Journal of Organizational and End User Computing
Volume 33 • Issue 2 • March-April 2021

Placement for Intercommunicating Virtual Machines in Autoscaling Cloud Infrastructure:

Autoscaling and Intercommunication Aware Task Placement

Sridharan R., National Institute of Technology, Tiruchirappalli, India
Domnic S., National Institute of Technology, Tiruchirappalli, India

ABSTRACT

Due to pay-as-you-go style adopted by cloud datacenters (DC), modern day applications having intercommunicating tasks depend on DC for their computing power. Due to unpredictability of rate at which data arrives for immediate processing, application performance depends on autoscaling service of DC. Normal VM placement schemes place these tasks arbitrarily onto different physical machines (PM) leading to unwanted network traffic resulting in poor application performance and increases the DC operating cost. This paper formulates autoscaling and intercommunication aware task placements (AIATP) as an optimization problem, with additional constraints and proposes solution, which uses the placement knowledge of prior tasks of individual applications. When compared with well-known algorithms, CloudsimPlus-based simulation demonstrates that AIATP reduces the resource fragmentation (30%) and increases the resource utilization (18%) leading to minimal number of active PMs. AIATP places 90% tasks of an application together and thus reduces the number of VM migration (39%) while balancing the PMs.

KEYWORDS

Autoscaling, Cloud Computing, Communication Latency, Elasticity, Intercommunication, Network Latency, Tasks Migration, VM Placement

INTRODUCTION

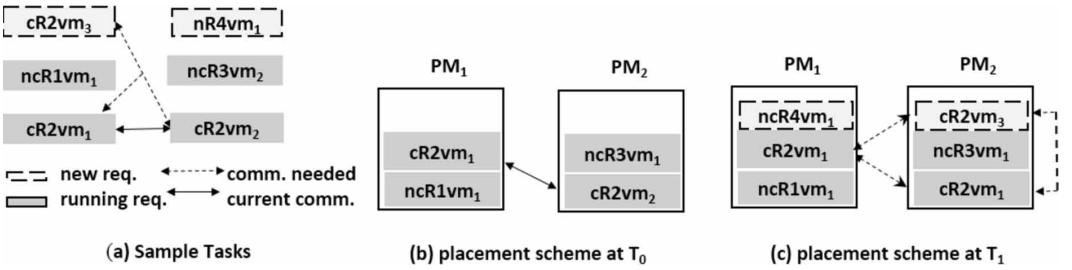
The rate at which data is generated by modern day applications is growing exponentially and is unpredictable with respect to time. Amount of information contained in the data are also disruptive for predictive data processing and analytics functionality. This data needs to be analyzed timely for building business intelligence leading to better decisions. Hence, these applications depend heavily on cloud computing paradigm. Using pay-per-use mode, cloud computing provides *autoscaling* services to manage the sporadic resource requirement of modern day applications. As industries desire reduced time to market for their applications, they adopt to cloud. Cloud operates via DC (datacenters), has huge computing and storage resources, resulting in heavy electric power consumption. This leads to higher operational cost for the CSPs (cloud service provider) and eventually the same will be passed to the customers. Hence, the current focus of researchers is to identify efficient DC management

DOI: 10.4018/JOEUC.20210301.oa2

leading to reduced power consumption while delivering high quality service to the customers. This presents several research challenges including resource management (Manvi, & Shyam, 2014). These challenges need to be addressed after considering multiple parameters related to application types, cloud services and networks of DC.

Application types could be multi-tier web applications, big data processing applications, machine learning and multimedia applications, and high-performance applications. Applications such as video surveillance, real time gaming, real time streaming and augmented reality have intercommunication tasks[1] which are too sensitive towards latency and also need more computational power. For example, VMs hosting database tasks and hosting application tasks need intercommunication in three-tier web applications. Similarly, VMs that hosts object tracker task in a video surveillance system that uses multiple cameras covering multiple areas, need inter task communication. As these applications are sporadic resource requester by nature, they depend heavily on elastic capability. Typically, these applications having group of jobs (could be long-duration jobs needing autoscaling), execute their intercommunicating or non-intercommunicating tasks using VMs (Virtual Machines instance) in a DC. Autoscaling service operates on an adjustable capacity (minimum and maximum number of VM), (Boucher Jr, et al, 2018; Barclay, 2016) within which the tasks need to be executed. Without violating this capacity range, addition or deletion of VM instances shall be carried out, upon meeting

Figure 1. Sample VM and its placement using FCFS



the resource threshold value specified by the user.

Virtual Machine Placement (VMP) (Lopez-Pires & Baran, 2015) is the process of selecting suitable Physical Machine (PM) to place the requested VM. VMP taking into account the auto-scalability of the applications along with intercommunication of theirs tasks is a challenging cloud problem.

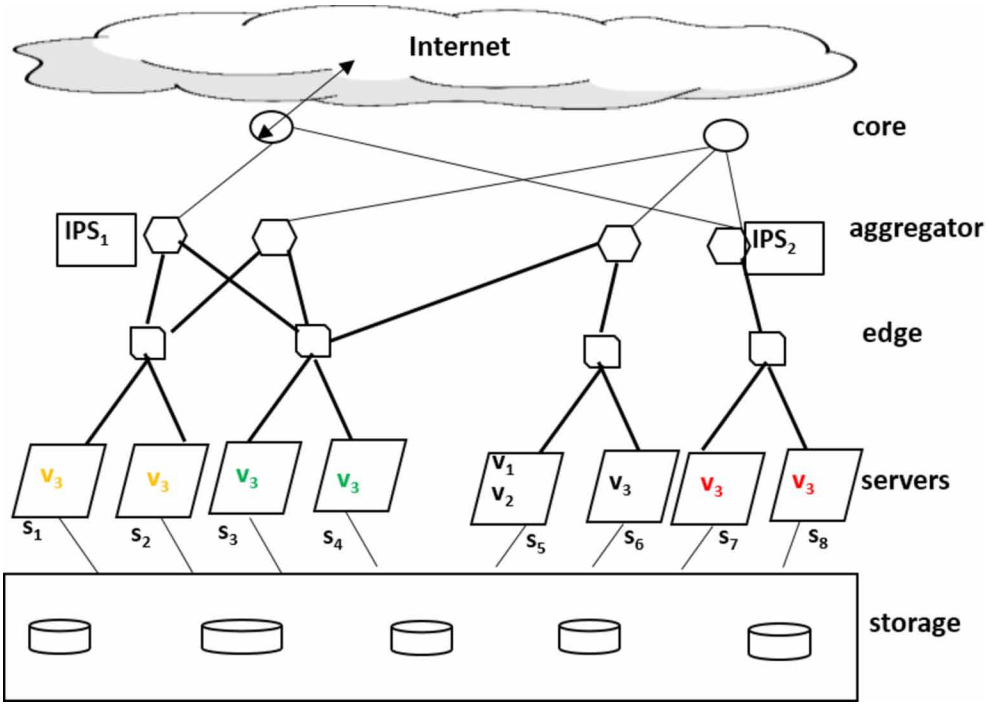
Consider an example of three application requests that are currently (at time T_0) executing in a DC. Let $ncR1vm_1$ and $ncR3vm_1$ be the VMs of first and third application respectively, which are non-intercommunication type. Let the second application have two inter-communicating VMs ($cR2vm_1$ and $cR2vm_2$) that needs autoscaling. Assume that at time T_1, fourth and fifth requests arrive to the DC. While fourth is a fresh non-intercommunicating type request ($ncR4vm_1$), the fifth request ($cR2vm_3$) is an additional VM of the second request that needs inter communication with $cR2vm_1$ and $cR2vm_2$. Further, let these tasks follow the communication pattern represented by arrows, as shown in Figure 1(a). Assuming that we have two PMs to place these tasks, using First Come First Serve (FCFS) VMP algorithm along with load balancing, placements of VMs at T_0 resembles to that of Figure 1(b). Placement of tasks at T_1 is shown Figure 1(c). If all the tasks are placed onto single PM itself, the other PM could be switched off to reduce the overall power consumption. Else placing all the intercommunicating tasks ($cR2vm_1$, $cR2vm_2$ and $cR2vm_3$) on to the same PM or on to a network

neighborhood machines will reduce the power consumption used for communication, as network latency among these tasks is minimal.

In a DC, neighborhood is defined as shortest path used for inter-server communication adhering to the pre-defined policies and topologies of the DC. Typically, servers are wired together to form a cluster that are inter-connected using routers, switches and aggregators to form a DC. Governed by network polices related to performance and security, an application over DC network has complex communication patterns. Network operators ensure these policies by deploying "middleboxes" like firewalls, load balancers, IPS (intrusion detection and prevention systems), etc. The policies demand that the traffic pass through a sequence of specified middleboxes.

Consider a flat-tree based DC network architecture as shown in Figure 2 in which, server S_5 currently (time T_0) executes two VMs (V_1 and V_2) of an autoscaling application. Assume that at time T_1 third VM (V_3) of the same application needing intercommunication with (V_1 and V_2) needs to be placed. Also, assume that S_5 does not have enough resources to hold V_3. Based on the communication path, S_6 is the best suitable PM to place the V_3. If it is not possible, next best possibilities are: S_3 or (/) S_4, S_1/S_2 and S_7/S_8 in that order. It is to be noted that even though the communication from S_5 to S_1/S_2 and S_7/S_8 needs five network-hops, it is preferred to have S_1/S_2 , as communication to S_7/S_8 mandatorily needs to go through the IPS_2 and hence wastes the network resources.

Figure 2. DC Architecture with policy aware placement



Motivation

From the above explained two scenarios, we can conclude that intercommunicating tasks are bound to give better performance, when they are placed together on to the same PM (Meng et al., 2010) or network neighborhood PM (Cui et al., 2017). This also reduces the power consumption of DC due

to reduced network latency and probable early VM migrations. Hence, after formulating AIATP (Autoscaling and Intercommunication Aware Tasks Placement) problem, authors propose an efficient AIATP algorithm to improve the resource utilization of DC. Novelty of this algorithm is, using the prior-tasks-placement knowledge of a request (application) for placement of its additional tasks adhering to co-allocation and fixed PM constraints. In summary, contributions of this paper are as follows:

- A mathematical modeling of Autoscaling aware VMP for intercommunicating tasks (AIATP) of cloud applications is developed. This model considers applications, having group of tasks or single task during their life-time for placement instead of individual tasks, a widely followed practice is presented.
- A comprehensive task placement algorithm is proposed, which uses prior-task-placement knowledge of autoscaling applications for the placements of their additional tasks. This algorithm produces VM-to-PM mapping with best possible network efficiency and thereby increases the application performance.
- Results of extensive simulation are presented to show the effectiveness of the proposed algorithm via reduced VM migrations and resource fragmentation, improved resource utilization and maximum co-placed tasks for autoscaling requests.

Rest of this paper is organized as follows. Related Works reviews the available literature related to the problem. Next problem formulation is described and proposed modeling is illustrated using an example. Subsequently, the proposed algorithmic solution along with its complexity is discussed. After presenting the implementation details, the paper proceeds to evaluate the AITAP with other algorithms before presenting the conclusion.

RELATED WORKS

As cloud computing is built on the concept of resource sharing through internet, there exists lot of challenges related to various cyber security (Zhaolong et al*., 2017*). Further using cloud as base new technologies like IoT, fog/edge computing have emerged and hence secured integration of these technologies with cloud is paramount (Stergiou et al*., 2018*). However, authors restrict the focus of the present work only to placement algorithms.

(Rahman & Graham 2017) have proposed static initial VM placement where, VMs are co-placed based on their compatibility in terms of their varying resource requirements. In this work, PMs and VMs are differentiated based on their past usage as resource critical and non-critical. But, they have not considered the autoscaling requirements of an application hosted by these VMs.

(Cui et al., 2017) have brought out the importance of policy and network awareness before considering VM migrations. They have demonstrated an efficient VM management scheme which reduces the communication cost while considering PMs to migrate the VMs.

Further, some contributions have explored two-stage approach as well for VMP problem. (Beloglazove et al., 2012) proposed two stages for the VMP: (i) initial placement of VMs and (ii) optimizing the current placement. In this, using CPU utilization of the VMs, a Modified Best Fit Decreasing (MBFD) algorithm was used for initial placement and the second phase was triggered whenever PMs breach the CPU thresh-hold value. Without considering autoscaling, (Zheng et al., 2016) have subdivided the VMP as two sub-problems. A Best Fit (BF) algorithm, used for incremental placements acted as first step followed by another step in which, periodical trigger of VM consolidation was suggested.

Another two phases online VMP was proposed by (Shi et al., 2015) where PMs and VM requests are represented as vectors. During the first phase, based on *cosine* similarity of the vectors, PM types

were assigned to VM requests and based on resource requests, VMs were categorized for each PM type. Using utilization thresholds of PMs, reconfiguration of VMs was triggered in the second phase. But, authors have not considered the elasticity in their work. Another two-phase approached for VMP proposed by (López-Pires et al., 2018) has considered complex IaaS environment including elasticity but did not consider the application performance while placing the additional VM of autoscaling requests.

While (Yokoyama et al., 2017) have used affinity awareness among the VMs to create the cluster node and do the placement, (Chen, et al., 2016) have simply used affinity awareness among the VMs for their placements. In (Chen, et al., 2016), it was demonstrated that the application running on the VMs that are placed using this placement scheme improves the performance rather than placing them individually without knowing their affinity. (Meng et al., 2013; Meng et al., 2010; Cui et al., 2017) have demonstrated the importance of network awareness during VM placement for an efficient DC management. (Al-Dulaimy et al., 2018) present a distributed approach for VM consolidation and use Multiple Choice Knapsack for VMP. (Li et al., 2013) discuss the elasticity aware VMP under the limitation of both PM capacities and bandwidth capacities to present a hierarchical VMP, where network-switch capacity from bottom to top is accumulated and used as a parameter for VMP.

Detailed survey on issues and challenges involved in cloud resource scheduling has been presented (Sukhpal & Inderveer 2016). Further, (Sukhpal & Buyya 2018) proposed a scheduling frame-work for resource provisioning by autonomic offering in heterogeneous cloud. However, these works have not addressed the situations needing inter-communication among the various tasks of single application.

To the best of author's knowledge, this work is the first attempt that can optimize the placements for the tasks of an application that needs autoscaling and intercommunication, using the knowledge of prior-placement of tasks from these applications.

PROBLEM STATEMENT AND PROPOSED MODEL

This section explains the problem addressed through this work and its mathematical modelling followed by an illustrative scenario.

Problem Statement

Applications that want to use cloud resources can be broadly classified as those needing autoscaling service and those not needing autoscaling. Further these applications can be differentiated as those having intercommunicating tasks and non-intercommunicating tasks. As implemented by popular CSPs (Boucher, et al, 2018; Barclay, 2016) applications aspiring to have a*utoscaling* service need to specify a minimum (*minVM*) and maximum (*maxVM*) number of instances to run and add or remove VMs automatically based on a set of rules. In effect, the number of VMs dedicated to these applications during theirs life-time shall be between *minVM* and *maxVM*.

VM requests emanated from the applications are queued for predefined time period in a batch and are processed together in a DC. These requests can be classified into three types, namely (a) new VM(s) requests, (b) migrating VMs requests and (c) incremental VM requests of autoscaling applications. Efficiently placed *minVMs* and their incremental VMs can give various benefits to both user and CSP. Firstly, Incremental VMP, when solved as a fixed placement problem is bound to increase the application performance as network latency is avoided. Secondly, placing *minVMs* together reduces the potential migrations, as these requests are normally long duration jobs during server consolidation. This reduced migration and network latency lead to minimal power consumption in DC. Thirdly, as individual PMs are proposed to be balanced with VMs serving both autoscaling and *non-autoscaling* requests, SLA violations for an autoscaling requests are bound to be minimal. This is because forced migration if any, to accommodate additional VMs for autoscaling requests, will start from non-autoscaling VMs.

Encouraged by above mentioned benefits, a model to the problem of identifying efficient placements for intercommunicating tasks of an autoscaling application is proposed.

Proposed Model

This model classifies the request into three types: autoscaling requests having intercommunicating tasks ($\mathbb{R}^c_{as}$) *non*-autoscaling requests but having intercommunication $\left(\mathbb{R}^c\right)$ and all other requests ($\mathbb{R}$) or modeling.

To formalize the problem, we make the following assumptions:

- Let $K\left(\tau\right)$ tasks belonging to each request of type $\mathbb{R}^c_{as}\left(\tau\right)$. $\mathbb{R}^c\left(\tau\right)$ and $\mathbb{R}$ be pooled in the batch to be placed at the time slot $\tau \;\; 1 \le \tau \le T$;
- Let $p = \left\{p_1, p_2, \ldots p_n\right\}$ and $v = \{v_1, v_2, \ldots v_m\}$ be the identifier PM and VMs constrained with *D*-dimensional resource vectors *R* (number of CPUs, memory size, storage space, bandwidth etc.);
- Let $r_k, k \in \left\{1 \ldots . \alpha\right\}$ be a positive integer that uniquely identifies the requests defined by a four tuple:

$$r = \left\{u, VM_{type}, minVM, maxVM\right\}$$

where:

$$minVM \;\&\; maxVM$$

are number of VMs:

$$VM_{type} = \left\{micro, small, medium, large, xlarge\right\}$$
$$u = \left\{\mathbb{R}^c_{as}, \mathbb{R}^c, \mathbb{R}\right\}$$

Note that when $u \notin \mathbb{R}^c_{as} \Rightarrow \;\; maxVM = 0$.

Let $S\left(T\right)$ be the requests sequence at the time slot $\tau \;\; 1 \le \tau \le T$. Then, from the assumptions we have:

$$S\left(T\right) = \sum_{k=1}^{\alpha} r_k(\tau), r_k\left(\tau\right) = K\left(\left(\tau\right) = \sum v_i, 1 \le i \le minVM\right)$$

Let function Φ indicate the placement schema $\Phi\left(r_k\right) = p$ if task of r_k th request is placed on the p th PM. Then the objective is, given a request sequence $S\left(T\right), 1 \le \tau \le T,$ and PMs $p, 1 \le p \le n$ with capacity $\mathcal{C}$ find a placement function Φ :

$$minimize \sum_{\tau=1}^{T} n\left(\tau\right) \tag{1}$$

subject to the constraints presented below.

- **Resource Constraints:** Ant any given point of time across different time slot, combined resource requirements of VMs in a PM cannot exceed the resource capacity of PM. During the placement of $K\left(\tau\right)$ VMs belonging to $r_k\left(\tau\right)$ request in the τ th time slot, the PMs identified to host these VMs, are subject to primary resource constraints:

$$\forall 1 \le p \le n, 1 \le d \le D$$

$$\sum_{r_k}^{\alpha} \sum_{v=1}^{r_k(\tau)} X_{vp} * R_v^d \le \mathcal{C} \tag{2}$$

where $R_v^d \;\; 1 \le d \le D$ is the resource demand of v^{th} VM and:

$$X_{vp} = \begin{cases} 1, if \;\; v^{th} \;\; VM \;\; is \;\; placed \;\; onto \;\; the \;\; p^{th} PM \\ 0, otherwise \end{cases}$$

- **Placement Constraints:** Each VM of user request r has to be placed to run on a single PM:

$$\forall 1 \le p \le n, \sum_{r_k=1}^{\alpha} \sum_{v=1}^{r_k(\tau)} X_{vp} = 1 \tag{3}$$

Adjusting resource capacity of individual PM and VM in each dimension measured on different scales to a notionally common scale is called normalization. Each item has resource vector $R_i = \{R_i^1, R_i^2, \ldots . R_i^d\}$ and each dimension $l, l \in \left[1 \ldots d\right]$, denotes resource (CPU, memory etc.) demand size R_i^l which is normalized $0 \ldots 1$ and capacity of each PM is also normalized to 1, then we have:

$$X_{vp}\left(\tau\right) \in \left\{0,1\right\} \text{ and } n\left(\tau\right) \in \left\{0,1\right\} \tag{4}$$

- **Co-Allocation Constraint:** In any request of type $\mathbb{R}_{as}^{c}$ and $\mathbb{R}^{c}$, let tasks $v_1\left(\tau\right)$ and $v_2\left(\tau\right)$ have co-location requirement and are required to be co-located onto same PM $p \in \left[1 \ldots n\right]$, then we can combine tasks $v_1\left(\tau\right)$ and $v_2\left(\tau\right)$ to one group and form a new task $v_{new}\left(\tau\right)$. For each resource dimension d, the size of the group task is denoted by sum of tasks $v_1\left(\tau\right)$ and $v_2\left(\tau\right)$:

$$R^d_{v_{new(\tau)}} = R^d_{v_1(\tau)} + R^d_{v_2(\tau)} \tag{5}$$

Hence, we add the following constraint to AIATP model:

$$\forall p \in \left[1 \ldots n\right], \left\{\forall v_1, v_2 \in \left[1 \ldots minVM\right]\right\} \in r_k\left(\tau\right) \& \, r_k \in \left\{\mathbb{R}^c_{as}, \mathbb{R}^c\right\}$$

$$X_{V_1 p} = X_{V_2 p} \tag{6}$$

- **Balanced PM Constraint:** Individual PMs shall contain task of requests belonging to all types namely, $\mathbb{R}^c_{as}, \mathbb{R}^c$ and $\mathbb{R}$. Note that this constraint is for initial task placements only and not for additional tasks placements:

$$\forall 1 \le p \le n, \sum_{a=1}^{\alpha} r_a\left(\tau\right) \underset{\sim}{<} \sum_{b=1}^{\alpha} r_b\left(\tau\right) + \sum_{c=1}^{\alpha} r_b\left(\tau\right) \tag{7}$$

where, $r_a \in \mathbb{R}^c_{as}$, $r_b \in \mathbb{R}^c$ and $r_c \in \mathbb{R}$. By adhering to this constraint, migration-related SLA violations during server consolidation can be avoided for $\mathbb{R}^c_{as}$ requests, as autoscaling requests are expect to utilize the resource for longer duration.

- **Comment:** This constraint might give the impression of forced resource fragmentation and this is true, when small numbers of VMs are considered for placements. However, in real world DCs, the number of VMs considered for placement will be huge and as number of VMs increases, the resource fragmentation tends to reduce which is demonstrated through our simulation.
- **Fixed PM Constraint:** Let $\mathcal{R}_{ij}$, where $i, j \epsilon \left\{1 \ldots m\right\}$ is a requests-relation matrix that represents relation among VMs of individual requests during theirs life-time. Each element of this matrix is filled with r_k the request identifier, to which the current VM belongs. Let $r^{'}_k\left(\tau'\right)$ denote the additional VM of an autoscaling request $r_k\left(\tau\right)$. Then, for each additional VM of an autoscaling request, we add the following fixed PM constraint:

$$\forall r_k \in \mathbb{R}^c_{as}, 1 \le k \le \alpha, \; x_{vp} = r^{'}_k\left(\tau'\right) = r_k \tag{8}$$

- **Comment:** On availability of resources, this constraint helps us in placing additional VM request of autoscaling user at time τ' on to the same PM where the earlier VMs of this request at time τ are placed. Otherwise, the placement is achieved by migrating the non-autoscaling requests VM(s) from the said PM or from the nearest network neighborhood machine so that the intercommunication cost of VMs belonging to this request over the period of time will be minimal.

Hence, the objective (1) with constraints of (2), (3), (6) and (7) needs to be achieved for an effective initial task placement of an intercommunication needing request while the objective (1) with constraints of (2), (3) and (8) needs to be satisfied during the additional VM requests of an autoscaling request.

Claim: AIATP is NP-Complete.

Proof: As per constraint (6), resource demand vectors of all the VMs specified by *minVM* of request having intercommunicating tasks, are combined to form a new VM having higher resource demand vector. Characteristics of VM, an equivalent to an item in a bin-packing problem, are not altered by adhering to this constraint. Similarly, constraint (7) which ensures that the individual PMs are balanced does not change the characteristics of bin while placing the item. As stated in (4), we can equate the normalized the AIATP problem to that of normalized VMP problem by dividing VM sizes with the corresponding resource capacities of PMs. Hence, similar to general VMP problem, AIATP is also NP-Complete.

Since minimization problem of AIATP is NP-Complete, we cannot obtain the optimal placement solutions in polynomial time unless P=NP (Heidelberg, 2006). Hence, we consider an algorithmic approach to get an approximate solution in this paper.

Illustrative Scenario

For an illustrative purpose, initially at time t_0 consider 8 that requests are pooled in the batch and 5 PMs are available to place the VMs of these requests. Among these 5 PMs, assume that first 2 follow different network policy than that of remaining 3 PMs. Also, assume that requests 1, 3, 4, 6, are of type non-autoscaling and requests 2, 5, 7, 8 are of type as having 3, 2, 2, and 3 respectively as the *minVM*. For the sake of simplicity in this illustration, we are not considering the resource dimensions of PMs and VMs.

After the completion of initial VMP using first fit algorithm, VM's request relation matrix will be formed as $\mathcal{R}_{t0}$ shown in Figure 3. In this matrix, the columns correspond to number of PMs and

Figure 3. Transistion of Autoscaling request-relation matrix for different timeline

$$\mathcal{R}_{t0}=\begin{pmatrix}1&4&7&8&0\\2&5&7&8&0\\2&5&0&8&0\\2&6&0&0&0\\3&0&0&0&0\end{pmatrix}\mathcal{R}_{t1}=\begin{pmatrix}2&4&7&8&0\\2&5&7&8&0\\2&5&9&0&0\\2&6&10&0&0\\3&1&0&0&0\end{pmatrix}\mathcal{R}_{t2}=\begin{pmatrix}2&4&7&8&10\\2&5&7&8&11\\2&5&9&12&11\\2&6&3&0&0\\2&1&0&0&0\end{pmatrix}\mathcal{R}_{t3}=\begin{pmatrix}2&2&7&8&10\\2&5&7&8&11\\2&5&9&12&11\\2&6&3&0&0\\2&1&4&0&0\end{pmatrix}$$

the individual elements represent the VM having unique request identifier as its value. Repetition of same value for the elements indicates that these elements belong to same request, while number of such repetition represents the value of *minVM* in that request. Consider the placement scheme for the request 8: this request has 3 *minVMs* and they are placed together in PM 4. Similarly, the placement scheme for other VMs can also be identified. Note that 0 value elements represent the free space available in the PM. Even though request 8 can be placed on to PM 3, owing to constraint (7) it is actually placed on PM 4.

Let us look at the changes in the request relation matrix $\mathcal{R}$ for various time intervals. It is to be noted that in the subsequent time intervals there can be two types of requests: addition or deletion of VMs by the already running autoscaling requests; new autoscaling and other requests types can be present. Assume that request 2 asks for an additional VM, request 8 asks to delete a VM and new autoscaling request 9 with *minVM* as 1 and *non*-autoscaling request 10 are pooled in at time t_1. Assuming that, all other VMs which are initially placed are still in running state, request 1 is migrated from PM1 to PM2 to accommodate the additional VM of request 2 as shown in $\mathcal{R}_{t1}$ of Figure 3. Further, when additional request emerged from request 2 at time t_2 together with a non-autoscaling request 12 and autoscaling request 11 having 2 *minVM* the matrix looks like $\mathcal{R}_{t2}$ of Figure 3. Similarly,

assume that at time t_3 also, the request 2 asks for an additional VM. Since PM1 does not have enough capacity to place this additional VM, it is to be placed onto a network neighborhood machine, which is PM 2 in our case, as this is an additional VM of autoscaling request 1. Hence, request 4 is moved to PM 3. The transformations at time t_3 are depicted in $\mathcal{R}_{t3}$ of Figure 3.

AUTOSCALING AND INTERCOMMUNICATION AWARE TASK PLACEMENT ALGORITHM

In this section, we present a VMP scheme guided by AIATP problem modelling based on request types ($\mathbb{R}^c_{as}$, $\mathbb{R}^c$, and $\mathbb{R}$ rather than VM_{type} for optimal operations of DC. This scheme recognizes autoscaling and tasks-intercommunication of the requests for placements and network policy for migration. Initial placement focuses on placing the VMs of individual requests efficiently, so that future migrations and network overheads are minimal. This also ensures that the overheads in-terms of identifying suitable VMs and PMs that are candidates for migrations are reduced. Subsequently, additional VMs of an autoscaling requests are placed in such a way that the inter-communication among the tasks of this request are confined to the individual PM or network neighborhood PMs resulting in better application performance.

To develop such an algorithm, we make some simplifying assumptions as follows:

- DC has enough capacity to serve all the requests including additional VMs.
- In consistent with CSP, Amazon (Boucher Jr, et al, 2018; Barclay, 2016) users are expected to specify the *minVM* and *maxVM,* which captures application's autoscaling and intercommunication requirements.
- Without loss of generality, we assume that minimizing cross-traffics and migrations of VMs reduces the overall energy consumption of a DC and increases its operational efficiency.

In this work, initial VMP is achieved by adhering to constraints (2), (3), (6) and (7). Fixed placement pattern that includes network neighborhood machines, is adopted for the additional VM of an *autoscaling* requests. The total (original) capacity, current utilization of PM which is in consideration for placement, and resource demand of VM are denoted by *Ct*, *Cu* and *Cr* respectively. Note that all these parameters are defined in terms of resource vectors (number of CPUs, memory, storage, bandwidth etc.).

AIATP algorithm organizes the tasks, after extracting their details from the individual request, according to various constraints listed in *problem modelling* section. Subsequently these tasks are sent to ALLOC sub-function along with candidate PM, which accomplishes the placements. Upon non-availability of enough resources in the PM, ALLOC function initiates new PM and places the VMs. The detailed description of the algorithm follows.

AIATP algorithm

```
Require:  batch containing requests at time τ active PM number n
Result: VM placement  /* refer section 3 for definition of symbols
*/
1.          while batch != NULL do
2.                Create requests identifier r_k(τ) ← K(τ)
3.          end while
4.           for each r_k(τ)
5.                  if r_k(τ) for deletion
```

```
6.                    process scale-down of VM
7.              else if r_k(τ) for addition /* fixed PM (r'_k(τ')) constraint */
8.                  C_u = ALLOC (X, r_k, p'_n)
9.              else if u ∈ ℝ^c_as /* co-location constraint */
10.                     X ← minVM * R^d_v(τ)(VM_type)
11.                         add X to List1
12.         else if u ∈ ℝ^c
13.                     X ← minVM * R^d_v(τ)(VM_type)
14.                         add X to List2
15.         else if u ∈ ℝ
16.                     X ← R^d_v(τ)(VM_type)
17.                         add X to List2
18.       end for
19.       sort on VM_type: List1 in descending order
20.       sort on VM_type: List2 in ascending order
21.       while List1 != NULL do
22.         C_u = ALLOC (X, u, p_n)
23.         remove X from List1
24.         if C_u >= 1/2 * C_t) /*balanced PM constraint */
25.             for each X in List2
26.                 C_u =ALLOC(X, u, p_n)
27.                     remove X from List2
28.                     if (C_u = C_t) break;
29.             end for
30.       end while
31.       for each X in List2
32.             C_u = ALLOC(X, up_n)
33.       end for
```

Explanation of AIATP Algorithm

This algorithm does a comprehensive tasks placement by considering all types of tasks ($\mathbb{R}^c_{as}, \mathbb{R}^c$ and $\mathbb{R}$) instead of concentrating only on *autosacling* requests that have intercommunicating tasks. Individual requests, both from cloud service and users, are collected into a batch. Additional VM of autoscaling requests emanating from cloud services, are also queued to the same batch. Each request is defined by four-tuple as explained in problem formulation. However, if the request is emanated from cloud service then, information indicating whether the request is to scale-up or scale-down the VM, and a unique request identifier through which the four-tuples can be extracted, defines the request.

At fixed time slot τ requests pooled in the batch are differentiated as fresh request and additional request. After extracting VM details from VM_{type}, each request are segregated as $\mathbb{R}^c_{as}$, $\mathbb{R}^c$ and $\mathbb{R}$ before storing them in two different lists namely *List1* and *List2* [lines 9- 17]. Note that, according to co-allocation constraint of the proposed AIATP, the demand resource vectors of all the VMs, given by $minVM$ of new requests are combined to form a new VM and considered as single placement. Hence, this process is completed before adding the VM details to *List1* [line 10, 13]. If the request

is for scale-up or scale-down of VM from an existing autoscaling request, then PM details are extracted from the request identifier and processed accordingly [lines 5-8]. Sorting of *List1* from large to small [line 19] and *List2* from smaller to larger [line 20], ensures that individual PMs are filled with combination of large sized VMs of $\mathbb{R}^c_{as}$ request with that of small sized VM of $\mathbb{R}^c$ and $\mathbb{R}$ requests. This helps in accommodating the additional VMs of $\mathbb{R}^c_{as}$ request in the future with minimum SLA violations if any. We accomplish the adherence to balanced PM constraint [lines 21-29] by allocating VMs of $\mathbb{R}^c_{as}$ request till PM capacity reaches half of its capacity, then switch to *List2* and start allocating the VMs of $\mathbb{R}^c$ and $\mathbb{R}$ users until the PM is fully used. Once the particular PM is full, the control switches back to *List1*. Finally, leftover VMs of $\mathbb{R}^c$ and $\mathbb{R}$ requests if any, are also allocated [line 30-32] on to available PMs instead of considering them with the next batch. The actual allocation of VMs on to PMs is done by ALLOC function, which is explained next.

Mapping of VMs to PMs (ALLOC)

```
Sub function ALLOC: Mapping of VMs to PMs
Input: X (VM to be placed), r_k, pn (candidate PM)
Output: C_u
1.        extract  u  from  r_k
2.        C_a ← C_t − C_u  and  C_r = X · R^d
3.        if  u ∈ ℝ^c_as  and X is an additional VM
4.           if (C_a > C_r) map X onto p_n
5.           else if (p_n ∋ X′ · R^d ∈ ℝ ≥ C_r)
6.              migrate X′ from p_n using FirstFit
7.              map X onto p_n
8.           else map X  onto n/w neighborhood machine by
repeating step 4-8
9.           else if (C_a > C_r) map X onto  p_n
10.        else map X onto new PM
11.        return C_u
```

For *ALLOC* function VM, request type and candidate PM are given as input. After calculating available C_a and extracting resource demand vector C_r of current VM in consideration for placement, [line 3] function proceeds to map the VM onto the candidate PM given as the input. If the request is for an additional task of $\mathbb{R}^c_{as}$ requests, on availability of enough resources greater than or equal to C_r, it is directly mapped on to the PM [line 5]. Otherwise, suitable VM is migrated belonging to $\mathbb{R}$ or combination of VMs belonging to $\mathbb{R}^c$, from that PM using First Fit algorithm and places the additional VM on to that PM. [lines 6-8]. Upon unsuccessful placement, suitable PM from network neighborhood is identified and executes steps 6-8 until successful placement happens. If the request is new one ($\mathbb{R}^c_{as}$, $\mathbb{R}^c$ and $\mathbb{R}$), then on availability of enough resources, we directly map the VM onto the give PM, failing which, new PM is initiated and mapping of VM is done[lines 10-11]. Finally, the current utilization of the PM used for the latest placement process is returned to the AIATP algorithm.

Cost Analysis of AIATP

Number of VM requests (N) in the batch will be a modest constant. For example, if scan interval of a batch is 5 minutes and on an average 10 VM requests get queued up, then AIATP needs to scan [lines 3-16] these 10 requests only and hence the cost is O(N). Cost of sorting the *List1* and *List2* is

O(N log N). Since *List1* and *List2* are of size O(N/2), the following loop[lines 19 - 28] will also execute O(N) times, implying that Allocate, will also be called O(N) times. The inner loop having break [line 27] will execute once sufficient VMs of $\mathbb{R}^c$ and $\mathbb{R}$ are placed matching to that of VMs from $\mathbb{R}^c_{as}$. As the number of VMs in a PM is bounded by a constant, the cost of inner loop [lines 24-27] is O(1). Last loop [line 29-31] that places the leftover VMs also has a cost of O(N). Let M be the number of candidate PMs in sub function ALLOC, then packing the VMs onto PM, results in maximum cost of O(M). Therefore total cost of AIATP is O(N log N) + O(M). Cost of the well-known algorithm like FFI/ FFD and FCFS which are under consideration for this work are O(N log N) and $O(N^2)$. Hence, cost of AIATP is asymptotic to FFI/FFD and also gives better application performance along with reduced operation cost for DC.

IMPLEMENTATION

Efficiency of any VMP algorithm needs to be evaluated in a large-scale cloud platform. However, it is difficult to conduct the repeatable VMP experiments in a real production cloud platform. Even though, there exist few open source solution to build own cloud environment (Barkat et al, 2015), researchers mostly uses simulation environments to evaluate their new algorithms. Hence, the proposed AIATP algorithm is also evaluated using *CloudSimPlus (*Manoel, et al., 2017) simulator using randomly generated VMs. The algorithms were implemented as an extension to *SimpleVmAllocationPolicy*, which determines how VMs are assigned to the host and additional tasks are assigned using *HorizontalVmScalingSimple* classes.

The proposed algorithm, written in Java, is tested on a Dell workstation with Intel Xeon 3.30Ghz and 16Gb memory, having x64 architecture. The simulation is performed over a datacenter made up of twenty homogeneous physical machines whose configuration is presented in Table 1. User requests are constructed using random generated values for u, VM_{type}, *minVM* and *maxVM*. VM types and their resource characteristics are adopted similar to Amazon-AWS autoscaling service, is presented in Table 2.

For each experiment, we created three bunches of randomly generated requests in the ratio of 2:1:1 for time slot τ_1, τ_2 and τ_3. Total resource demand of VM is approximately made equal to that

Table 1. Host configuration used for simulation

Core	RAM	VMM	Storage	MIPS
16	64 GB	XEN	1 TB	100000

Table 2. VM characteristics adopted from Amazon

	Micro	Small	Medium	Large	Xlarge
vCPU Memory(GB)	1 1	1 2	2 4	2 8	4 16

Table 3. Characteristics of requests in a batch

Total Requests	Requests Type: $\mathbb{R}^c_{as}$	Requests Type : $\mathbb{R}^c$	Requests Type: $\mathbb{R}$	New VM Requests: $\mathbb{R}^c_{as}$	New VM Requests: $\mathbb{R}^c$	New VM Requests: $\mathbb{R}$	Additional VM Requests	Delete VM Requests
40	9	11	20	22	32	40	8	4

of total resource capacity of n PMs. While the first bunch is made up of fresh requests, the second and third bunch can also contain requests for scale-up or scale-down of resources from those of earlier autoscaling requests in addition to the fresh requests. Representing workloads, tasks having 1000 byte is attached with each VM. Characteristics of requests in a single batch are presented in Table 3. While delete request is relevant for batches used in τ_3 onwards additional request is relevant for batches used in time τ_2 onwards, the remaining columns are valid for all batches. Values of new VM requests for type $\mathbb{R}^c_{as}$ and $\mathbb{R}^c$ are equal to the number of *minVMs* instance requested.

Running the simulator with a request batch as input for FFD, FFI, FCFS and AIATP algorithms constitute single experiment for our study. To avoid transient anomalies, we run ten such experiments with different set of request batches and collect the following metrics.

Resource Utilization

Figure 4 depicts the average resource utilization of all 20 PMs involved in each experiment in percentage terms. When we calculate average host utilization over ten experiments each for the three timeslots, AIATP outperforms other algorithms by using 18% more. Slope of the trend lines drawn in dotted-lines also indicates that the resource utilization of other algorithm is reducing as compared to AIATP when more requests are served.

Resource Fragmentation

As explained in the previous section, three groups of requests have been created for each experiment. While the combined resource demand capacity of the first group is approximately equal to the combined capacity of 10 PMs, the capacity of remaining two groups is equal to that of 5 PMs each. In order to measure the efficacy of AIATP with other algorithms in our consideration, we normalize the number of PMs utilized for placing each request groups to 10, 5, and 5 respectively. When we consider the placements across the timeslots, fully non-utilized PM count increases with other algorithms whereas, it is decreasing for AIATP by 30% as indicated by the trend lines of Figure 5. This indicates that the fragmentation created as AIATP adheres to constraint (7), is reduced over the period of time. Reduced fragmentation and increased utilization leads to minimal active PMs.

Autoscaling Request Placement

Next parameter in our consideration is success of placing VMs of $\mathbb{R}^c_{as}$ type request on to the same PM. This is believed to improve or at least maintain the performance of an application executing using these VMs (Chen, et al., 2016; Meng et al., 2013; Meng et al., 2010; Cui et al., 2017) as inter-communication among them is minimal. It is to be noted that, from time τ_2 onwards the autoscaling requests can be an addition/deletion of VM for an existing request or for a new one. The same is

Figure 4. Average PM Utilization during each experiments

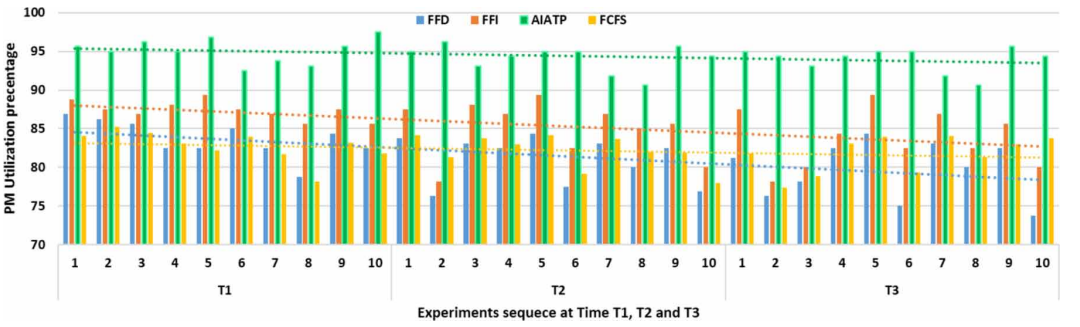
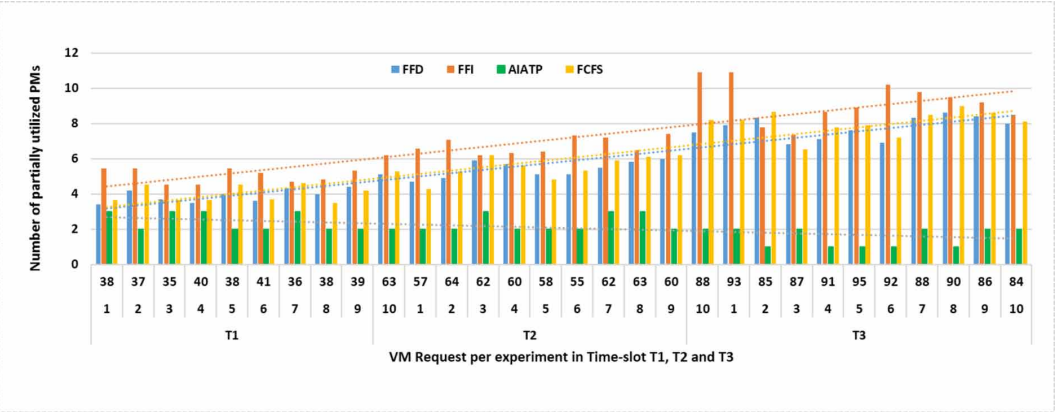


Figure 5. Resource Fragmentation across time slot

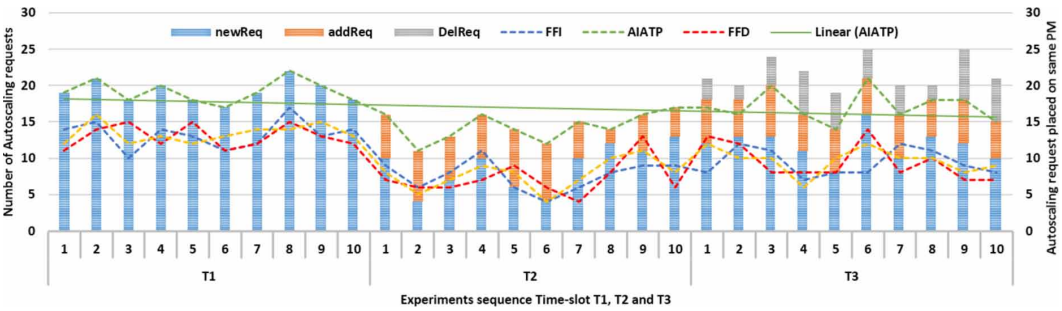


portrayed through different shades for each experiment in Figure 6. The actual number of requests placed during these timeslots is also shown (note that the y-axis is defined by the legend on the right). Trend lines of this picture display that AIATP outperforms, by placing all the request onto the same PM during the timeslot τ_1 and τ_2. However, in τ_3 understandably, it places only 90%, because of the combined resource demand by some individual requests have exceeded the capacity of individual PM and hence, got placed on the network neighbourhood machines. Obviously, this led us to analyse the migration needs of algorithms that are in our consideration.

Potential Migration Needed

Potential number of migrations needed to place all the VMs of $\mathbb{R}^c_{as}$ *and* $\mathbb{R}^c$ requests on to the same PM across τ_1, τ_2 and τ_3 for all the algorithms is calculated for our next experiment. It is to be noted that this metric is applicable to AIATP from τ_2 onwards only. Figure 7 indicates along y-axis, the number of VM migrations needed for adherence of constraint (6), when using FFD, FFI anf FCFS. Evidently, this count is zero for AIATP at time τ_1 and slowly increases over τ_2 and τ_3 due to forced migration of VMs to accommodate additional VMs. Number of migrations is directly proportional to the number of autoscaling requests, having $minVMs = 1$for other algorithms. For AIATP, number of migrations is directly proportional to that of additional VMs requested. So the cost of VM migration and the performance of application running in these VMs are directly proportional in case of AIATP.

Figure 6. Placement of Autoscaling requests having intercommunication Tasks on same PM



Hence, optimal number of *minVMs* together with reduced number of additional requests will yield better performance of AIATP.

Balanced PMs

Final evaluation parameter is to identify how well-balanced are the individual PMs. As per the constraint put forth in (7), individual PMs of datacenter needs to be blanced in-terms of hosting VMs of autoscaling requests needing intercommunication together with VMs of other type of requests. This ensures that, if enough resources are not avilable to host the additional VM of $\mathbb{R}_{as}^{c}$, in a particular PM, then migration of VMs belonging to $\mathbb{R}$ and $\mathbb{R}^{c}$ in that order is affected to accommodate the additional VM.

First, resource utilization percentage of active PMs for hosting VMs of $\mathbb{R}_{as}^{c}$ in each experiment is calculated. Obviously, the remaining resources, after accounting for fragmentation if any, are utilized for hosting VMs of $\mathbb{R}$ and $\mathbb{R}^{c}$. Subsequently, average standard deviation of placement variance from 50% for all the active PMs in an experiment is calculated. Ten such experiments are conducted, in order to conclude the findings,. The same is presented in Figure. 8 for timeslots τ_1, τ_2 and τ_3 respectively. When considering placements across timeslots, as expected standard deviation increases for AIATP due to additional requests getting placed onto same PM. Standard deviation of AIATP is still better and predictable than those of FFD, FFI and FCFS which are erratic. This parameter helps CSPs in reducing future migrations and thereby reducing violations of SLA, probably committed for a higher price, to high-priority users ($\mathbb{R}_{as}^{c}$).

Figure 7. Migration needed to co-allocate VMs of requests having intercommunicating Tasks

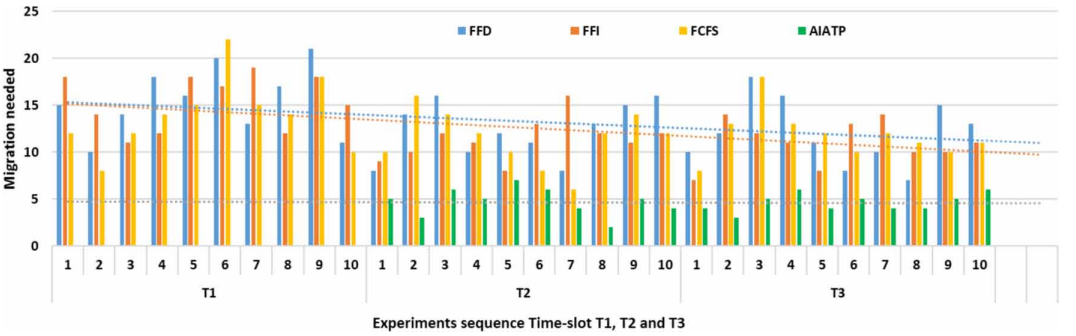
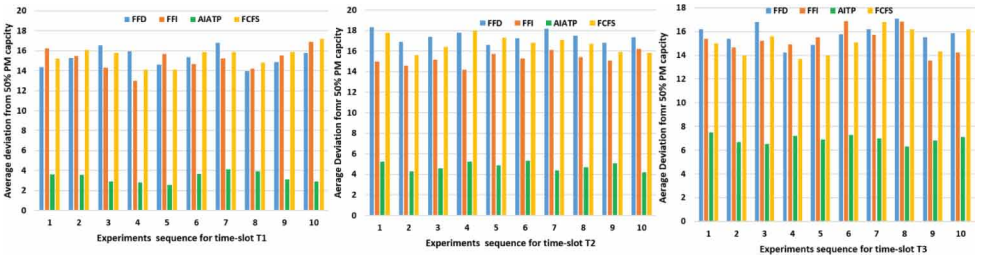


Figure 8. Average deviation from 50% capacity of individual PMs in τ_1, τ_2 and τ_3



CONCLUSION AND FUTURE WORK

In this paper, a novel virtual machine placement algorithm AIATP for autoscaling cloud applications that have intercommunicating tasks is presented. This algorithm has considered applications request encompassing one or more VMs for placement rather than individual VM, a practice conventionally used. Using placement knowledge of earlier tasks belonging to the current request, it has been shown that AIATP can place all the intercommunicating tasks onto same PM and/or on to network neighborhood PMs, thereby resulting in increased application performance by reducing the network latency between these tasks. The proposed AIATP has been employed as an extension to the scheduling polices of CloudSimPlus simulator and evaluated for metrics such as PM utilization, fragmentation, migrations and balanced PMs. It is to be noted that the reduced value for fragmentation and migrations along with increased value for PM utilization implies, increased operational efficiency for DC. Experimental results have demonstrated that AIATP outperforms other considered algorithms namely, FFD, FFI and FCFS for the identified metrics. It has been shown that use of AIATP gives increased application performance for users and also reduces the operational costs of CSPs.

The effect of AIATP on DC power consumption, analysis of various security issues related to placement of tasks and challenges related to integration of latest technologies like Internet of Things (IoT) and Fog/Edge computing with the cloud are of interest for future study.

ACKNOWLEDGMENT

I would like to express my gratitude to Dr Subrata Chattopadhyay and Dr P V Anandamohan for their contribution towards the preparation of this report.

REFERENCES

Al-Dulaimy, A., Itani, W., Zantout, R., & Zekri, A. (2018). *Type-Aware Virtual Machine Management for Energy Efficient Cloud Data Centers. Journal of Sustainable Computing: Informatics and Systems* , 19.

Barclay. (2016). *Overview of autoscale.* https://aws.amazon.com/blogs/compute/fleet-management-made-easy-with-auto-scaling/

Barkat, A., Diniz, A., & Ikken, S. (2015). Open Source Solutions for Building IaaS Clouds. Scalable Computing Practice and Experience, 187-204. doi:10.12694/scpe.v16i2.1089

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, *28*(5), 755–768. doi:10.1016/j.future.2011.04.017

Boucher, R., Jr., Wallace, G., & Wren, B. (2018). O*verview of autoscale*. https://docs.microsoft.com/en-us/azure/azure-monitor/platform/autoscale-overview

Chen, J., He, Q., Ye, D., Chen, W., Xiang, Y., Chiew, K., & Zhu, L. (2016). Joint affinity aware grouping and virtual machine placement. *Microprocessors and Microsystems*, *52*, 365–380. doi:10.1016/j.micpro.2016.12.006

Cui, L., Tso, F., Pezaros, P., Jia, W., & Zhao, W. (2017). PLAN: Joint policy- and network-aware VM management for cloud data centers. *IEEE Transactions on Parallel and Distributed Systems*, *28*(4), 1163–1175. doi:10.1109/TPDS.2016.2604811

Gill, S., & Buyya, R. (2018). Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: From fundamental to autonomic offering. *Journal of Grid Computing*, 1–33.

Heidelberg, B. (2006). Bin-packing In Combinatorial Optimization. *Algorithms and Combinatorics. Springer*, *21*, 426–441. doi:10.1007/3-540-29297-7_18

Li, K., Wu, J., & Blaisse, A. (2013). Elasticity-aware Virtual Machine Placement for Cloud Datacenters. *IEEE International Conference on Cloud Networking (CloudNet)*. doi:10.1109/CloudNet.2013.6710563

Lopez-Pires, F., & Baran, B. (2015). A many-objective optimization framework for virtualized datacenters. *Proceedings of Fifth International Conference Could Computing and Service Science*, 439-450. doi:10.5220/0005434604390450

López-Pires, F., Barán, B., Benítez, L., Zalimben, S., & Amarilla, A. (2018). A., Virtual machine placement for elastic infrastructures in overbooked cloud computing datacenters under uncertainty. *Future Generation Computer Systems*, *79*(Part 3), 830–848. doi:10.1016/j.future.2017.09.021

Manoel, C. S. F., Raysa, L. O., Claudio, C. M., Pedro, R. M. I., & Mario, M. F. (2017). CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. *Proceedings of IFIP/IEEE Symposium on Integrated Network and Service Management*.

Manvi, S. S., & Shyam, G. K. (2014). Resource management for infra-structure as a service (iaas) in cloud computing. A survey. *Journal of Network and Computer Applications*, *41*, 424–440. doi:10.1016/j.jnca.2013.10.004

Meng, X., Isci, C., Kepart, J., Zhang, L., Bouillet, E., & Pendarakis, D. (2013). Efficient resource provisioning in compute clouds via VM multiplexing. *Proceedings of Seventh International Conference on Autonomous Computing*, 215-228.

Meng, X., Pappas, V., & Zhang, L. (2010). Improving the scalability of data center networks with traffic-aware virtual machine placement. *Proceedings - IEEE INFOCOM*, *2010*, 1–9. doi:10.1109/INFCOM.2010.5461930

Rahman, M., & Graham, P. (2017). Compatibility-based static VM placement minimizing interference. *Journal of Network and Computer Applications*, *84*, 68–81. doi:10.1016/j.jnca.2017.02.004

Shi, J., Dong, F., Zhang, J., Luo, J., & Ding, D. (2015). Two-phase online virtual machine placement in heterogeneous cloud data center. *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1369–1374 doi:10.1109/SMC.2015.243

Singh, A., & Inderveer, C. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing, 14*(2), 217-264.

Stergiou, C., Psannis, K., Kim, B., & Gupta, B. (2018). Secure integration of IoT and Cloud Computing. *Future Generation Computer Systems*, *78*, 964–975. doi:10.1016/j.future.2016.11.031

Yokoyama, D., Schulze, B., Kloh, H., Bandini, M., & Rebello, V. (2017). Affinity aware scheduling model of cluster nodes in private clouds. *Journal of Network and Computer Applications*, *95*, 94–104. doi:10.1016/j.jnca.2017.08.001

Zhaolong, G., Shingo, Y., & Gupta, B. B. (2017). *Analysis of Various Security Issues and Challenges in Cloud Computing Environment: A Survey. In Identity Theft: Breakthroughs in Research and Practice*. IGI Global.

Zheng, Q., Li, R., Li, X., Shah, N., Zhang, J., Tian, F., Chao, K. M., & Li, J. (2016). Virtual machine consolidated placement based on multi-objective biogeography-based optimization. *Future Generation Computer Systems*, *54*, 95–122. doi:10.1016/j.future.2015.02.010

ENDNOTE

[1] Tasks and VM are interchangeably used.

Sridharan R. has obtained M.Sc, degree from Bharatidasan University, India in 1992 and M.S degree from Birla Institute of Technology, Pilani, India in 1997. He was awarded PhD by National Institute of Technology, Tiruchirappalli in 2020. He has been with Centre for Development of Advanced Computing, Bangalore in R&D division. His research interests are in the area of Parallel and Distributed computing, Cloud and Quantum Computing. He has published several papers in these areas in international conferences.

Domnic S. received the B.Sc. and M.C.A degrees from Bharathidasan University, Tiruchirappalli, India, in 1998 and 2001, respectively, and the Ph.D. degree from Gandhigram Rural University, Dindigul, India, in 2008. He is currently an Associate Professor with the Department of Computer Applications, National Institute of Technology at Tiruchirappalli, Tiruchirappalli. His current research interests are data compression, image/video processing, and information retrieval.