GLAP: Distributed Dynamic Workload Consolidation through Gossip-based Learning

Mansour Khelghatdoust†‡1, Vincent Gramoli†‡2, Daniel Sun‡3
†The University of Sydney, Australia
‡NICTA/DATA61-CSIRO, Sydney, Australia
1mansour.khelghatdoust@nicta.com.au, 2vincent.gramoli@sydney.edu.au, 3daniel.sun@nicta.com.au

Abstract*—Dynamic virtual machine consolidation (DVMC) using live migration is one of the most promising solutions to reduce energy consumption in cloud data centers. Distributed DVMC often aggressively consolidates virtual machines (VMs) at the high expense of Service Level Agreement (SLA) of customers due to virtual machines (VMs) workload fluctuations. To alleviate this, static and adaptive threshold algorithms were proposed. However, the former is unable to predict the VMs future resource demands and the latter calculates a fixed threshold value for all physical machines (PMs) while each PM hosts VMs with different workload patterns. Moreover, both methods cannot consolidate VMs to maintain PMs in a long-term safe state. To overcome these problems, we propose a fully distributed and threshold-free DVMC algorithm called, GLAP. We combine Q-Learning with a gossip-based protocol to characterize workload patterns of VMs and take consolidation decisions. We also propose a novel two-phase distributed algorithm by which PMs unify the learned pattern which is vital for efficient execution of the algorithm. Finally, we compare GLAP experimentally against three existing techniques and show that GLAP reduces by from 43% to 78% the number of overloaded PMs under the Google Cluster VMs workload traces.

I. Introduction

Dynamic virtual machine consolidation (DVMC) is the process of reducing the number of active physical machines (PMs) through virtual machine (VM) live migration to diminish energy consumption and improve resource utilization in data centers. Most of the solutions perform on the basis of initial resource requirements defined by VM types. However, VMs often utilize resources much less than their initial allocation. To maximize resource utilization efficiently, cloud providers tend to consolidate VMs based on resource demand. It may adversely impact the SLA because resource utilization of each VM on a PM varies over time [16] and thus PMs can become overloaded and SLA be violated, i.e., the aggregate resource demand from their VMs exceeds their capacity. It causes more VM migrations that in turn deteriorates SLA. To mitigate this deterioration, static and adaptive utilization threshold techniques were proposed [18], [13], [10]. In static techniques, each PM accepts migrating VMs until resource utilization reaches a threshold. However, this technique only considers the current state of VMs and neglects the varying demand of VMs. In adaptive algorithms, a historical trace of VMs resource utilization is monitored by a central PM to calculate a fixed threshold value to be used by all PMs. Yet, this approach is inefficient because the VMs workload patterns of PMs are different from one another and thus each PM requires a dedicated threshold value.

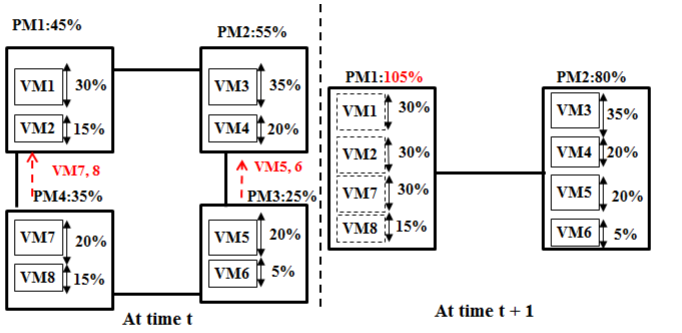


Figure 1: Previous distributed solutions could not cope with the varying load of VMs

Fully distributed DVMC algorithms [8], [9], [11], [17], [20], [23] overcome the deficiencies of centralized [10], [15], [16] or hierarchical [21], [22] approaches. By having PMs making consolidation decisions based on a small part of the data center, they do not suffer from scalability and/or packing efficiency with the increasing number of PMs and VMs [17]. Unfortunately, none of the distributed DVMC solutions adapt to the varying load of VMs. To illustrate the problem, consider Figure 1 that depicts a data center with 4 PMs and 8 VMs and the relationships between PMs. In distributed solutions, PMs accept VMs up to a threshold, say 80%. At round t, PM4 communicates with PM1 and migrates VM7 and VM8 and switch off before the overlay network gets reconfigured. Similarly, PM3 migrates its VMs to PM2 and switches off. At round $t + 1$, the demand of VM2 and VM7 increases and PM1 becomes overloaded. However, PM1 cannot migrate any VM and remains in an overloaded state since its only neighbor (PM2) has reached the threshold. The load variation induces SLA violations. These distributed algorithms switch off a high number of PMs at the expense of overloading most active PMs. In the previous example, one could migrate the VM by starting new PMs, however, this would tend to create several scattered under-utilized PMs which reduces the resource utilization efficiency.

Our contribution is GLAP (Gossip Learning Resource Allocation Protocol), the first fully distributed DVMC algorithm considering variable resource demand of VMs. The key idea is a gossip-based learning strategy to predict VM load variations. We devise an unstructured gossip-based protocol as well as a Q-Learning technique by which workload patterns of VMs are characterized. Using both techniques, each PM cooperates with its fellow PMs to improve its status to a new state incrementally moving towards the consolidation goals with minor SLA violations and without sacrificing scalability.

To reduce the chance of SLA violation, we propose a threshold-free technique, using Q-Learning, that considers subsequent load state of VMs. Q-Learning consists of States (S), Actions (A), and Rewards (R). States are defined as calibrated PMs load state. An action is moving out/migrating any specific VM. Also, each VM has a calibrated load state considering time-varying load of VMs. We design two reward systems one is to encourage PMs to migrate their VMs to switch off mode and the other assists PMs in accepting or rejecting migrating VMs to avoid moving to an overloaded state. In fact, on the basis of the PM state, GLAP predicts whether adding a specific VM would cause the PM to move to an overloaded state immediately or in the future. If so, it rejects the VM to ensure that the PM stays in a desirable state for a longer period of time. It is vital for all PMs to eventually own the identical set of Q-values as a global knowledge of the time-varying load of VMs to take appropriate consolidation decisions. Using a central manager to build Q-values becomes, however, a bottleneck.

To preserve scalability, we also implement a novel two-phase (learning, aggregation) distributed learning protocol that computes the Q-values. In the learning phase, PMs locally calculate Q-values and in the aggregation phase, through a gossip-based protocol, PMs converge to their own unified values. Finally, through a gossip-based protocol, each PM periodically connects with one of its neighbors randomly and migrate its VMs according to Q-Learning system. This process is repeated continuously by all PMs to incrementally switch off unused PMs.

Finally, we conducted extensive experiments using the Google Cluster VMs traces [12]. To this end, we augmented an existing simulator of fully decentralized systems, PeerSim [1]. Under various workloads, we compared GLAP with two well-known distributed consolidation protocols, GRMP [8] and EcoCloud [9], and a centralized consolidation protocol, PABFD [10]. The results show that GLAP, as an autonomous distributed DVMC, addresses the aforementioned problem by not sacrificing SLA for reducing the number of active PMs. Our results show that GLAP reduces the number of overloaded PMs in EcoCloud, GRMP and PABFD by 43%, 78% and 73%, respectively.

The rest of the paper is organized as follows. In section II, we present the related work. In section III, we explain the system model. Our solution is described in section IV in detail. Section V evaluates the performance and present a comparison with other approaches. And finally, section VI briefly states a conclusion and the future work.

II. Background and Related Work

Q-Learning is a reinforcement learning technique employed in many research areas [14]. It is used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). It attains an optimal policy by learning an action-value function that finally gives the expected utility of taking a given action in a given state. The strength of Q-learning is that, unlike MDP, it does not require a model or prior knowledge of the environment. More precisely, it consists of an agent, states S, a set of actions per state A, and a reward system R. The Q-Learning runs several error-and-trial iterations with a dynamic environment to obtain the optimal solution. After each iteration the Q-value is updated according to the following formula:

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= (1-\alpha)Q_t(s_t, a_t) \\ &\quad +\alpha\big(R + \gamma * max_a \; Q_t(s_{t+1}, a_t)\big) \end{aligned}$$

where R is the reward observed after performing action a_t in s_t and γ is a discount factor that determines the importance of future rewards. A factor of zero causes the agent to only consider the current rewards, while a factor approaching one makes it strive for a long-term high reward. The learning rate α is the rate at which the new information overwrites the old one. It takes a value between zero and one such that a value one indicates a deterministic action in the sense that it only considers the latest value while a value between zero and one shows a stochastic behavior taking into account both the latest action and all the previous taken actions for the current state.

The dynamic VM consolidation has been well studied in the literature. However, most algorithms are centralized solutions and few are decentralized. Marzolla et al. [19] and Wuhib et al. [8] propose two fully decentralized gossip based solutions following aggressive approach. However, [8] formulate the problem as multi-dimensional bin packing and do not consider load variation of VMs and [19] does not consider neither multi-resources nor workload changes. Yazir et al. [20] propose distributing migration decisions to each host, but hosts should have global knowledge of all hosts as well as require a performance model of the application, and do not consider SLA performance. Authors in [17] propose a fully decentralized algorithm based on unstructured P2P overlay networks. The system employs a dynamic topology which is built by periodically and randomly exchanging neighborhood information among the PMs. In addition, it considers cost of migration but unlike our work it does not take into account load of VMs and does not ensure SLA well. The work explained in [11] organizes PMs into a hypercube structure in order to scale up and down as resources are either added or removed in response to changes in the number of provisioned instances. They follow a static gradual threshold approach and does not consider the load fluctuation of VMs. Mastroianni et al. [9] proposed a probabilistic gradual threshold based algorithm for placement and migration of VM instances. When a VM request arrives, it is broadcast to all the PMs and they respond to the coordinator if they can accept the request based on a Bernoulli trial. Although the migrations decisions are made based on local information, it still relies on a central manager for the coordination of PMs and hence is not scalable.

Bloglazov et al. [10] used a power aware best fit decreasing for initial VM placements and upper lower threshold based consolidation algorithm to migrate VMs if violation of resource utilization occurs. They compared several methods for capturing dynamic workload of VMs to determine an appropriate upper threshold such as Median Absolute Deviation (MAD), Inter-quartile Range (IQR), and Robust Local Regression. However, it only consider CPU as resource and the most important it is a centralized approach. Secron [16] is a centralized and static threshold based algorithm to prevent

CPUs host from reaching 100% utilization that leads to performance degradation. However, setting static thresholds are not efficient in which different types of applications may run on a host. Farahnakian et al. [15] used reinforcement technique for dynamic VM consolidation. However, they do not consider utilization dynamism of each VM but instead they look at the arrival and departure rate of VMs to make migration decisions. In addition it is a centralized approach and thus is not scalable. The same authors in [18] propose a utilization prediction aware algorithm for dynamic consolidation of VMs using K-nearest neighbor regression based model. However, it is centralized and needs a global manager to have knowledge of resources utilization of all VMs during time and hence is not scalable.

A gossip-based protocol commonly relies on random peer selection and local information and runs at each node in a round-based fashion with limited communication/computation per round [3]. Several gossip-based protocols, like Cyclon [6], provide a random and dynamic sampling of nodes, and in our cases PMs. Other gossip-based protocols were used in the context of resource allocation to partition the distributed resources into groups [5], [4], but not for consolidation.

III. System Model

We model a cloud data center as a set of N machines that are interconnected by a data center network. We deem that each PM has a CPU, Memory, and Network interface. The software layer of the system comprises of four components which are deployed on any participating PM (see Figure 2). VM monitor (VMM) is in charge of profiling total resources utilization of PM as well as monitoring variable resources demand of VM and resizing VMs according to their resource needs. Further, it serves GLAP components by sharing PM and VMs information as requested. Our protocol, GLAP, comprises of three components called Cyclon, Gossip Learning, and Gossip Consolidation. Gossip Learning component has two duties. Firstly, it executes an algorithm to build Q-values locally and secondly, through a gossip process, it ensures that PMs own identical Q-values. Finally, Gossip Consolidation component performs consolidation through gossip protocol. It decides when and which VMs should be migrated out and also decides when and which VMs suggested by other PMs should be accepted to be migrated in. As it can be seen, there is no centralized PM in the architecture of our data center.

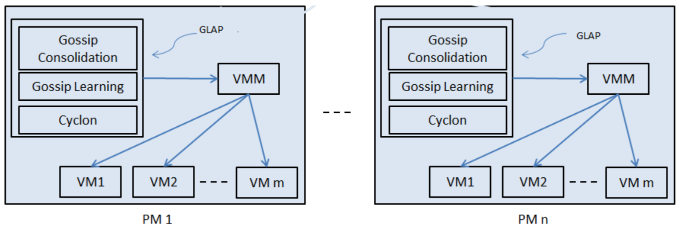


Figure 2: System Architecture

IV. Solution

We describe the algorithm in three sections. We explain how we model Q-Learning components including states, actions, and two reward systems. Then, we propose a novel two phase algorithm by which PMs train and unify Q-values in a decentralized manner. Finally, we propose distributed DVMC algorithm in which PMs using the Q-values consolidate VMs.

A. Construction and usage of Q-Learning in Cloud

A usual MDP model requires 4-tuple input (States (S), Actions (A), Transition Probabilities (P), and Rewards (R)). However, since Q-Learning is model free, knowing P is not required. We define two sets of Q-values, one is used for moving out VMs and the other for making decision to either accept or reject migrating VMs. We devote the former by ϕ_p^{out} and the latter by ϕ_p^{in} for node p.

States(S) and Actions(A):

We define states as PMs load state (PM-State) and actions as VMs load state (VM-State) which can be thought of as migration of a VM in a certain state. In cloud environment, there are different resources (CPU, Memory, IO, Network), each with variable and different workload demand. Thus, workload PMs and VMs are multi-attribute. We calibrate states and actions on the basis of average resource utilization degree of PMs and VMs respectively in order to limit to a finite number of states and actions. Consider a set of n resources $M = \{m_1, m_2, m_3, ..., m_n\}$ in the cloud system and L resource utilization level $L = \{l_1, l_2, l_3, ..., l_n\}$. The maximum total number of states of PMs and VMs are the same and is equal to the Cartesian product of the two sets $|L| \times |M|$. For example, if $M = \{CPU,\ Memory\}$ and $L = \{High,\ Medium,\ Low\}$, total number of possible states is $3^2 = 9$.

We consider 2 resources (CPU and Memory) and 9 resource utilization levels with thresholds used to distinguish between them. $x_p^{au}(t)$ indicates to the average utilization x at time t.

$$S, A = \begin{cases} \textit{Low} & x_p^{au}(t) \leq 0.2 \\ \textit{Medium} & 0.2 < x_p^{au}(t) \leq 0.4 \\ \textit{High} & 0.4 < x_p^{au}(t) \leq 0.5 \\ \textit{xHigh} & 0.5 < x_p^{au}(t) \leq 0.6 \\ \textit{2xHigh} & 0.6 < x_p^{au}(t) \leq 0.7 \\ \textit{3xHigh} & 0.7 < x_p^{au}(t) \leq 0.8 \\ \textit{4xHigh} & 0.8 < x_p^{au}(t) \leq 0.9 \\ \textit{5xHigh} & 0.9 < x_p^{au}(t) < 1 \\ \textit{Overload} & x_p^{au}(t) = 1 \end{cases}$$

For example, if there is a VM with average CPU and memory demand 0.85 and 0.56 respectively, then it indicates an action (*4xHigh, xHigh*). If we assume that the PM includes another VM with specification 0.1 and 0.2 then the PM's state is measured as an aggregation of average resource utilization of VMs and equals to (*5xHigh, 3xHigh*).

Reward (R):

Rewards are incentives that are given to a PM after performing an action a $\in$ A (VM live migration). In fact, rewards are given to PMs to persuade them to perform VMs migrations to consolidate VMs to as few PMs as possible. To this end, we assign rewards for doing actions. PMs are acting as either sender to migrate their own VMs to switch to sleep mode or recipient to decide whether to accept or reject the suggested VM. Therefore, we design two different incentive reward systems called reward *out* and reward *in*.

Reward out: In sender mode, if the state is overload, the reward system encourages the PM to move from heavily loaded state to a lightly loaded state to eliminate SLA violation such that it imposes minimum number of migrations. While a PM with any other state should migrate its VMs to switch to sleep mode. Usually the PM with lower resource utilization migrates its VMs to the other PM so that it can switch off with less number of migrations. However, often a PM can migrate a subset of VMs due to the filled capacity of destination PM. Thus, the sender remains active and may become a recipient when deals with another PM. Such fact leads to the several filling and emptying of PMs which results in redundant number of migrations. To mitigate this, the reward system encourages PMs to aggressively reduce their resource utilization so that they move to sleep mode earlier. Therefore, any transition to a state with a lower resource utilization is given higher reward. Let $R_{out} = \{r_L, r_M, r_H, ..., r_O\}\ \forall r \in R_{out}\ ,\ r > 0$ be a set of reward values for states when PM is sender then:

$$r_L > r_M > r_H > ... > r_O$$

Reward in: In recipient mode, the important factor is preventing SLA violations. It occurs when a PM moves to an overload state after acceptance of VM. On the other hand, PMs should be avaricious and accept as many VMs as possible to be able to maximize their resource utilization. However, this in turn increases the probability of moving to an overload state. To capture such contradiction, we divide level state of PMs and VMs into several smaller scales as shown before. We give a positive reward to PM for any action (live migration) that transit the state of PM to a state towards overload state (but not overload state itself). However, transition to an overload state, given a negative reward. The final Q-value of state and action is the resultant of several training sessions according to the formula 1. If the Q-value of $(s_i\ , a_1)$ is less than zero, the suggested VM is rejected otherwise accepted. More precisely, if the Q-value is negative, accepting action a_1 (VM) when PM is in state s_i, very likely ends in an overload state immediately or in the near future and thus should be avoided. In fact, the smaller negative reward value, the less probability of producing SLA violations and inefficient resource utilization.

Let R_{in} = $\{r_L, r_M, r_H, ..., r_O\}$ be a set of reward values for states when PM is in destination mode then:

$$\{r_L, r_M, r_H, ..., r_{5xH} > 0\ ; r_O \ll 0\}$$

Note for both *out* and *in*, the total reward of any transition from s to $\acute{s}$ is aggregation rewards of each resource.

Optimal Action Selection:

The optimal action determination finds an action for each certain state to maximize the cumulative function of expected rewards. We define two action determination functions for *out* and *in* Q-values. Let π_{out} denotes the function returns the most suitable available VM when PM p is in sender mode (i.e., the greatest value among available actions for state s_p from ϕ_p^{out}, to move towards a lightly loaded state or to move towards emptying the PM). $V_p(t)$ is a set of available VMs within PM p at time t.

$$\pi_{out}(s_p(t)) = \arg\max_a\ (\phi_p^{out}(s_p, a)), a \in V_p(t)$$

Assume q to be a destination PM with current state $s_q(t)$ and let π_{in} denotes the function to decide whether to accept or reject action a. This function rejects action a if it finds that very likely PM q moves to an overloaded state in the future after this transition. This is achieved by looking at ϕ_q^{in} and if the Q-value of certain state and action is less than zero.

$$\pi_{in}(a) = \begin{cases} 1 & ; \phi_q^{in}(s_q, a) \geq 0 \\ -1 & ; \phi_q^{in}(s_q, a) < 0 \end{cases}$$

B. Gossip Learning Component

To achieve the consolidation goals, it is vital for the algorithm to make efficient live migration decisions which among other factors strongly depends on characterizing workload variation of VMs. In other words, for any state-action pair, we should carefully evaluate and quantify how worth it is to do a certain action (VM live migration) when PM is in a specific load state. Technically, we need a component to calculate Q-values in our Q-Learning algorithm with respect to the reward systems that we have explained in the previous section. Although DVMC is a continuous service, the learning process does not need to execute endlessly. In fact, according to Figure 2, Cyclon and workload consolidation components run continuously while learning component runs as required by a predefined policy e.g., if the arrival and departure rates of VMs exceed a threshold compared to the last learning time or based on a fixed time interval.

A naive approach is to utilize a dedicated server. However, it is in contradiction with P2P overlay networks. We propose a decentralized gossip based protocol by which PMs cooperatively measure Q-values and exchange among themselves to obtain identical ones. The protocol consists of two phases, called learning and aggregation, executed consecutively. Once PMs are triggered to run the algorithm by an oracle, in the learning phase, each PM locally simulates consolidation process and trains Q-values until a predefined period, and in the aggregation phase, they exchange values using a merge function so that PMs converge to the unified ones.

The learning phase is executed within PMs. To eliminate any impact on collocating VMs in highly loaded PMs, only PMs with resource utilization less than a threshold (e.g., 80%) execute it locally. Such PM needs to collect VMs profiles of only one neighbor and aggregate with local profiles. To cover highly loaded states, it may need to duplicate profiles. Then, each PM resembles both sender and destination types by assigning a subset of VMs profiles randomly to each one. Then, it simulates the consolidation process for both types of PMs by removing VM from the one and adding to the another and updating Q-values using Equation (1). Such process is repeated several times (Algorithm1).

However, consideration of only current resources demand of VMs to determine state and action is unsuitable for an environment with dynamic and unpredictable workloads. To capture VMs workload variation more efficiently, we remark both **current** and **average** resource demands of VMs that have been monitored up to now in order to specify states and actions of PMs and VMs. To calculate the average demand, each VM piggybacks a tuple $\{c, v\}$ in which c represents the number of times the resource demand is monitored and v indicates the average observed demands. In the next profiling time, the new average can be calculated simply by $((c * v) + d(t))/(c + 1)$ where $d(t)$ is the current resource demand.

The state of a PM before performing an action as well as the state of migrating VM are calculated according to the **average** VMs demand while the **current** VMs demand are used to calculate the state of a PM after performing an action. In Figure 3, the average resource demand VM1 is 41% which is mapped to *High*. The state of vPM1 before migrating VM1 is the aggregation of its VMs average demands (79% = *3xHigh*). While the state after migrating VM1 is calculated as aggregation of remaining current demand of its VMs (50% = *High*). Therefore, according to Formula 1, to calculate $Q_{t+1}(3xHigh, High)$, in vPM1, s_t = *3xHigh*, s_{t+1} = *High*, a_t = *High*, and $Q_t(3xHigh, High)$ and $Q_t(High, High)$ are extracted from out map and the reward value R for moving to *High* state is obtained from R_{out}. The same rule is applied for vPM2 as destination PM.

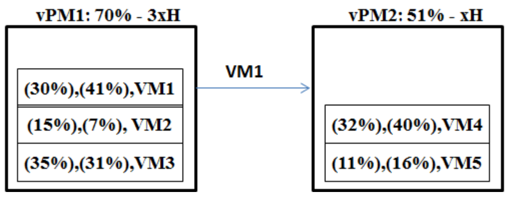


Figure 3: Learning Example. In each VM, the left number indicates the current demand and the right represents the average demand until now.

At the end of this phase, beside PMs without any Q-value (Due to lack of enough resources to execute the algorithm), others may possess different Q-values while it is essential for all PMs to own identical ones. To this end, we execute aggregation phase of the protocol (Algorithm2). This is a gossip protocol in which in each round, every PM exchanges its Q-values with one randomly selected neighbor and updates values via a merge function. More precisely, if a state-action pair exists in both PMs, the new value is calculated as the average value but if the pair is in only one PM, the other just adds it to its Q-value list. This process is repeated several rounds until PMs converge to the unique Q-values.

C. Analysis of the convergence of Q-value distribution

In this section, we show that the gossip-based process repeatedly aggregates Q-values that eventually converge to a normal distribution. To this end, we make the following assumption: the Q-values obtained during the aggregation phase by a node comes from a random node and are independent. Note that we do not require the distribution of selected nodes to be uniform or that Q-values are identical.

***Theorem** 1:* The random variable of the Q-value at round n converges to a normal distribution as n tends to ∞.

Proof: We consider $n + 1$ rounds numbered from 0 to n. Let x_i be the Q-value at round i, $0 \le i \le \infty$. Since all

Algorithm 1 Learning Phase

```
procedure LOCALTRAIN($p$)
    while triggered do
        if resource utilization $p \le$ threshold then
            vms ← collect profiles of one neighbor
            vms ← aggregate with local VMs profiles
            vms ← duplicate vms if required
            for k times do
                vmss ⊂ vms;                              ▷ sender vms
                vmst ⊂ vms;                              ▷ target vms
                vm ← select random vm of vmss to migrate;
                updateOUT(vmss,vm);                      ▷ according to (1)
                updateIN(vmst,vm);                       ▷ according to (1)
            end for
        end if
        sleep until end of Round
    end while
end procedure
```

Algorithm 2 Aggregation Phase

Require: Any node p has map of Q-values $\phi_p^{io} \leftarrow \phi_p^{in} \cup \phi_p^{out}$

```
while triggered do                                   ▷ Active thread
    q = selectPeer();
    send(q , $\phi_p^{io}$); $\phi_q^{io}$ = receive(q);
    UPDATE($\phi_p^{io}$ , $\phi_q^{io}$);
    sleep until end of round
end while

while triggered do                                   ▷ Passive thread
    $\phi_q^{io}$ = receive(q); send(q , $\phi_p^{io}$);
    UPDATE($\phi_p^{io}$ , $\phi_q^{io}$);
end while

procedure UPDATE($\phi_p^{io}$ , $\phi_q^{io}$)
    for each {s,a} ∈ $\phi_p^{io} \cup \phi_q^{io}$ do
        if {s,a} exists in both $\phi_q^{io}$ and $\phi_p^{io}$ then
            calculate average two values, set new values
        else
            add {s,a} and value to the other map
        end if
    end for
end procedure
```

Q-values are updated in parallel and independently, we only need to focus on independent x_i values as follows.

Let X be the random variable of the Q-value at round n.

First, we have $X = x_0$. For $n = 1$, $X = \frac{x_0+x_1}{2}$. For $n = 2$, $X = \frac{\frac{x_0+x_1}{2}+x_2}{2} = \frac{x_0}{4} + \frac{x_1}{4} + \frac{x_2}{2}$. To generalize this, note that $\frac{1}{2^n} + \frac{1}{2^n} + \frac{1}{2^{n-1}} + \cdots + \frac{1}{2} = 1$, hence we obtain $X = \frac{x_0}{2^n} + \frac{x_1}{2^n} + \frac{x_2}{2^{n-1}} + \cdots + \frac{x_n}{2}$.

Let u_x be the expectation and σ_x^2 be the variance of x_i respectively, because of independent variables.

$$X - u_x = \frac{x_0 - u_x}{2^n} + \frac{x_1 - u_x}{2^n} + \frac{x_2 - u_x}{2^{n-1}} + \cdots + \frac{x_n - u_x}{2}.$$

Let y_i, $0 \leq i \leq \infty$, be random variables such that $y_i = \frac{x_i - u_x}{2^{n-i}}$. The expectation $u_{y,i}$ and the variance $\sigma^2_{y,i}$ of y_i are, respectively, $u_{y,i} = 0$ and $\sigma^2_{y,i} = \frac{\sigma^2_x}{2^{2(n-i)}}$, given any fixed n and i.

Then according to Lindeberg or Lyapunov Central Limit Theorem, the summation of independent random variables, not necessarily identically distributed, converges in distribution and pre. Since y_i are the random variables in this kind, $X - u_x$ converges to the normal distribution. ■

Note that in reality some of the Q-values may not be random because they are identical or null. However, the more random the Q-values in each round, the closer to expectation the final result could be. Also since we have known the distribution of final results, we can optimally decide how many rounds are needed at least to assure a satisfying convergence.

D. The Gossip Workload Consolidation Component

The consolidation component is built on top of the two other components (see Figure 2). Each PM runs two threads called active and passive threads and follows the push-pull interaction pattern, where a machine pushes its state to another machine and pulls that machines state. State is meta data information of the last monitored PM's resources utilization (Algorithm 3, Lines 1-10).

If the resource utilization of initiator PM (at least one of the resources) is overloaded, that PM requires to migrate some of its VM(s) to quit of the overload state (Lines 11-13). Otherwise, to consolidate VMs, the PM with totally less current utilized resources is selected as sender PM to migrate out its own VMs to switching to sleep mode (Lines 14-16). The sender PM, using its own current state, looks up the appropriate action from ϕ_p^{out} which is actually the action with the greatest Q-value for that state. Then, it picks a VM that corresponds to the chosen action. If there are several corresponding VMs for the action, the one with the least migration cost is selected. If there is no VM available with the specific action, the round is finished and no migration takes place. After selection of VM, sender PM finds the relevant Q-value from ϕ_p^{in} to check whether the target PM is able to accept this VM or not. If $\phi_p^{in}(s,a) \leq 0$, it means that the target PM cannot accept the VM and the round is finished without migrating VM. The negative value indicates that such VM migration very likely leads to SLA violation of the target PM and should be avoided. The interesting point is that because PMs own identical Q-values and also sender PM is aware of the target PM state, the decision is made in the sender PM on behalf of the target PM to eliminate communication overhead. Further, sender PM checks to ensure that the target PM has enough capacity to place the current demands of VM. In the end, VM migrates and states of both PMs are updated (Lines 18-24). As we mentioned earlier, recalculation of Q-values is done by learning component. In fact, learning component feeds consolidation component. During this process, consolidation component can be configured to either continue using the previous Q-values or pause for a while and resume by using new Q-values. Clearly, it can be implemented through synchronization between two components running within PMs and thus there is no communication overhead or any need for centralized server.

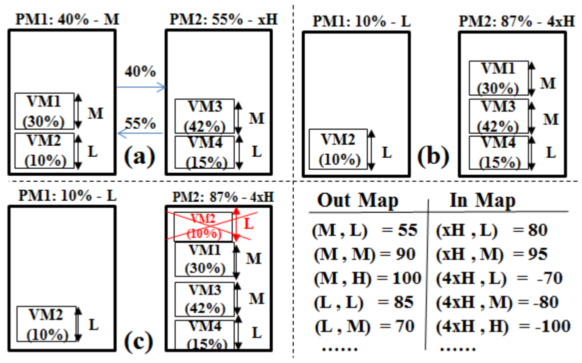


Figure 4: Consolidation Example. **(a)** PM1 exchanges its state with PM2 and it performs as sender because it has lower average resource utilization (40%). **(b)** VM1 migrates from PM1 to PM2 because it has greater Q-value 90 than VM2 with 55. **(c)** VM2 cannot migrate to PM2 because (4xH, L) has a negative Q-value -70 in "*in*-map". The round is finished.

V. Performance Evaluation

In this section, we compare the performance of GLAP against one centralized [10] and two distributed [8], [9] consolidation solutions using Google Cluster VMs traces [12].

A. Simulation Settings

To keep the environment under control so that we provide a stable configuration and execute repeatable experiments we carried out the experiments on a simulated cloud environment which supports running distributed P2P algorithms. We conducted experiments on PeerSim [1], a simulator for modeling large scale P2P networks. We simulated several configurations of cloud data center with 500, 1000, and 2000 nodes and VM-PM workload ratios of 2, 3, and 4 for each data center size. The PMs are modeled as HP ProLiant ML110 G5 servers (2660 MIPS CPU, 4GB memory, 10 GB/s network bandwidth) and the VMs are modeled from EC2 micro instance (500 MIPS CPU, 613 MB memory). At the beginning, VMs are allocated resources based on the demand of VM type, however, during execution they utilize less resources which gives the chance for dynamic consolidation.

The resource utilization trace from Google Cluster VMs are used to drive the VM resource utilization in the simulation [12]. The trace contains utilization of CPU and memory of VMs. We executed each experiment for 720 rounds such that each round mimics 2 minutes to simulate 24 hours and the evaluation metrics are sampled at the end of each round. It can show the efficiency of each protocol in long term how they deal with VMs variable workload. At the beginning, the VMs are randomly allocated to the PMs. For the sake of having fair comparison, such VM-PM mapping is used identically for all different algorithms in each experiment. We repeatedly carried out each experiment for 20 times and extracted the results. For GLAP, we executed 700 more rounds to calculate Q-values beforehand.

To evaluate GLAP, we compared it with two distributed and one centralized workload consolidation algorithms called

Algorithm 3 Consolidation Component

```
Require: Any node P has the following methods & variables:
  • $s_p$, $s_q$, $\phi_p^{in}$ : states p and q, map in
  • findVM($s_p$): find action/vm for state $s_p$
 1: while true do                                   ▷ Active thread
 2:     q ← selectPeer();
 3:     send(q , $s_p(t)$); $s_q(t)$ ← receive(q);
 4:     UPDATESTATE();
 5:     sleep until end of round
 6: end while
 7: while true do                                   ▷ Passive thread
 8:     $s_q(t)$ ← receive(q); send(q , $s_p(t)$);
 9:     UPDATESTATE();
10: end while

11: procedure UPDATESTATE()
12:     if p is in overloaded state then
13:         call MIGRATE() as long as p is overloaded
14:     else if p = $\arg\min_{n\in\{p,q\}} (s_n(t))$ then      ▷ p is sender
15:         call MIGRATE() as long as switch off p
16:     end if
17: end procedure

18: procedure MIGRATE()
19:     a , vm := p.findVM($s_p$)
20:     if $\phi_p^{in}(s_q, a) < 0$ or $vm = \perp$ or no capacity then
21:         return and sleep until end of round;
22:     end if
23:     migrate vm from p to q and update $s_p$ , $s_q$
24: end procedure
```

GRMP [8], EcoCloud [9], and PABFD [10] respectively. GRMP is an aggressive gossip based protocol with a static upper threshold 0.8 while EcoCloud is a gradual probabilistic static upper and lower threshold based protocol with the configuration of (T1 = 0.3 and T2 = 0.8). PABFD is a centralized dynamic threshold based heuristic consolidation algorithm in which a centralized server periodically monitors resources usage of PMs and using global information makes consolidation decisions. It calculates upper threshold by of-fline statistical analysis of historical data collected during the lifetime of VMs. The Median Absolute Deviation (MAD) is used as an estimator of upper threshold value. Continuously, we execute this algorithm for the same time as decentralized algorithms to be able to compare them fairly.

B. Performance metrics

One performance metric is SLA. It is often determined as throughput or response time ensured by applications. However, such characteristics vary for different applications and thus we define SLA violation (SLAV) as percentage of time during which the active hosts have experienced a CPU utilization of 100% (SLAVO) which indicates that some VMs may not be allocated required resource and performance degradation due to live VM migration (SLALM). [10]

$$SLAVO = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{s_i}}{T_{a_i}} \quad , \quad SLALM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}} \tag{1}$$

$$SLAV = SLAVO * SLALM \tag{2}$$

In the above formulas, N is the number of PMs, T_{s_i} is the accumulated time during which the PM i has encountered the CPU utilization of 100% , T_{a_i} is the total time during which the PM i is in active mode, M is the number of VMs, C_{d_j} is the performance degradation of the VM j caused by migration which is estimated as 10% of CPU utilization during all migrations of the VM j [10]. C_{r_j} is the total CPU capacity requested by the VM j during its lifetime.

Energy overhead that each live migration imposes is a metric for comparing the algorithms. We assume that the data center patronages VM live migration which is currently supported by some hypervisor technologies, such as Xen or VMware. The cost of migration is measured as energy overhead it imposes which is defined as the power consumption machine n, multiplied by the migration time τ. The migration time strongly varies with VM's memory size and available transmission bandwidth at the source and destination servers. The power consumption of machine n is modeled as a linear function of its CPU consumption for migration and is repre-sented as P_n^{lm}. Power consumption of machine n when it is in the idle mode is demonstrated as P_n^{idle}. The cost of migration one VM from node i to node j is considered as energy overhead and is calculated with the following formula [2]:

$$E_{i\to j}^{lm} = ((\ P_i^{lm}\ -\ P_i^{idle}\ ) + (\ P_j^{lm}\ -\ P_j^{idle}\ ))\ *\ \tau_{i\to j}^{lm}\ , \tag{3}$$

Packing efficiency is another metric which shows how each algorithm captures energy-performance trade-off. However, only comparing the number of active PMs to the optimal one cannot show such trade-off. Beside, we calculate the number of overloaded PMs to active ones as well to see with which SLA cost, each algorithm consolidates PMs.

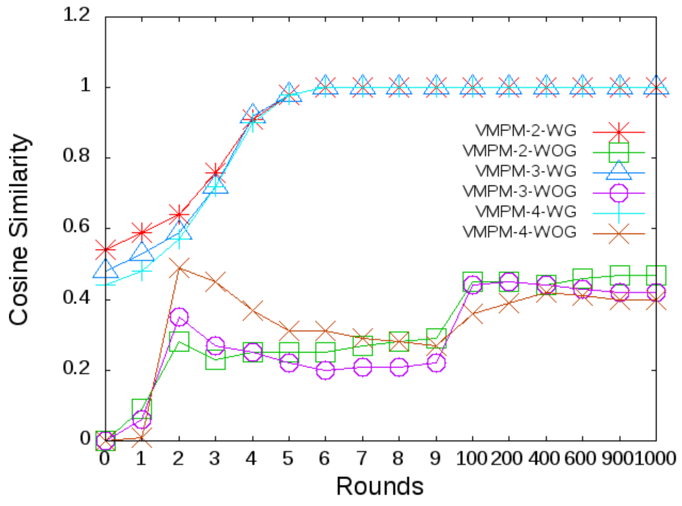
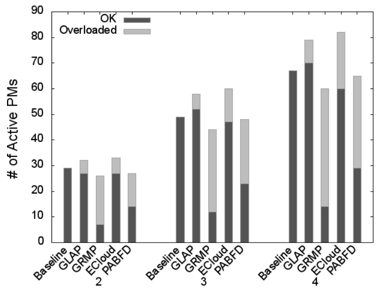
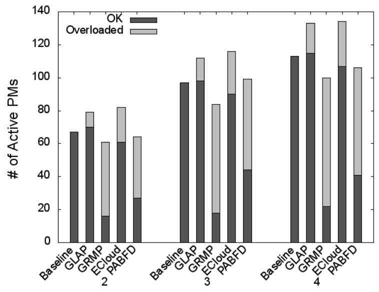


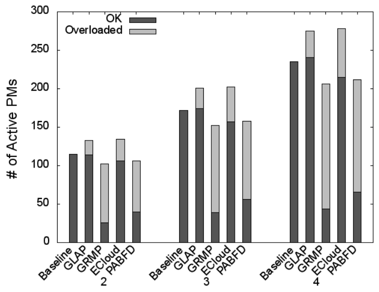
Figure 5: convergence Q-values after learning phase (WOG) and aggregation phase(WG) for VM-PM ratios 2,3, and 4.



(a) 500 nodes

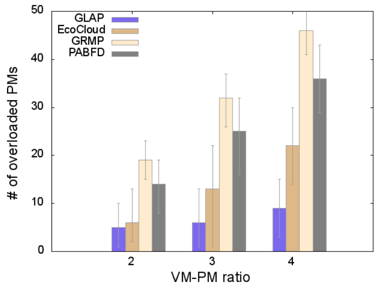


(b) 1000 nodes

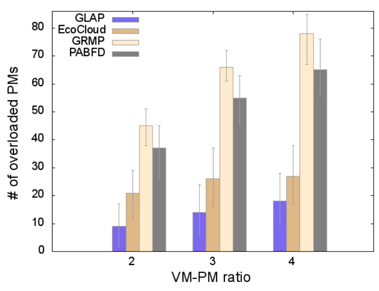


(c) 2000 nodes

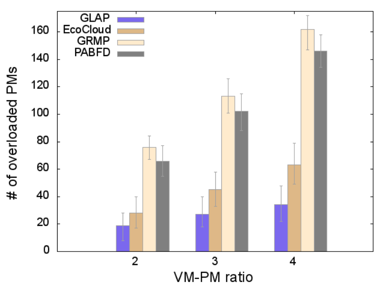
Figure 6: The fraction of overloaded / active PMs with increasing workload ratio and various cluster sizes



(a) 500 nodes

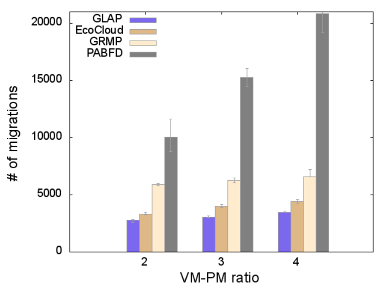


(b) 1000 nodes

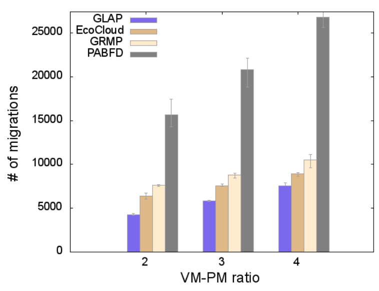


(c) 2000 nodes

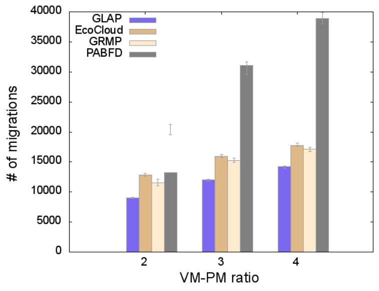
Figure 7: The number of overloaded PMs with increasing workload ratio and various cluster sizes



(a) 500 nodes

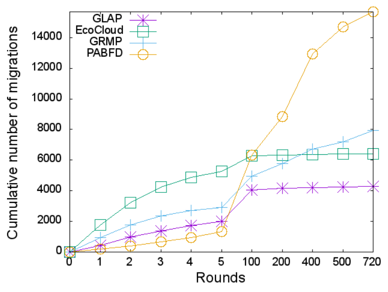


(b) 1000 nodes

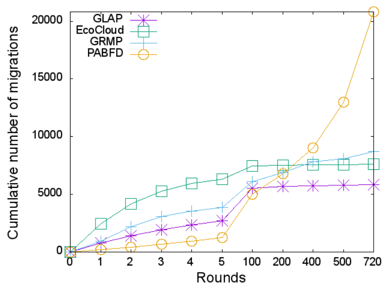


(c) 2000 nodes

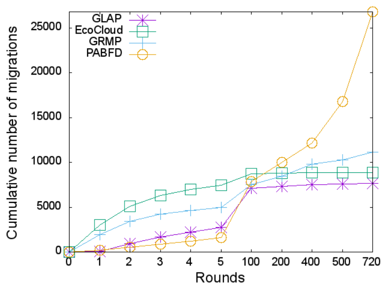
Figure 8: The number of migrations with increasing workload ratio and various cluster sizes



(a) VM-PM workload ratio 2

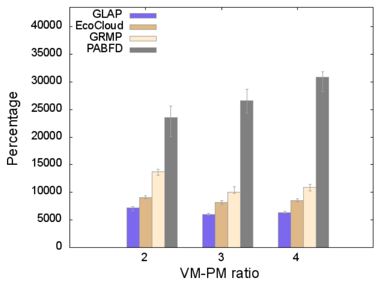


(b) VM-PM workload ratio 3

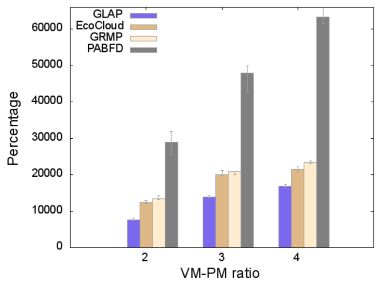


(c) VM-PM workload ratio 4

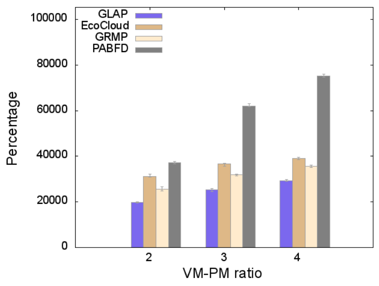
Figure 9: The cumulative number of migrations with increasing workload ratio for 1000 nodes



(a) 500 nodes



(b) 1000 nodes



(c) 2000 nodes

Figure 10: Energy overhead of migrations with increasing workload ratio and various cluster sizes

C. Experimental Results

*1) **Gossip learning component works correctly and Q-values converge rapidly**:* Q-values should be identical for all PMs. To evaluate correctness of the 2 phase gossip learning protocol, we run experiments for 1000 nodes with workload ratios 2, 3, and 4. PMs with up to 50% free CPU are configured to run the algorithm locally to prevent any adversely impact on collocating VMs. We calculate Cosine similarity Q-values of PMs every cycle to observe how they converge towards identical values. In Figure 5, the protocol converges up to 45% after running the learning phase (WOG) for all VM-PM ratios. Then, in aggreagtion phase (WG), the PMs exchange their local values and as it can be seen in Figure 5 that Q-values converge rapidly for all PMs and VM-PM ratios. It shows the importance of the gossip learning protocol to make sure that every PM owns identical Q-values.

*2) **Minimizing the number of active PMs autonomously with much lower overloaded PMs**:* Figure 6 shows how aggressive each algorithm consolidate PMs. We calculated BFD (Best Fit Decreasing) using the VMs resource utilization of the last round to determine a baseline packing without producing any SLA violation. GRMP and PABFD switch off PMs even more than the baseline but at the high expense of SLA violations. While GLAP and EcoCloud keep a bit more number of PMs active than the baseline to impose much less SLA violation. It results that they autonomously consolidate PMs in a wisely manner. The results show that 58% of the PMs running PABFD, 22% of the PMs running EcoCloud and 75% of the PMs running GRMP are overloaded whereas only 12% of the PMs running GLAP get overloaded.

*3) **Minimizing the number of overloaded PMs**:* Figure 7 shows the number of overloaded PMs for all the algorithms. We extracted the value of overloaded PMs at the end of each round in all the executions and calculated the median, the 10th and 90th percentiles in the experiments. As it can be seen, GLAP generates the smallest number of overloaded PMs. However, GRMP shows the worst result since it aggressively consolidates VMs into fewer PMs and switches off more PMs quicker. In fact, It improves power consumption at the high expense of performance degradation. Unlike GRMP, EcoCloud and GLAP, consolidate VMs in a slower slope and thus capture trade-off energy performance efficiently. GLAP outperforms EcoCloud, GRMP, and PABFD for 43%, 78%, and 73% less number of overloaded PMs, respectively. Figures 7 (a), (b), (c), show similar results for other cluster sizes and workload ratios, indicating the stability of the outcomes in various circumstances.

*4) **Minimizing the number of VM migrations**:* Figures 8 and 9 show the number and cumulative number of migrations during 1 day respectively. We measured the median, the 10th and 90th percentiles for this experiment. GLAP imposes the fewest number of migrations while PABFD considerably incurs the highest number of migrations. It stipulates this fact that such heuristic centralized algorithm is not efficient for continuous workload consolidation of PMs. GLAP outperforms EcoCloud, GRMP, and PABFD for 23%, 37%, and 70% less number of migrations respectively. These results confirm that our approach is advantageous for consolidation of VMs, because prediction of workload variation, it considerably reduces the probability of PMs being overloaded, which accordingly eliminates the need for excess migrations. It is noteworthy that with increasing the workload ratio, the total number of migrations increases. Figures 9 shows the cumulative number of migrations of 1000 nodes for three workload ratios of 2, 3, and 4. It can be observed that three distributed algorithms do most of the migrations in early rounds, however PABFD almost follows a linear relationship between time and number of migrations.

*5) **GLAP results in less continuous SLA violation**:* We measured SLA metric for all the combinations of data center sizes and workload ratios. According to Table I, GLAP causes less SLA violation (GLAP $<$ EcoCloud $<$ PABFD $<$ GRMP). With increment of workload, we can observe that SLA violation degree of the protocols increases, Yet, GLAP performs better than the other protocols.

*6) **Minimizing energy overhead migrations**:* As Figure 10 reveals, among the evaluated solutions PABFD consumes the highest energy while GLAP consumes the least. The higher number of migrations does not always lead to the highest energy consumption. For 500 nodes, in GLAP, although workload ratio 4 has more migrations than 2, workload ratio 2 consumes more energy. This is because of the size of VMs and time of migration that directly influence energy consumption.

VI. Conclusion

This paper proposed a gossip-based mechanism for consolidating VMs in a cluster of PMs. It handles VM load fluctuations (which reduces the number of unnecessary migrations) using a Q-Learning technique while remaining scalable and

Table I: SLA Metric for various cluster sizes and workload ratios

	GLAP	EcoCloud	GRMP	PABFD
500-2	0.00011	0.00016	0.27	0.07
500-3	0.00017	0.00045	0.48	0.19
500-4	0.00027	0.00078	0.72	0.36
1000-2	0.00017	0.00018	0.38	0.18
1000-3	0.00035	0.00078	0.61	0.36
1000-4	0.00059	0.00097	0.88	0.57
2000-2	0.00033	0.00076	0.41	0.29
2000-3	0.00066	0.0014	0.84	0.48
2000-4	0.001	0.002	1.24	0.48

not relying on any fixed thresholds. The problem of highly sacrificing SLA for reducing the number of PMs is better addressed than with the state-of-the-art by incurring less SLA violations. As future work, we would like to evaluate our work under bursty workload patterns. In addition, we plan to extend the algorithm to be aware of the network topology such that it will switch off network switches, an important factor of energy consumption in cloud data centers.

References

[1] Montresor, Alberto, and Mark Jelasity. "PeerSim: A scalable P2P simulator." Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on. IEEE, 2009.

[2] Strunk, Anja, and Waltenegus Dargie. "Does live migration of virtual machines cost energy?." Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. IEEE, 2013.

[3] P. Costa, V. Gramoli, M. Jelasity, G. Paolo Jesi, E. Le Merrer, A. Montresor, L. Querzoni. "Exploring the Interdisciplinary Connections of Gossip-based Systems". ACM SIGOPS Operating Systems Review, 41(4):51-60 2007.

[4] Antonio Fernndez, Vincent Gramoli, Ernesto Jimnez, Anne-Marie Kermarrec, Michel Raynal. "Distributed Slicing in Dynamic Systems". IEEE Transactions on Parallel and Distributed Systems, 2015.

[5] Vincent Gramoli, Ymir Vigfusson, Ken Birman, Anne-Marie Kermarrec and Robbert van Renesse. "Slicing Distributed Systems". IEEE Transactions on Computers, 58(11):1444–1455, 2009.

[6] Voulgaris, Spyros, Daniela Gavidia, and Maarten Van Steen. "Cyclon: Inexpensive membership management for unstructured p2p overlays". Journal of Network and Systems Management 13.2 (2005): 197-217.

[7] Chen, Liuhua, Haiying Shen, and Karan Sapra. "Distributed Autonomous Virtual Resource Management in Datacenters Using Finite-Markov Decision Process." Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014.

[8] Wuhib, Fetahi, Rerngvit Yanggratoke, and Rolf Stadler. "Allocating compute and network resources under management objectives in large-scale clouds." Journal of Network and Systems Management 23.1 (2015): 111-136.

[9] Mastroianni, Carlo, Michela Meo, and Giuseppe Papuzzo. "Probabilistic consolidation of virtual machines in self-organizing cloud data centers." IEEE Transactions on Cloud Computing, 1.2 (2013): 215-228.

[10] Beloglazov, Anton, and Rajkumar Buyya. "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers." Concurrency and Computation: Practice and Experience 24.13 (2012): 1397-1420.

[11] Pantazoglou, Michael, Gavriil Tzortzakis, and Alex Delis. "Decentralized and Energy-Efficient Workload Management in Enterprise Clouds."

[12] GoogleTraceWebsite. Google cluster data. https://code.google.com/p/googleclusterdata/.

[13] Beloglazov, Anton, and Rajkumar Buyya. "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints." Parallel and Distributed Systems, IEEE Transactions on 24.7 (2013): 1366-1379.

[14] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research (1996): 237-285.

[15] Farahnakian, Fahimeh, Pasi Liljeberg, and Juha Plosila. "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning." Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on. IEEE, 2014.

[16] Murtazaev, Aziz, and Sangyoon Oh. "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing." IETE Technical Review 28.3 (2011): 212-231.

[17] Feller, Eugen, Christine Morin, and Armel Esnault. "A case for fully decentralized dynamic VM consolidation in clouds." Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012.

[18] Farahnakian, Fahimeh, et al. "Utilization Prediction Aware VM Consolidation Approach for Green Cloud Computing." Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on. IEEE, 2015.

[19] Marzolla, Moreno, Ozalp Babaoglu, and Fabio Panzieri. "Server consolidation in clouds through gossiping." World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a. IEEE, 2011.

[20] Yazir, Yaiz Onat, et al. "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis." Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. Ieee, 2010.

[21] Feller, Eugen, Louis Rilling, and Christine Morin. "Snooze: A scalable and autonomic virtual machine management framework for private clouds." Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society, 2012.

[22] Graubner, Pablo, Matthias Schmidt, and Bernd Freisleben. "Energy-efficient management of virtual machines in eucalyptus." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011.

[23] Quesnel, Flavien, and Adrien Lbre. "Cooperative dynamic scheduling of virtual machines in distributed systems." Euro-Par 2011: Parallel Processing Workshops. Springer Berlin Heidelberg, 2012.