework is presented in Section V. Section VI evaluates the effectiveness of the proposed framework. Finally, we concluded in Section VII.

## II. Background and related work

### A. Background of the IO model

In the IO model, the economy is divided into sectors. Each sector produces goods except for the open sector, which only consumes goods. When a sector produces goods, it needs to consume the goods produced by other sectors. The consumption matrix $C$ captures the consumption relations among sectors. An element $c_{ij}$ in $C$ represents the amount of goods produced by sector *i* that have to be consumed by sector *j* in order for sector *j* to produce one unit (e.g., in terms of US dollars) of goods. The consumption matrix $C$ represents the internal demands. Assume the column vector $D$ contains the goods demand from the open sector, which represents the external demand. The element $d_i$ in *D* represents the amount of goods from sector *i* required by the open sector. Let the column vector $X$ be the equilibrium levels of production output that can satisfy both internal and external demands in the economy. The element $x_i$ in $X$ represents the equilibrium level of output by sector $i$. $X$ must satisfy Eq.1, which may be solved for $X$ by transforming it to Eq.2.

$$X = CX + D \tag{1}$$

$$X = (I - C)^{-1}D \tag{2}$$

### B. Related work

The related work of the following three aspects is discussed in this subsection. In Subsection II-B1, we discuss the related work about resource allocation for VMs. Then Subsection II-B2 discusses the existing techniques to address the VM-to-PM placement. Finally, as introduced in Subsection II-A, the IO model needs to know the consumption matrix $C$. In the modelling method proposed in this work, the consumption matrix corresponds to the interaction relation among the services in the Cloud. If the interactions among services are static, it is straightforward to determine the interaction relation (i.e., calculate the elements, $c_ij$, in $C$. However, as we discussed in Section I, the service interactions may be dynamic, depending on the system information at runtime. An approach to determining the interaction relation among services is to analyze the invocation trace of each individual service in the Cloud. In Subsection II-B3, we discuss the existing techniques regarding this.

*1) Resource allocation for VMs:* Various methodologies have been proposed to construct the performance model, i.e., to establish the relation between the performance of a VM (e.g., throughput, the time needed to complete a request) and the resource capability allocated to the VM (e.g., CPU, memory, network bandwidth) [1] [2] [11]. For instance, the work in [2] used layered queuing network to model the response time of a request in a multi-tiered web service hosted in VM environments, while hardware resources (e.g., CPU and disk) are modeled as processor sharing (PS) queues. The work in [1] modeled the contention of visible resources (e.g., CPU, memory, I/O) and invisible resources (e.g., shared cache, shared memory bandwidth) as well as the overheads of the VM hypervisor implementation. The work in [11] and [12] then investigates the impact of the network bandwidth allocated to a set of VMs on their collective performance. In this paper, we assume that the performance model is already know. In the simulation experiments, we uses the queuing theory as the exemplar technique to derive the performance model.

There is also the work addressing resource allocation for a group of VMs with communications among them [13] [4]. The work in [13] can translate the performance goals of the tasks submitted by clients to the resource allocation in terms of the combination of the number of VMs and the network bandwidth between the VMs. Since different resource combinations may produce similar performance, the work further propose a method to select the resource combination that can balance the resource utilization. The work in [13] bear similarity with the work presented in this work. The work in [13] also calculates the number of VMs required to meet performance goals. The difference is that the work in [13] is job-oriented (or client-oriented), i.e,, to calculate the resource allocation given the specific tasks submitted by the clients. However, as we discussed in Section I, service invocations (i.e., the tasks) may vary according to the dynamic system information, and it may be difficult to know the full picture of

the tasks/workflows to be run in the Cloud. The work in this paper is service-oriented, which does not focus on allocating resources for a set of specific tasks or workflows, but aims to allocate resources based on the service interaction patterns. This work does not even have to know the full information of the tasks/workflows to be run.

*2) VM-to-PM placement:* The VM-to-PM placement problem mainly aims to consolidate resources and improve resource utilization [5] [10]. Various methods have been proposed in literature to address the VM-to-PM placement problem, including the knapsack modelling [5], the mixed integer programming [9], the genetic algorithms [6], the ant colony optimization [7] and the heuristic methods [10]. For example, the work in [10] develops the heuristic squeeze and release measures to dynamically redistribute the workloads in the cluster according to the workload level in each individual node, so as to minimize the usage of physical machines. The work in [5] develops a server consolidation scheme, called Entropy. Entropy strives to find the minimal number of nodes that can host a set of VMs, given the physical configurations of the system and resource requirements of the VMs. The objective is formalized as a multiple knapsack problem, which is then solved using a dynamic programming approach. In the implementation, a one-minute time window is set for the knapsack problem solver to find the solution. The solution obtained at the end of the one-minute time space is the new state (new VM-to-PM placement). The work in [6] designed a genetic algorithm to find a VM-to-PM placement that uses the minimal number of PMs. However, the above 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 of the interacting VMs and focus on finding the VM placement that can minimize the communication cost.

There is also the work tackling the placement problem for the VMs with inter-VM communications, aiming to minimize the communication costs [4] [8]. The work in [4] model such a VM-to-PM placement as a min-cost network flow problem and then use the Breadth First Search to find the optimal placement solution. The work in [8] then uses the classical min-cut graph algorithm to obtain the optimized placement solution. In order to model the placement problem as the min-cost network flow problem or the min-cut graph problem, they need to know the specific communication pattern between each pair of VMs. As we have discussed, in some cases, we may not know the full picture of the submitted workload, and therefore can not accurately determine the communication pattern between each individual VM. In this paper, we model the interactions among the services, and treat a service (and the set of VMs supporting the service) as a whole without the need to know the specific communication pattern between each pair of VMs.

*3) Anaysis of invocation pattern:* There are the existing techniques to obtain the invocation pattern of the services [14] [15] [16]. The work in [14] implements a multi-level probabilistic model to infer the probability of a service calling another service. The fundamental idea is to monitor the packets sent and received by a service, and then compute the dependency probability between the services by leveraging the observation that if accessing service B depends on service A, then packets exchanged with A and B are likely to co-occur.

TABLE I. NOTATIONS

| notations | Explanation |
|---|---|
| $C$ | consumption matrix |
| $c_{ij}$ | the amount of goods produced by sector *i* that have to be consumed by sector *j* in order for sector *j* to produce one unit of goods |
| $S$ | the number of services |
| $s_i$ | service *i* |
| $e_{ij}$ | the amount of data that are sent when service $s_i$ invokes $s_j$ |
| $p_{ij}$ | the probability that one invocation of $s_i$ causes the invocation of $s_j$ |
| $n_i$ | physical machine *i* |
| $VM^i$ | a virtual machine hosting service $s_i$ |
| $v_{ik}$ | the number of $VM^i$s in $n_i$ |
| $p_{ji}$ | the possibility that executing $s_j$ causes a further invocation of $s_i$ |

The work in [15] then uses the k-means clustering technique in data mining area to analyze the service trace and calculate the correlation probability between services.

## III. WORKLOAD AND SYSTEM MODELS

$\mathbb{S} = \{s_1, \ldots, s_M\}$ denotes a set of *M* services deployed in a Cloud. $\lambda_i$ denotes the arrival rate of the requests directly from the clients for service $s_i$. $p_{ij}$ denotes the probability that after service $s_i$ is invoked and executed, service $s_i$ will further call service $s_j$. A service is hosted in a set of VMs (i.e., a virtual cluster). Assume each VM that hosts the same service (i.e., each VM in a virtual cluster) is allocated with the same resource capacity (e.g, the proportion of CPU, memory size, etc). This assumption is reasonable because this is the normal practice when using a virtual cluster to host a service [11]. $VM^i$ denotes a VM that hosts service $s_i$. There may be multiple VMs in a PM. We assume that PMs and network links are homogeneous, i.e, the PMs and the network links connecting any two PMs in the Cloud has the same performance. This assumption is reasonable since homogeneous machines and communication networks are typically used to construct a Cloud system.

Given the arrival rate of the requests for service $s_i$ and given $VM^i$'s resource capacity, there are a number of existing techniques in literature [1] and [2] to calculate the adequate number of $VM^i$s that can satisfy the desired QoS in terms of a particular performance metric (e.g., average waiting time of the requests, throughput).

Table.I lists the notations used in the paper.

## IV. MODELLING RESOURCE DEMANDS OF CLOUD SERVICES

This section applies the IO model to formalize and calculate the equilibrium level of resource capacity demanded by the external clients and the interacting services in a Cloud economy. The constructed model is called the Cloud-IO model in this paper. In order to apply the IO model to formalize a Cloud economy, we have to use the entities in the IO model (i.e., sector and goods) to represent the entities in Cloud environments, such as service, request, VM, resource, etc.

In this paper, a service in the Cloud economy is regarded as a sector in the IO model while the external clients are regarded as the open sector, which is straightforward. However, the challenge is to identify the entity in the Cloud economy that is suitable to be regarded as goods, and also determine the consumption relations among services. We first attempted a straightforward option and use the requests sent by the

clients or the services to represent goods. This option seems to be intuitive, because a service processes (consumes) requests from clients and other services, and also generates (produces) requests to invoke other services. Then the problem comes down to how to determine the resource capacity for services so that the requests can be processed in a way that the desired QoS can be met. However, we later realize that it is not appropriate to treat the requests as goods. This is because the requests generated by services are not going to be consumed by the clients while the goods produced in the IO model are consumed by the open sector. In this paper, a group of VMs hosting a service are regarded as goods produced by the service.

Now we present how to determine the consumption relations among services, i.e, obtain the consumption matrix. Note that $c_{ij}$ in the consumption matrix $C$ represent the amount of goods produced by sector *i* that have to be consumed by sector *j* in order for sector *j* to produce one unit of goods (e.g., in terms of US dollars). Consider one VM (a unit of good) of service $s_j$. $\psi_j$ denotes the arrival rate of the requests that one VM of service $s_j$ (i.e., one $VM^j$) can handle to deliver the specified QoS. As discussed in Section III, there are existing techniques to calculate $\psi_j$, given the resource capacity allocated to the VM. We use a function $f$ to represent such a technique, i.e., Eq.3, where the first parameter represents service index (i.e., $s_j$), the second parameter $R_j$ represents the resource capacity allocated to each VM of $s_j$ (we assume every VM in the same service has the same resource capacity), and the third parameter represent the number of VMs of the service.

$$\psi_j = f(j, R_j, 1) \tag{3}$$

Every time service $s_j$ is invoked, there is the possibility of $p_{ji}$ that $s_j$ will send a request to further invoke $s_i$. Therefore, in a time unit one $VM^j$ sends $\psi_j \times p_{ji}$ requests to $s_i$. The number of VMs that need to be produced by $s_i$ to handle the requests with the arrival rate of $\psi_j \times p_{ji}$ is then equivalent to the goods produced by service $s_i$ that have to be consumed by service $s_j$ in order for $s_j$ to produce one unit of goods (i.e., one VM), which is actually $c_{ij}$ in the IO model. Again, the existing techniques in literature can calculate $c_{ij}$ based on the arrival rate of $\psi_j \times p_{ji}$ and the given resource capacity allocated to each $VM^i$. We use a function *g* to represent such a technique, i.e., Eq.4, where the first and second parameters have the same meanings as those in Eq.3, and the third parameter represents the arrival rate of the requests.

$$c_{ij} = g(i, R_i, \psi_j \times p_{ji}) \tag{4}$$

In doing so, we have established the consumption matrix in the Cloud-IO model. Let $\lambda_i$ be the rate at which the clients (open sector) send the requests to service $s_i$. Then we can use the $g$ function in Eq.4 to calculate the number of $VM^i$ that have to be produced by $s_i$ to process the requests with the arrival rate of $\lambda_i$, which is $d_i$ in the column vector $D$ in the IO model. Namely, $d_i$ can be obtained using Eq.5.

$$d_i = g(i, R, \lambda_i) \tag{5}$$

By doing so, the external demand vector *D* is obtained. $X = [x_1, ..., x_i, ..., x_M]^T$ denotes the column vector that represents the number of VMs required for each of *M* services in the Cloud economy. $X$ can be calculated by Eq. 2.

## V. The communication-aware VM placement

Section IV calculates the number of VMs required for each service in the Cloud. This section investigate the issues of mapping all the VMs obtained in Section IV to PMs. The VM-to-PM mapping in literature often focuses on minimizing the number of PMs used to accommodate the VMs, so as to minimize the resource and/or energy consumption. However, in this paper, there is the possibility that after a service is run, it may send a request to another service for further actions. Some data may be sent along with the request. If the VMs that host the different services with frequent communications can be placed in the same PM, then the communication cost could be reduced. This section develops a framework to find the VM-to-PM mapping that minimizes the communication cost in the Cloud.

According to the Cloud-IO model, $c_{ij}$ represents the number of VMs that need to be produced by $s_i$ to handle the requests send by one VM in $s_j$. Therefore, if the ratio of the number of $VM^i$s to the number of $VM^j$ in PM $n_k$, denoted as $\alpha_{ijk}$, is no less than to $c_{ij}$, then the requests (along with the data) sent by the $VM^j$s can be handled by the $VM^i$s in the same PM without breaching the QoS of $s_i$, and therefore eliminates the necessity to send the requests and data to the $VM^i$ in a different PM. On the contrary, if $\alpha_{ijk}$ is less than $c_{ij}$, then a proportion of the requests sent by the $VM^j$s in $n_k$ have to be processed by $VM^i$s in a different PM. The greater the difference between $c_{ij}$ and $\alpha_{ijk}$ is, a larger proportion of the requests and data sent by $VM^j$s in $n_k$ have to be sent out of $n_k$ and therefore a higher communication cost in the Cloud. The communication-aware service placement framework developed in this paper is based on this insight and aims to find a VM-to-PM mapping with the minimal communication cost in the Cloud.

### A. Formalizing the problem

This section models the total communication cost incurred by an arbitrary VM-to-PM mapping in the Cloud. As discussed above, when $\alpha_{ijk}$ is less than $c_{ij}$, the communication will occur between $n_k$ and another PM where there are $VM^i$. $v_{ik}$ denotes the number of $VM^i$s in $n_k$, given a VM-to-PM mapping $\mathcal{M}$. The communication cost incurred by the mapping $\mathcal{M}$, denoted as $\mathcal{C}(\mathcal{M})$, can be calculated by Eq. 6 and Eq.7. In Eq.7, the term $(f(j, R_j, v_{jk}) \times p_{ji} - f(i, R_i, v_{ik}))$ calculates that the amount of requests that are sent from $s_j$ in PM $n_k$ to $s_i$ in a time unit, but cannot be handled by $VM^i$s in $n_k$ (if $\alpha_{ijk} < c_{ij}$) in order to maintain the QoS. Therefore, these requests have to be sent to be processed by $VM^i$s in a different PM. The number of these requests times $e_j i$ is then the total amount of data that have to be communicated in the Cloud caused by the inadequate resource capacity of $s_i$ in PM $n_k$ comparing with that of $s_j$ in the same PM. Since we assume that the communication network in the Cloud is homogeneous, we do not have to consider which PM these data will be sent to. The communication cost is then the sum of all these data

that have to be sent out of the local PM by any service in the Cloud, which is Eq.6.

$$\mathcal{C}(\mathcal{M}) = \sum_{k=1}^{N}\sum_{j=1}^{M}\sum_{i=1}^{M}\beta_{ijk} \tag{6}$$

$$\beta_{ijk} = \begin{cases} e_{ji} \times (f(j, R_j, v_{jk}) \times p_{ji} - f(i, R_i, v_{ik})) & \texttt{if } \alpha_{ijk} < c_{ij} \\ 0 & \texttt{otherwise} \end{cases} \tag{7}$$

The objective is to find a VM-to-PM mapping such that $\mathcal{C}(\mathcal{M})$ is minimized, subject to certain constraints. This can be formalized as Eq. 8, where $x_i$ is the number of $VM^i$s obtained in Section IV.

$$\begin{aligned} \texttt{miminize} \quad & \mathcal{C}(\mathcal{M}), \\ \texttt{subject to:} \quad & \forall i : 1 \le i \le M, \sum_{k=1}^{N} v_{ik} = x_i \\ & v_{ik} \ge 0 \end{aligned} \tag{8}$$

### B. Designing the genetic algorithm

A Genetic Algorithm, called CAGA (Communication-Aware Genetic Algorithm), is developed in this paper. CAGA tries to find the optimal mapping with the least communication cost. In a typical Genetic Algorithm (GA), a solution is encoded and then the crossover and mutation operations are applied to evolve the solutions. Moreover, a fitness function is used to judge the quality of the solutions and guide their evolution direction so that better solutions can be gradually generated over generations. In the GA developed in this paper, the communication cost defined in Eq.6 is used as the fitness function. This section mainly presents the encoding of the solution, the crossover and the mutation operations designed in our GA.

*1) Encoding the solution and fitness function.:* In CAGA, a solution is a VM-to-PM mapping. It is encoded as an one-dimensional array, denoted as *A*. An element $a_i$ in *A* holds the index of a VM. $B_r$ denotes the capacity of the *r*-th type of resources in a PM. Given an encoded solution, the PM that a VM is mapped to is determined in the following way. Starting from the first element in the solution, the VMs are placed into $PM_1$ in the order of their positions in *A*, until the total capacity of the VMs starts to exceed the capacity of $PM_1$. The VMs are then placed into the next PM. Formally, if the first *k* PMs have been fully occupied and the VM in $a_i$ (i.e., $VM_{a_i}$) is the first VM that cannot be placed into $PM_k$ any more, the VMs in the positions from $a_i$ to $a_{j-1}$ should be placed into $PM_{k+1}$. $j$ can be determined using Eq. 9, in which $b_r(a_u)$ is the capacity of the $r$-th type of resource allocated to the VM with the index of $a_u$. For each of *R* types of resource in consideration, Eq. 9 obtain the least $j_r$ such that the total capacity of that resource of the VMs from $a_i$ to $a_{j_r}$ begins to exceed $B_r$. Then *j* is the minimum number among $j_r$ ($1 \le r \le R$). The procedure repeats until all VMs have been placed into PMs. By doing so, CAGA knows which PM a VM is placed into.

$$\begin{aligned} & j = min\{j_r | 1 \le r \le R\} \\ & \texttt{subject to:} \sum_{u=i}^{j_r} b_r(a_u) > B_r \end{aligned} \tag{9}$$

In the encoding, CAGA starts to place a VM to a new PM only when the current PM does not have enough remaining capacity to host the VM. Therefore, the method used by CAGA to encode and calculate the VM-to-PM mapping will not generate excessive spare capacity in PMs, and therefore reduce the number of PMs used to host VMs. Indeed, our experiments show that the number of PMs used by CAGA is very close to that obtained by the VM-to-PM mapping method aiming to use the minimal number of PMs to host VMs.

CAGA aims to find a VM-to-PM mapping with minimal communication cost. Therefore Eq. 6 that calculates the communication cost of a mapping is used as the fitness function of a solution.

*2) Selecting solutions.:* In GA, the solutions need to be selected from the current generation of solutions to perform the crossover and the mutation operations. CAGA applies the tournament method [17] to select the solutions used to generate next generation of solutions. The tournament method is as follows. Assume there are *h* solutions in one generation. Each time, CAGA randomly selects *k* solution (*k* is called tournament size) from the current generation. Then CAGA takes the one with the lowest communication cost among these *k* solutions and uses it as one parent solution in the crossover operation. The same way is used to obtain the other parent solution. Then the crossover operation, which is presented in subsection V-B3.C, is performed over the two parent solutions to generate two child solutions. The procedure repeats until there are *h* solutions in the next generation.

*3) Crossover and mutation:* The two-point crossover is used in CAGA. In the crossover, two points are randomly selected for two parent solutions to divide each parent into three portions. All VMs in the middle portion are swapped between the parent solutions. The resulting two solutions are children solutions in the new generation. But such a swap may cause repetitive VMs in a child solution, i.e., there may be two VMs with the same index in one solution. In order to eliminate such repetitive VMs, the swapping action is performed in the following way in CAGA. At position *i* in the middle portion of both parents, $a1_i$ and $a2_i$ are the indexes of VMs in parent 1 and parent 2, respectively. In parent 1, the crossover operation finds the VM with the index of $a2_i$ and swap $a1_i$ and $a2_i$. In parent 2, similarly, the crossover operation finds the VM with the index of $a1_i$ and swap $a2_i$ and $a1_i$. Such swapping is performed at every position in the middle portion of two parents. By doing so, we effectively swap the middle portions between parents, and the resulting children solutions will not have the repetitive VMs.

After crossover, the mutation operation is performed on the two newly generated child solutions. A mutation probability $\delta$ is set. For each VM in a child solution, there is the probability of $\delta$ that the VM will swap the positions with another randomly selected VM in the child solution. The mutated child solutions become the solutions in the new generation.

## VI. Performance Evaluation

We have conducted simulation experiments to evaluate the performance of the proposed communication-aware framework. A pool of *S* Cloud services are assumed in a Cloud. In the simulation experiments of this work, the workflows are generated to simulate the interactions among services. In real systems, we typically do not know the entire invocation workflows across multiple services in the Cloud. In this case, the service interaction patterns, i.e., $p_{ji}$ in Table I, can be obtained by analyzing the invocation trace of each individual service in the Cloud, or analyze the source code of a service and its execution flow.

With the information of the generated workflows, $p_{ji}$ can be calculated as follows. A workflow has *h* nodes with the random topology. A node in a workflow represents the invocation of a service randomly selected from the service pool. Therefore, a service may appear multiple times in a workflow. A link from service (node) $s_i$ to $s_j$ represents that after $s_i$ is run, $s_i$ sends a request to further invoke $s_j$. The weight of a link represents the amount of data that needs to be sent from $s_i$ to $s_j$ when $s_i$ invokes $s_j$. A workflow has a entry service (the first service that has to been invoked in the workflow). External requests arrive to invoke the entry service, which is regarded as the external demand. The arrival rate of the external requests to workflow $w_i$ is denoted as $\lambda_i$. The invocations among services inside the workflow is regarded as internal demand. With the topology of $w_i$ and $\lambda_i$, we can easily calculate the following variables for $w_i$: 1) the rate at which $s_j$ is invoked (denoted as $\lambda_i(s_j)$); 2) the rate at which $s_j$ invokes $s_k$ (denoted as $\lambda_i(s_j, s_k)$); 3) the the amount of data sent from $s_j$ to $s_k$ in a time unit (denoted as $e_i(s_j, s_k)$). In $w_i$, the probability of $s_j$ invoking $s_k$ (denoted as $p_i(s_j, s_k)$) can be calculated as $\frac{\lambda_i(s_j,s_k)}{\lambda_i(s_j)}$. If the number of different workflows generated in the simulation is *W*, then the probability of $s_j$ invoking $s_k$ (i.e., $p_{jk}$ in TableI) can be calculated as Eq.10. The total amount of data sent from $s_j$ to $s_k$ (denoted as $E_{jk}$) in a time unit can be calculated as Eq.11, while the total arrival rate of the requests to $s_j$, denoted as $\lambda^j$, can be computed using Eq.12. In the experiments, three types of workflows are generated in the experiments: communication-intensive, computation-intensive and general workflow. In the communication-intensive, computation-intensive, and general workflow, $e_{ij}$ is randomly obtained from the range of [*min_comme*, *max_comme*], [*min_compe*, *max_compe*] and [*min_gene, max_gene*], respectively. The computation time of a node in all workflows is randomly selected from the range of [*min_comp*, *max_comp*] with the average value of *avg_comp*.

$$p_{jk} = \sum_{i=1}^{W}\left(\frac{\lambda_i}{\sum_{i=1}^{W}\lambda_i} \times p_i(s_k, s_j)\right) \tag{10}$$

$$E_{jk} = \sum_{i=1}^{W}(\lambda_i \times e_i(s_k, s_j)) \tag{11}$$

$$\lambda^j = \sum_{i=1}^{W}(\lambda_i \times \lambda_i(s_j)) \tag{12}$$

TABLE II. Experimental parameters

| Parameters | Value |
|---|---|
| $S$ | 40 |
| $B$ | 50 |
| [*min_comme, max_comme*] | [20, 30] |
| [*min_gene, max_gene*] | [10, 20] |
| *avg_comp* | 15 |
| $h(the number of tasks in a workflow)$ | 40 |
| $W$ | 3 |
| $[b\_min], [b\_max]$ | [5, 15] |
| [*min_compe, max_compe*] | [2, 8] |
| [*min_comp, max_comp*] | [10, 20] |
| $slack$ | 20% |
| $\delta$ (mutation probability) | 0.2 |

There are the existing techniques [1] to obtain the function $f$ in Eq.3. The value of the function $f$ is the processing rate of a VM. In the experiments, we apply the queuing theory [18] to obtain the *g* function. Assume that the external requests arrive following the Poisson process, and the computation time of a service and the communication time of sending data between services follow the exponential process. According to the queuing theory, the average response time of service $si$, denoted as $T_i$, can be calculated by Eq.13, where $|s_i|$ is the number of VMs that is used to host $s_i$, $\mu_i$ is the mean process rate of a VM hosting $s_i$ (which is the inverse of mean computation time of an invocation in the VM of $s_i$ and is actually the value of the $f$ function) and $P_n$ is the probability that the number of requests being processed in the virtual cluster is no less than *n*. Assume the QoS of service $s_i$ is that the average response time of an invocation of the service is no more than $q_i$. $q_i$ is normally set as $avg\_comp \times (1 + slack)$. Given $\lambda^i$ and $q_i$, we can calculate from Eq.13 the minimum $|s_i|$ that satisfies the QoS, which is the *g* function in Eq.4 and Eq.5. $p_jk$ has been calculated in Eq.10. Therefore, $c_kj$ in the consumption matrix can be calculated using Eq.4. With the arrival rate of the external requests, we can apply the queueing theory to calculate the number of VMs required to serve the external requests, which is *D* in Eq.1. Finally, the number of VMs allocated to each service can be calculated using Eq.2.

$$T_i = \frac{1}{\mu_i} + P_n\frac{1}{|s_i| \times \mu_i - \lambda^i} \tag{13}$$

The capacity of a physical machine is set to be *B*. The resource capacity allocated to a VM in $s_i$ is set to be $b_i$, which is randomly selected from the range of [*b_min, b_max*]. Unless otherwise stated, the value of the experimental parameters are set as in Table II.

The existing work on placing VMs to PMs mainly focuses on achieving the minimum number of PMs used to host the VMs [5] [10] (which is called the *Min-nodes* algorithm in this paper), assuming that the VMs are independent with each other. The CAGA framework developed in this paper takes the service (VM) interactions into account. The Min-nodes method presented in [5] models the VM-to-PM placement as the bin-packing problem and then uses the existing solver to solve the problem for the VM-to-PM placement that minimizes the usage of PMs. In the experiments, we compared CAGA with the Min-nodes algorithm in terms of communication cost and the number of used PMs. Moreover, we compared CAGA with a heuristic VM-to-PM placement algorithm. In the heuristic,

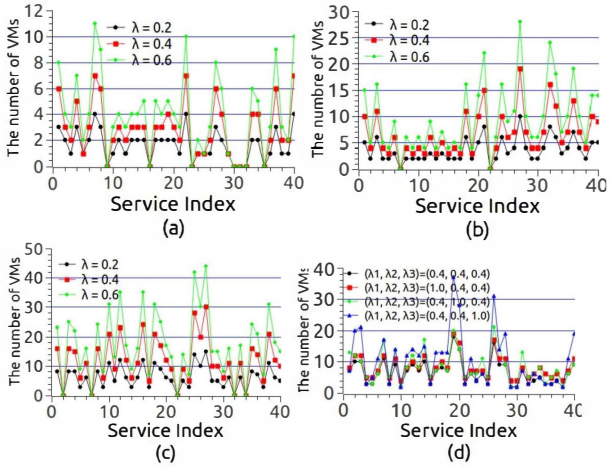


Fig. 1. Impact of the increase in external demands; a)computation-intensive workflow; b) general workflow; c) communication-intensive workflow; d) the three types of workflow combined.

the VMs from different services are placed in a PM in a round-robin fashion [19]. Starting from $s_0$,the heuristic algorithm places a VM in $s_i$ to the PM, then places a VM in $s_{(i+1)\%S}$ to the PM, until the PM cannot accommodate more VMs. Then the VMs are placed to a new PM in the same fashion, except for starting from the VM that cannot be placed to the previous PM.

### A. Impact of the increase in external demands

The experiments presented in this subsection investigates the impact of service interactions on resource capacity allocated to each service. Fig.1(a, b and c) show the number of VMs allocated to each service under different arrival rates of external requests for communication-intensive, computation-intensive and general workflows, respectively. Fig.1d shows the the number of VMs allocated to each service for the three workflows combined. The number of VMs is obtained using the Cloud-IO model. As can be seen from Fig.1(a, b and c), when the arrival rate of external requests increases, not only the number of VMs allocated to the entry service of the workflow increases ($s_1$ in the figures), but that allocated to other services in the workflow also increases. The level of increment in some services is even much greater than that in the entry service. With the Cloud-IO model, we can quantitatively obtain the impact of the increase in external demands on the resource requirements on each service in the Cloud. For example, in Fig 1b, when the arrival rate of the external requests increase from 0.2 to 0.6, it imposes the biggest resource burden on service $s_{27}$, whose VM quantity increases from 10 to 28.

### B. Comparing CAGA with the existing placement methods

This subsection compares CAGA with two existing VM-to-PM placement methods: Min-nodes [5] and the round-robin heuristic [19]. Fig.2(a, b and c) present the results for computation-intensive, general and communication-intensive workflows, respectively. It can be seen from these figures that in all cases, CAGA significantly reduces the communication

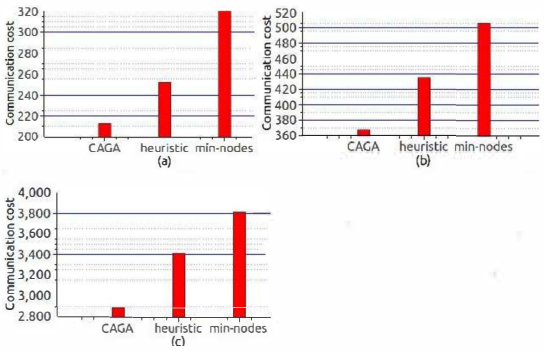


Fig. 2. Comparing CAGA with Min-nodes and the round-robin heuristic in terms of communication cost; a)computation-intensive workflow; b) general workflow; c) communication-intensive workflow.

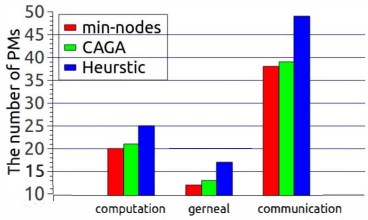


Fig. 3. Comparing CAGA with Min-nodes and the round-robin heuristic in terms of the number of used PMs

cost compared with other two methods, which suggests the effectiveness of the proposed framework.

Fig.3 compares CAGA with Min-nodes and the round-robin heuristic under different types of workflow in terms of the number of PMs used to host the VMs. It can be seen that although Min-nodes can achieve the least number of PMs, CAGA only uses one more PMs than Min-nodes in all cases. As it has been shown in Fig.3, CAGA can significantly reduce the communication cost. These results indicate that CAGA is able to greatly reduce communication overhead in the Cloud with only a tiny fraction of increase in resource usage. This is because CAGA takes the communication cost into account when designing the framework. Moreover the way used by CAGA to encode and calculate the VM-to-PM mapping ensures that there will not be the excessive spare capacity in PMs, and therefore effectively reduces the number of PMs used to host VMs.

### C. Convergence of CAGA

Fig.4(a, b and c) show the convergence of the CAGA algorithm over time under computation-intensive, general and communication-intensive workflows, respectively. In theory, one major factor that influence the convergence speed is the number of VMs to be placed into the PMs. This is because the size of the encoded solution equals to the number of VMs to placed. The size of the solution in turn determines the complexity of the crossover and mutation operation. Another major influential factor is the number of services in the Cloud, because when calculating communication cost, CAGA needs

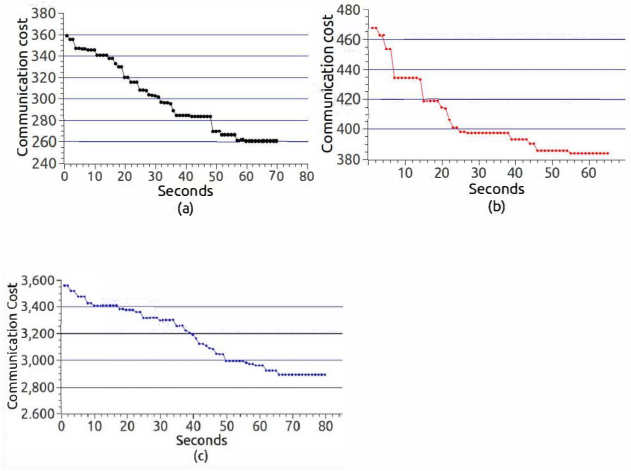


Fig. 4. Convergence speed of CAGA; a) computation-intensive workflow, b) general workflow, c) communication-intensive workflow

to calculate $\alpha_{ijk}$ for each pair of services. More services, more calculations are involved. The number of services in the experiments are 40 and the number of VMs to be placed is about 150 VMs. It can be seen from Fig.4 that the CAGA can reach the stable result for about 60 seconds in all three cases, and the longest time (65 seconds) is spent by the communication-intensive workflows in which the number of VMs to be placed is 167. The results suggest that CAGA can find a VM-to-PM placement with low communication cost fairly efficiently.

## VII. Conclusions

This paper applies the input-output model in economy to model the resource demand for interacting services in a Cloud. Based on the modelling, this paper further develops a communication-aware VM-to-PM placement framework. The framework takes into account the interaction costs among services, and aims to find a VM-to-PM placement so that the communication overhead can be minimized. The framework designs a genetic algorithm to search for the placement that can optimize communication overhead in the Cloud. The experimental results show that the proposed communication-aware framework is able to significantly reduce the communicate cost in the Cloud with little increase in the number of used PMs.

## References


[1] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 55–60, 2010.

[2] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. IEEE, 2010, pp. 62–73.

[3] A. W. Services. Amazon ec2 pricing. [Online]. Available: http://aws.amazon.com/ec2/pricing

[4] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 15.

[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] L. He, D. Zou, Z. Zhang, C. Chen, H. Jin, and S. A. Jarvis, "Developing resource consolidation frameworks for moldable virtual machines in clouds," *Future Generation Computer Systems*, 2013.

[7] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE Computer Society, 2011, pp. 26–33.

[8] 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.

[9] V. Petrucci, O. Loques, and D. Mossé, "A dynamic optimization model for power and performance management of virtualized clusters," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM, 2010, pp. 225–233.

[10] L. Hu, H. Jin, X. Liao, X. Xiong, and H. Liu, "Magnet: A novel scheduling policy for power reduction in cluster with virtual machines," in *Cluster Computing, 2008 IEEE International Conference on*. IEEE, 2008, pp. 13–22.

[11] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[12] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2876–2880.

[13] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 10.

[14] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 13–24.

[15] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, 2010.

[16] A. Williams, M. Arlitt, C. Williamson, and K. Barker, "Web workload characterization: Ten years later," in *Web content delivery*. Springer, 2005, pp. 3–21.

[17] B. L. Miller and D. E. Goldberg, "Genetic algorithms, selection schemes, and the varying effects of noise," *Evolutionary Computation*, vol. 4, no. 2, pp. 113–131, 1996.

[18] L. Kleinrock, *Theory, volume 1, Queueing systems*. Wiley-interscience, 1975.

[19] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd, "Allocating non-real-time and soft real-time jobs in multiclusters," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 2, pp. 99–112, 2006.