A Case for Fully Decentralized Dynamic VM Consolidation in Clouds

Eugen Feller, Christine Morin, Armel Esnault

A Case for Fully Decentralized Dynamic VM Consolidation in Clouds

Eugen Feller and Christine Morin
INRIA Centre Rennes - Bretagne Atlantique
Campus de Beaulieu, 35042 Rennes Cedex, France
{Eugen.Feller, Christine.Morin}@inria.fr

Armel Esnault
ISTIC, University of Rennes 1
Campus de Beaulieu, 35042 Rennes Cedex, France
Armel.Esnault@irisa.fr

***Abstract*—One way to conserve energy in cloud data centers is to transition idle servers into a power saving state during periods of low utilization. Dynamic virtual machine (VM) consolidation (VMC) algorithms are proposed to create idle times by periodically repacking VMs on the least number of physical machines (PMs). Existing works mostly apply VMC on top of centralized, hierarchical, or ring-based system topologies which result in poor scalability and/or packing efficiency with increasing number of PMs and VMs.**

In this paper, we propose a novel fully decentralized dynamic VMC schema based on an unstructured peer-to-peer (P2P) network of PMs. The proposed schema is validated using three well known VMC algorithms: First-Fit Decreasing (FFD), Sercon, V-MAN, and a novel migration-cost aware ACO-based algorithm. Extensive experiments performed on the Grid'5000 testbed show that once integrated in our fully decentralized VMC schema, traditional VMC algorithms achieve a global packing efficiency very close to a centralized system. Moreover, the system remains scalable with increasing number of PMs and VMs. Finally, the migration-cost aware ACO-based algorithm outperforms FFD and Sercon in the number of released PMs and requires less migrations than FFD and V-MAN.

***Keywords*-Cloud Computing, Dynamic VM Consolidation, Ant Colony Optimization, Unstructured P2P Network, Virtualization**

I. Introduction

Cloud data center providers are now provisioning an increasing number of data centers to handle customers growing resource demands. Such data centers do not only require tremendous energy amounts to power their IT and cooling infrastructures but also impose scalability challenges on their system management frameworks. For example, according to [1] cloud computing services consumed approximately 662 billion kWh of energy in 2007. This is not surprising when considering today's public cloud providers data center scales. For instance, Rackspace which is one of the leading Infrastructure-as-a-Service (IaaS) providers is estimated to host approximately *78 000* servers in 2012 [2].

While the system architecture details of Rackspace and other cloud providers (e.g. Amazon EC2) are not publicly available, several research attempts have been made in the last years on the design of energy-efficient virtual machine (VM) management frameworks in order to ease the creation of private clouds. Energy efficiency is typically obtained by transitioning idle physical machines (PMs) into a power-saving state during periods of low utilization. To facilitate the creation of idle times dynamic VM consolidation (VMC) can be used, whose objective is to pack the VMs on the least number of PMs while minimizing the number of migrations. In contrast to static VMC which assumes empty PMs prior starting the VM to PM assignment, dynamic VMC starts from pre-loaded PMs. Starting from pre-loaded PMs adds a new dimension to the problem as it not only requires the number of PMs to be minimized but also the number of migrations needed in order to arrive to the new VM-PM assignment. In other words, dynamic VMC is a *multi-objective problem* while static VMC is a *single-objective problem*. Throughout the rest of the document we assume dynamic VMC when we refer to VMC.

VMC even in its single-objective variant is an NP-hard combinatorial optimization problem and thus is expensive (in time and space) to compute with increasing numbers of PMs and VMs. Consequently, choosing the appropriate system topology is crucial in order to obtain good scalability and packing efficiency. Packing efficiency is defined as the ratio between the number of released PMs to the total number of PMs. Existing VMC approaches either rely on centralized (e.g. [3]–[5]), hierarchical (e.g. [6], [7]), or ring-based [8] system topologies. While the centralized topology yields to the best packing efficiency, it suffers from poor scalability due to the large amount of considered PMs and VMs. On the other hand, the hierarchical approach achieves better scalability through partitioning of PMs in groups and applying VMC only in the scope of the groups. However, this is achieved at the cost of decreased packing efficiency as the VMC solutions remain local to the groups. Finally, in the ring-based approach PMs are organized in a ring and scheduling is performed event-based upon underload/overload detection by forwarding the VMC requests in the ring. The scalability of this system is limited by the cost of ring-maintenance and its worst-case complexity which requires a full ring traversal to place the VMs.

This paper makes the following two contributions[1]:

- To address the scalability and packing efficiency issues we propose a *novel fully decentralized dynamic VMC*

[1]An earlier version of this document is available as Inria research report №8032 at http://hal.inria.fr/hal-00722245/PDF/RR-8032.pdf

schema based on an unstructured peer-to-peer (P2P) network of PMs. Our VMC schema does not enforce any particular system topology shape. Instead, the topology is periodically modified to form random neighbourhoods of PMs. Considering the computational complexity of VMC we limit its application to the scope of the neighbourhoods. Moreover, the randomness of the system topology facilitates the convergence of the schema towards a global packing efficiency very close to a centralized system by leveraging *traditional centralized VMC algorithms*.

- We adapt our previously proposed ACO-based static VMC algorithm [9] to *minimize the number of VM migrations* in the framework of dynamic VMC. Minimizing the number of migrations required to move from the current state (VM-PM assignment) to the newly computed one is crucial as each migration has a direct impact on the VM performance as well as the amount of data to be transferred over the network.

Both contributions are integrated within a distributed emulator along with three state-of-the-art greedy VMC algorithms, namely FFD, Sercon [10] and V-MAN [11]. Extensive evaluation performed on the Grid'5000 experimental testbed shows that once integrated all the evaluated algorithms achieve a global packing efficiency very similar to a centralized system. Moreover, the migration-cost aware ACO-based algorithm outperforms FFD and Sercon in the number of released PMs and its solutions require less migrations than the ones from FFD and V-MAN (see Table I).

TABLE I: Evaluation Summary

Criteria	Best algorithm	2nd	3rd	4th
#Migrations	Sercon	ACO	V-MAN	FFD
Packing efficiency	V-MAN	ACO	Sercon	FFD

This paper is organized as follows. Section II reviews existing dynamic VMC works. Section III details the first contribution, a fully decentralized dynamic VMC schema. Section IV presents the second contribution, a migration-cost aware ACO-based dynamic VMC algorithm. Section V discusses the evaluation results. Finally, Section VI closes this paper with conclusions and future work.

II. Background

Several dynamic VMC systems and algorithms have been proposed during the last years. In this work we distinguish between four types of systems: *centralized, hierarchical, structured P2P, and unstructured P2P* which integrate different types of algorithms (e.g. mathematical programming, greedy).

In [4] a centralized system called Entropy is introduced. Entropy employs constraint programming to solve the VMC problem using a centralized architecture. Both design choices result in limited scalability and make it suffer from single point of failure (SPOF). A similar approach can be found in [5] which applies a greedy VMC algorithm on top of a centralized system architecture.

In our previous work [6] we have introduced Snooze, a scalable and autonomic VM management framework based on a hierarchical architecture. Snooze currently relies on a greedy VMC algorithm called Sercon proposed by Murtazaev et al [10]. VMC is only applied within groups of PMs thus allowing it to scale with increasing number of PMs and VMs. However, due to a lack of inter-group coordination currently the VMC solutions remain local thus limiting the packing efficiency of the system.

In [8] a system based on a structured P2P network of PMs is presented. Each PM attempts to apply VMC event-based upon underload/overload detection by first considering all the predecessor PMs and otherwise forwards the VMC request to the successor PM in the ring which repeats the procedure by considering itself and all its predecessor PMs. Besides the fact that the achievable packing efficiency remains unclear when used with periodic VMC, its scalability is limited by the cost of the ring maintenance and its worst-case complexity which requires a full ring traversal to consolidate the VMs.

In contrast to all these works our schema does not rely on a static system topology but rather employs a fully decentralized dynamic topology construction schema. Particularly, the system topology is periodically and randomly modified through the exchange of neighbourhood information among the PMs to form random PM neighbourhoods. VMC is applied periodically only within the scope of the neighbourhoods. This property does not only allow the system to scale with increasing number of PMs and VMs but also facilitates the VMC algorithm convergence towards almost the same global packing efficiency as of a centralized system. Finally, the schema is not limited to any particular VMC algorithm and supports heterogeneous PMs and VMs.

The closest work in terms of system architecture can be found in [11] where the authors introduce V-MAN, a fully decentralized VMC schema. Similarly to our work V-MAN leverages randomized system topology construction and applies VMC only within the scope of the neighbourhoods. However, it is limited to a simple VMC algorithm which considers at most two PMs at a time. Moreover, it makes its decisions solely based on the number of VMs thus ignoring the actual VM resource demands. As our results show both design decisions yield to a large number of migrations. Finally, V-MAN as introduced is restricted to homogeneous servers (physical/virtual) and has been validated by simulation only.

III. A Fully Decentralized Dynamic VM Consolidation Schema

This section presents our first contribution: *a fully decentralized dynamic VMC schema*. First, the system assumptions and notations are introduced. Afterwards, the design principles are discussed and the system topology construction mechanism is detailed. Finally, the VM consolidation process is presented.

A. System Assumptions

We assume a data center whose PMs are interconnected with a high-speed LAN connection such as Gigabit Ethernet or Infiniband. PMs hardware can be either homogeneous or heterogeneous. VMs can be live migrated between the PMs.

This work considers the *dynamic VMC problem*. Particularly, VMs are assumed to be running on the PMs and VMC is triggered periodically according to the system administrator specified interval to further optimize the VM placement.

VM resource demands are assumed to be static. In other words, VMC decisions are based on the *requested* VM capacity as specified during VM deployment. Taking into account the actual resource usage could further improve the data center utilization. However, it is out of the scope of this work.

B. Notations

We now define the notions used in this work. Let P denote the set of PMs and V_p the set of VMs hosted by a PM p.

PMs are represented by d-dimensional total capacity vectors $\mathbf{TC_p} := \{TC_{p,k}\}_{1 \leq k \leq d}$. In this work three dimensions are considered ($d = 3$). Each dimension k represents the PMs capacity of resource $R_k \in R$ with R being defined as $R := \{CPU, Memory, Network\}$.

VMs are represented by requested capacity vectors $\mathbf{RC_v} := \{RC_{v,k}\}_{1 \leq k \leq d}$ with each component specifying the requested VM capacity of resource R_k.

The total used PM capacity $\mathbf{l_p}$ is computed as the sum of all VM requested capacity vectors: $\mathbf{l_p} := \sum_{\forall v \in V_p} \mathbf{RC_v}$.

C. Design Principles

Due to the NP-hardness of the VMC problem it is not sufficient for a system to rely on a centralized server which executes the VMC algorithm. Indeed, to remain *scalable* and avoid SPOF with increasing numbers of servers (physical and virtual) a more distributed approach is desirable. Moreover, the system should compute VMC solutions which have a *high packing efficiency* thus able to release a large number of PMs.

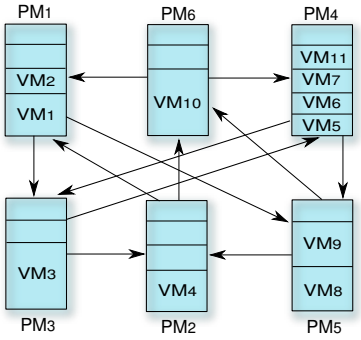
In order to achieve both scalability and high packing efficiency we have designed a fully decentralized VMC system based on an unstructured P2P network of PMs. For scalability our system is built on top of the Cyclon protocol [12]. Cyclon is an epidemic membership protocol which allows to periodically construct time-varying randomized P2P overlays in which each PM has only a partial system view, the so-called neighbourhood. This property allows the system to scale with increasing number of PMs as it does not rely on a central server. PMs can join and leave the network at any time without the need to notify a central server.

Our system exploits the randomized neighbourhoods in order to achieve a high packing efficiency by *periodically applying the VMC algorithms in the scope of the neighbourhoods*. As the neighbourhoods are modified periodically and randomly the entire system tends to converge towards a high packing efficiency by solely making local VMC decisions within neighbourhoods without the need of a central server.

D. System Topology Construction

The system topology construction mechanism is based on the Cyclon membership protocol. Cyclon has been designed for fast information dissemination while dealing with a high number of nodes that can join and leave the system. It is based on a periodic and random exchange of neighbourhood information among the peers, the so called *shuffling operation*. Each peer maintains a local list of neighbours, called *cache entries*. A cache entry contains the address (IP/port) and the *age* value of a neighbour. The role of the *age* field is to bound the time a neighbour is chosen for shuffling thus facilitating the early elimination of dead peers. The shuffling operation is repeated periodically according to a parameter $\lambda t > 0$. The resulting system topology can be viewed as a directed graph where vertices represent nodes and edges the relations. For example, $X \longrightarrow Y$ means *Y is a neighbour of X*. Note, that the relations are asymmetric (i.e. Y is a neighbour of X does not imply that X is a neighbour of Y). More details on the system topology construction can be found in [12].

Figure 1 depicts an example system topology with six PMs ($PM_1, \ldots, PM_6$) and eleven VMs ($VM_1, \ldots, VM_{11}$) that are distributed among the PMs. The neighbourhood size is two. For the sake of simplicity a single resource "number of cores" is considered in this example: physical (PCORES) and virtual (VCORES). PMs are homogeneous and have five PCORES. VMs can request one, two or three VCORES. The total capacity of PMs in the system is 30 PCORES. 19 PCORES are currently utilized, which corresponds to an utilization of approximately 63%.



PM	Neighbours
1	{3,5}
2	{1,6}
3	{2,4}
4	{3,5}
5	{2,6}
6	{1,4}

Fig. 1: Example system topology

E. VM Consolidation Process

Each PM periodically triggers a VM consolidation process within its neighbourhood in order to optimize the VM placement. The process is composed of the following six steps:

1) First, PM p which initiates the consolidation checks whether it is not already under consolidation. If not it attempts to acquire a lock for each member (including itself) of its neighbourhood. Otherwise, the consolidation is aborted. Locks are associated with all PMs in order to avoid concurrent access to PM resources in case of multiple ongoing consolidations. Acquiring a lock is a *non-blocking operation*. If it does not succeed, the member will not participate in the consolidation process.
2) For each successful lock acquisition, PM p requests from the corresponding node its total capacity, currently packed VMs and their requested capacity vectors.
3) The VMC algorithm is started once all the resource information is received from the locked members. It

outputs a migration plan (MP) which corresponds to the ordered set of the new VM-PM assignments. Any centralized VMC algorithm can be used in this operation.
4) An actuation module on the PM enforces the MP by sending migration operations to the PMs hypervisors.
5) After the actuation all locks are released.
6) Each PM which does not accommodate VMs anymore power-cycles itself in order to conserve energy.

Figure 2 illustrates the decentralized consolidation process using the topology introduced in Figure 1. Starting from the initial state as shown in Figure 1, PM_4 initiates the first consolidation with its neighbours PM_3 and PM_5. The result of this consolidation is shown in (1). VM_{11} has been migrated from PM_4 to PM_5 and VM_6 & VM_7 have been migrated from PM_4 to PM_3. PM_5 and PM_3 resources are now better utilized than in the previous configuration. In (2), PM_2 triggers a consolidation with its neighbours PM_1 and PM_6. VM_4 has been migrated from PM_2 to PM_1. PM_1 is now fully utilized and PM_2 has become idle. Finally, in (3) PM_6 starts another consolidation with PM_1 and PM_4. VM_5 has been moved from PM_4 to PM_6. The node PM_4 has now become idle and PM_6 better utilized. The final system state with two released PMs (PM_2 and PM_4) is shown in (4). It results in a new, almost global optimal data center utilization of approximately 95%.

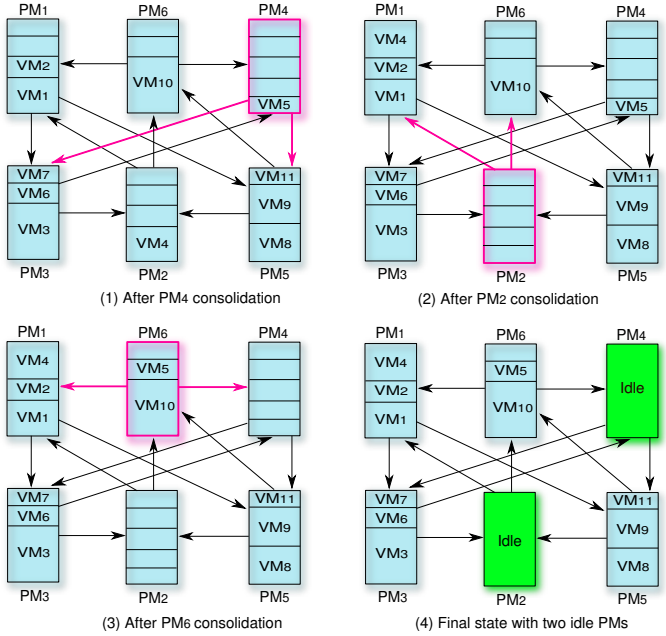


Fig. 2: Example VM consolidation

IV. A Migration-cost Aware ACO-based Dynamic VM Consolidation Algorithm

This section presents our second contribution: *a novel migration-cost aware ACO-based dynamic VMC algorithm*, one possible algorithm which can be integrated in the previously introduced fully decentralized VMC schema. First, a brief description of ACO is given and the algorithm design principles are outlined. Afterwards, the formal problem definition is introduced and the algorithm details are presented.

A. Ant Colony Optimization

Ant Colony Optimization (ACO) [13] is a probabilistic meta-heuristic for finding near optimal solutions to complex combinatorial optimization problems (e.g. VMC). ACO design is inspired from the natural food discovery behaviour of real ants. Particularly, while traversing their environment ants deposit a chemical substance called *pheromone*. Other ants can smell the concentration of this substance on a certain path and tend to favour paths with a higher concentration. To favour the exploration of alternative paths pheromone evaporates after a certain amount of time. Controlled experiments have shown that after some time the entire colony tends to converge towards the shortest path to the food source [14].

B. Design Principles

In our previous work [9] we have applied ACO to solve the static VMC problem. ACO is especially attractive for VMC due to its polynomial worst-case complexity and ease of parallelization (i.e. ants work independently).

The key idea of the algorithm is to let multiple artificial ants construct VMC solutions concurrently in parallel within multiple cycles. As the *VMC model does not incorporate the notion of a path*, ants deposit pheromone on each VM-PM pair within a *pheromone matrix*.

To choose a particular VM as the next one to pack in a PM a probabilistic decision rule is used. This rule is based on the current pheromone concentration information on the VM-PM pair and a heuristic information which guides the ants towards choosing VMs leading to better overall PM utilization. Hence, the higher the amount of pheromone and heuristic information associated with an VM-PM pair is, the higher the probability that it will be chosen.

At the end of each cycle, all solutions are compared and the one requiring the least number of PMs is saved as the new global best solution. Afterwards, the pheromone matrix is updated to simulate pheromone evaporation and reinforce VM-PM pairs which belong to the so-far best solution.

In the following sections we describe the modifications (marked bold) done to the original algorithm for considering the dynamic VMC problem. They involve the objective function, heuristic information, the pheromone evaporation rule and finally the algorithm pseudo-code.

C. Formal Problem Definition

The objective function (OF) we attempt to maximize is defined by Eq. 1. It takes as input the set of PMs and the migration plan (MP). MP denotes the ordered set of new VM to PM assignments.

$$\begin{aligned} \textbf{max } \mathbf{f}(\mathbf{P}, \mathbf{MP}) := (nReleasedPMs)^e \\ \times \mathrm{Var}((|\mathbf{l_p}|_1)_{p \in P})^g \times (\frac{1}{|MP|})^m \end{aligned} \tag{1}$$

The OF is designed to favour the number of released PMs, the variance [15] of the scalar valued PM used capacity vectors $\mathbf{l_p}$ and smaller MPs. In other words, the higher the number of released PMs and the variance between the PMs used capacity

vectors, the better it is. Indeed, one of our objectives is to release as many PMs as possible. Releasing PMs also helps to increase the variance which is an important indicator for increased resource utilization. In this work the L1 norm [16] is used to compute the scalar values. The second objective is to minimize the number of migrations. Consequently, we favour MPs with the least number of migrations. This is reflected by defining the OF to be inverse proportional to the MP size. MPs with high number of migrations (i.e. VM-PM pairs) will lower its value, while smaller MPs will increase it.

Three parameters, $e, g, m > 0$, are used to either give more weight to the number of released PMs, the PM load variance or the MP size.

D. Probabilistic Decision Rule and Heuristic Information

The probability $p_{v,p}$ for an ant to choose a VM v to be migrated to PM p is defined in Eq. 2.

$$p_{v,p} := \frac{[\tau_{v,p}]^{\alpha} \times [\eta_{\mathbf{v},\mathbf{p}}]^{\beta}}{\sum_{v \in V_p} [\tau_{v,p}]^{\alpha} \times [\eta_{\mathbf{v},\mathbf{p}}]^{\beta}}, \quad \forall v \in V, \forall p \in P \tag{2}$$

where $\tau_{v,p}$ represents the amount of pheromone associated with a particular VM-PM pair. $\eta_{v,p}$ denotes the heuristic information. Its definition is shown in Eq. 3.

$$\eta_{\mathbf{v},\mathbf{p}}, := \begin{cases} \frac{\kappa_{v,p}}{|MP|} & \text{if } \mathbf{l_p} + \mathbf{RC_v} \leq \mathbf{TC_p} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The key idea is to emphasize VM to PM migrations which yield in: high PM used capacity and are part of a small MP. Consequently, $\eta_{v,p}$ is defined as the ratio between $\kappa_{v,p}$ and the MP size in case the VM fits into the PM, and 0 otherwise. We use the constraint $\mathbf{l_p} + \mathbf{RC_v} \leq \mathbf{TC_p}$ to prevent new VMs from exceeding the total PM capacity.

To reward VMs which fill the PMs better $\kappa_{v,p}$ is defined as the inverse of the scalar valued difference between the static PM capacity and the utilization of the PM after placing VM v (see Eq. 4). Consequently, VMs which yield to better PM used capacity result in higher $\kappa_{v,p}$ value.

$$\kappa_{v,p} := \frac{1}{|\mathbf{TC_p} - (\mathbf{l_p} + \mathbf{RC_v})|_1} \tag{4}$$

Finally, two parameters, $\alpha, \beta \geq 0$ are used to either emphasize the pheromone or heuristic information.

E. Pheromone Trail Update

After all ants have computed a MP, the pheromone trail update rule is used to reward VM-PM pairs which belong to the smallest MP (MP_{gBest}) as well as to simulate pheromone evaporation on the remaining VM-PM pairs. The pheromone trail update rule $\tau_{v,p}$ is defined in Eq. 5.

$$\tau_{v,p} := (1 - \rho) \times \tau_{v,p} + \mathbf{\Delta^{best}_{\tau_{v,p}}}, \quad \forall (v,p) \in V \times P \tag{5}$$

where ρ, $0 \leq \rho \leq 1$ is used to control the evaporation rate. Consequently, higher values for ρ yield to faster pheromone evaporation. In order to reward VM-PM pairs which belong to the best MP, $\Delta^{best}_{\tau_{v,p}}$ is defined as the cycle-best VM-PM pheromone amount. Particularly, VM-PM pairs which belong to the best MP receive an increasing pheromone amount and thus become more attractive during subsequent cycles. Other pairs, which are not part of the best MP continue loosing pheromone and thus become less attractive.

The goal of the algorithm is to release as many PMs as possible using the least number of migrations. Consequently, it attempts to maximize the OF f. Therefore, $\Delta^{best}_{\tau_{v,p}}$ is defined to give $f(P, MP_{gBest})$ pheromone amount to VM-PM pairs (v, p) which belong to MP_{gBest}, and 0 otherwise.

$$\mathbf{\Delta^{best}_{\tau_{v,p}}} := \begin{cases} \mathbf{f(P, MP_{gBest})} & \text{if } (v,p) \in MP_{gBest} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Finally, to bound the pheromone amount on the VM-PM pairs, $\tau_{v,p}$ is restricted to the range $[\tau_{min}, \tau_{max}]$.

F. Algorithm Definition

The pseudo-code is shown in Algorithm 1. It takes as input the set of PMs P including their associated VMs and a set of parameters (e.g. τ_{max}, α, β, nCycles, nAnts) (line 1). The algorithm then sets the pheromone value on all the VM-PM pairs to τ_{max} and iterates over a number of nCycles (lines 5 to 35). In each cycle multiple (nAnts) ants compute MPs concurrently in parallel (lines 7 to 21). The MPs accommodate at most $|VM|$ migrations (line 10). Particularly, first the ants compute a probability $p_{v,p}$ for migrating a VM v to PM p for all VMs and PMs (line 11). Based on the computed probability they choose a VM-PM pair (v, p) stochastically and add it to the MP (lines 12 to 13). The source and destination PM capacity is then updated by removing the selected VM from the source and adding it to the destination PM (line 14). Afterwards, a score is computed by applying the OF (see Eq. 1) on the set of PMs P and the MP (line 15). Finally, if the newly computed score is greater than the local best score the local best score is updated and the VM-PM pair (v, p) is added to the local MP (lines 16 to 19). Note, that a VM is allowed to appear multiple times in the MP as long as it yields to a better score.

After all ants have finished computing the MPs, they are compared according to the OF f. The cycle best MP is selected and saved as MP_{cBest} (line 22). If the cycle best MP is also the global best one, it becomes the new global best one (lines 23 to 25). Finally, the pheromone values on all VM-PM pairs are updated by applying the pheromone trail update rule (see Eq. 5) and enforcing the τ_{min}, τ_{max} bounds (lines 26 to 34).

The algorithm terminates after nCycles and returns the global best migration plan MP_{gBest} (line 36).

V. Experimental Results

This section presents the evaluation results. First, the prototype implementation principles are introduced. Afterwards, the system setup is detailed and the results are discussed.

A. Prototype Implementation

To validate our work we have implemented a distributed Python-based VMC-enabled Cyclon P2P system emulator. Each PM is *emulated* by a daemon which listens for TCP

Algorithm 1 Migration-cost Aware ACO-based VMC

Input: Set of PMs P with their associated VMs, Set of parameters
Output: Global best migration plan MP_{gBest}

$MP_{gBest} := \emptyset$
Set pheromone value on all VM-PM pairs to τ_{max}
for all $q \in \{0 \dots nCycles - 1\}$ **do**
  for all $a \in \{0 \dots nAnts - 1\}$ **do**
    $Score_{lBest} := 0$
    MP_{tmp}, $MP_a := \emptyset$
    while $|MP_{tmp}| < |VMs|$ **do**
      Compute $p_{v,p}$, $\forall v \in V, \forall p \in P$
      Choose (v, p) randomly according to the probability $p_{v,p}$
      Add (v, p) to the migration plan MP_{tmp}
      Update PMs used capacities
      $Score_{tmp} := f(P, MP_{tmp})$
      if $Score_{tmp} > Score_{lBest}$ **then**
        $Score_{lBest} := Score_{tmp}$
        $MP_a := MP_a \cup \{(v, p)\}$
      end if
    end while
  end for
  Compare ants migration plans and choose the best one according to the objective function $f(P, MP_a) \rightarrow$ Save cycle best migration plan as MP_{cBest}
  if $f(P, MP_{cBest}) > f(P, MP_{gBest})$ **then**
    $MP_{gBest} := MP_{cBest}$
  end if
  for all $(v, p) \in V \times P$ **do**
    $\tau_{v,p} := (1 - \rho) \times \tau_{v,p} + \triangle\tau_{v,p}^{best}$
    if $\tau_{v,p} > \tau_{max}$ **then**
      $\tau_{v,p} := \tau_{max}$
    end if
    if $\tau_{v,p} < \tau_{min}$ **then**
      $\tau_{v,p} := \tau_{min}$
    end if
  end for
end for
return Global best migration plan MP_{gBest}

connections. One node serves as the introducer to bootstrap the system. To prevent concurrent access to PMs the prototype implementation integrates a distributed locking mechanism. PMs shutdown themselves when they do not host any VMs. VMs are represented by their requested capacity vectors. Each PM writes events (e.g. migration, consolidation, shutdown) in a local SQLite database during the experiment execution. Once the experiment is finished all databases are collected and merged into a single one for post-analysis. Emulator modules such as the introducer mechanism, consolidation algorithms, scheduler for shuffling and consolidation are defined in a configuration file for the ease of replacement.

The current implementation integrates four VMC algorithms: the FFD, Sercon, V-MAN state-of-the-art algorithms and the introduced migration-cost aware ACO-based algorithm. FFD is a static VMC algorithm which is often applied in the context of dynamic VMC. Consequently, it serves as the baseline for comparison in our work. In contrast, Sercon, V-MAN and our algorithm are dynamic VMC algorithms which were specifically designed to reduce the number of migrations.

B. System Setup

We have deployed the emulator on 42 servers of the Grid'5000 testbed in France. All servers are equipped with two CPUs each having 12 cores (in total 1008 cores). This allowed us to emulate one PM per core. In other words, throughout all the experiments each server hosts 24 emulator instances (one per core) which represent the emulated PMs.

The emulator considers three types of resources: CPU, memory, and network. It supports six kinds of VM instances: nano, micro, small, medium, large and xlarge, which are represented by their corresponding requested capacity vectors: (0.2, 0.5, 0.1), (1, 1, 1), (2, 1, 1), (4, 2, 2), (8, 4, 4) and (16, 8, 4) respectively.

PMs have a total capacity of (48, 26, 20). They host 6 VMs, one of each type at the beginning of the experiment. Consequently, in total 6048 VMs are emulated. The experiment runs for six minutes. Consolidation is triggered by the PMs concurrently and independently every 30 seconds. The neighbourhood size is set to 16 PMs and the shuffling operation is triggered every 10 seconds by the PMs. Table II provides a summary of the introduced system parameters and their corresponding values.

TABLE II: System parameters

Parameter	Value
Number of PMs and VMs	1008 (resp. 6048)
Experiment duration	360s
Consolidation interval	30s
Shuffling interval	10s
Neighbourhood size	16 PMs
Considered resources	CPU, memory and network
PM total capacity vector	(48, 26, 20)
VM requested capacity vectors	(0.2, 0.5, 0.1), (1, 1, 1), (2, 1, 1), (4, 2, 2), (8, 4, 4), (16, 8, 4)

We run the emulator once for each of the evaluated algorithms: FFD, Sercon, V-MAN and the proposed ACO-based algorithm. The ACO parameters are shown in Table III. They were derived empirically through numerous experiments.

TABLE III: ACO parameters

α, β	ρ	$\tau_{\min}$, $\tau_{\max}$	**e**, **g**, **m**	**nCycles**	**nAnts**
0.1, 0.9	0.1	0.2, 1	5, 3, 1	2	2

The evaluation is focused on: (1) analysis of the number of active PMs (packing efficiency) and migrations; (2) scalability of the system; and (3) comparison of the packing efficiency with the centralized topology for all the VMC algorithms.

C. Number of Active PMs and Migrations

We first analyze the number of active PMs. The results of this evaluation are shown in Figure 3. As it can be observed the consolidation phase starts at the 30th second. FFD performs the worst as it only manages to release 246 nodes. V-MAN achieves the best result with 323 released PMs which is closely followed by the ACO-based algorithm with 322 released PMs. Note, that Sercon performs worse than V-MAN and ACO.

Figure 4 depicts the number of migrations with the progress of the experiment. As it can be observed the number of migrations quickly converges towards zero with Sercon, V-MAN and the ACO algorithm thus demonstrating the good reactivity of our schema. Note, that the ACO algorithm requires more migrations than Sercon. Indeed, it trades the number of migrations for the amount of released PMs (see Figure 3). Finally, V-MAN performs the worst among the three dynamic

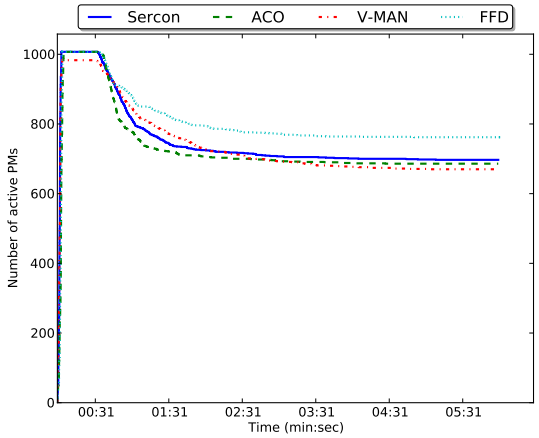


Fig. 3: Number of active PMs

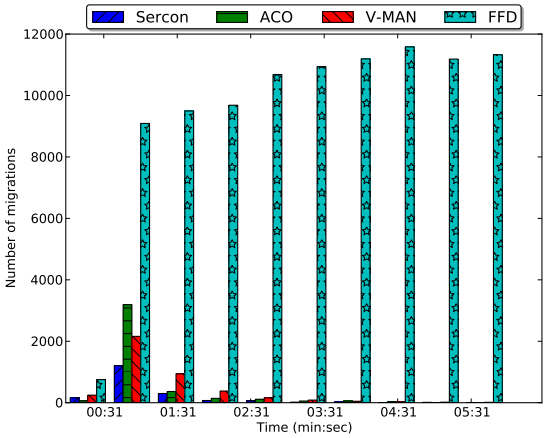


Fig. 4: Number of migrations

VMC algorithms. FFD yields in a tremendous amount of migrations (in total 96494). We explain this with the fact the algorithm is *not designed to take into account the current VM-PM assignment*. Particularly, due to its static nature it assumes that the PMs do not host any VMs prior computing the new VM-PM assignment. This results in a permanent movement of most of the VMs in each VMC iteration.

D. Scalability

To evaluate the scalability of our system we have varied the number of PMs from 120 to 1008 and analyzed the obtained packing efficiency. The results are summarized in Table IV. As it can be observed, except the outlier with V-MAN at 504 PMs, the packing efficiency does not change significantly with increasing numbers of PMs and VMs, thus demonstrating the good scalability of our schema.

TABLE IV: Scalability

Algorithm	PMs	VMs	Released PMs	Migrations per VM	Packing Efficiency (%)
FFD	120	720	29	26	24.1
	240	1440	58	26	24.1
	504	3024	124	27	24.6
	1008	6048	246	26	24.4
ACO	120	720	36	5	30.0
	240	1440	77	7	32.0
	504	3024	161	8	31.9
	1008	6048	322	9	31.9
V-MAN	120	720	39	3	32.5
	240	1440	79	5	32.9
	504	3024	122	4	24.2
	1008	6048	323	4	32.0
Sercon	120	720	37	1	30.8
	240	1440	74	1	30.8
	504	3024	155	1	30.7
	1008	6048	311	1	30.8

Another important metric to evaluate is the *maximum number of migrations per VM* during the whole duration of the experiment. In other words, due to the fully decentralized nature of the system and the random neighbourhood construction, VMs could traverse multiple PMs during subsequent consolidation rounds. As our results show, the maximum number of migrations per VM highly depends on the current VM-PM assignment and the VMC algorithm, less on the number of PMs and VMs. Particularly, in the current setup, Sercon requires at most one migration per VM. On the other hand, V-MAN results in at most 5 and ACO needs at most 9 migrations. Finally, FFD as it does not consider the current VM-PM assignment yields to the largest number of migrations.

E. Comparison with a Centralized System Topology

Table V depicts the results from the comparison of the number of migrations and packing efficiency of our approach with the centralized topology for 1008 PMs and 6048 VMs. To simulate a centralized topology we have run the VMC algorithms (FFD, Sercon, ACO) on a single PM. Note that, V-MAN is a decentralized algorithm thus its evaluation is not part of the centralized topology evaluation.

TABLE V: Centralized vs. Unstructured P2P Topology

Topology	Algorithm	Migrations	Released PMs	Packing Efficiency (%)
Centralized	FFD	6040	249	24.7
	Sercon	1920	320	31.7
	ACO	-	-	-
P2P	FFD	96494	246	24.4
	V-MAN	4189	323	32.0
	ACO	4015	322	31.9
	Sercon	1872	311	30.8

As it can be observed the ACO-based VMC algorithm is unable to compute a solution in a reasonable amount of time when used in the centralized topology for this kind of scale. Sercon on the other hand outperforms FFD in both the number of migrations (1920 vs. 6040) and released PMs (320 vs. 249). This is not further surprising as in contrast to FFD, Sercon is designed to minimize the number of migrations.

More interestingly, our fully decentralized VMC schema achieves almost equivalent packing efficiency for the evaluated algorithms when compared to the centralized topology. When considering the number of migrations, FFD achieves the worst result with 96494 migrations. We explain this with the fact that the algorithm by nature does not take into account the current VM-PM assignments. Consequently, its solutions result in a permanent reassignment of VMs within neighbourhoods during subsequent consolidation rounds.

Our ACO-based algorithm outperforms FFD as well as Sercon in the number of released PMs and performs equal with V-MAN. However, the gains in the number of released PMs come at the cost of an increased number of migrations. For example, when compared to Sercon twice as many migrations are required. On the other hand, when considering V-MAN more than 150 migrations are saved by the ACO algorithm. This demonstrates that the ACO algorithm can serve as a competitive alternative to the other evaluated algorithms in the fully decentralized VMC schema.

VI. Conclusions and Future Work

This paper has made two novel contributions. The first contribution has introduced a fully decentralized dynamic VMC schema based on an unstructured P2P network of PMs. In contrast to existing works which mostly rely on static system topologies our system employs a dynamic topology which is built by periodically and randomly exchanging neighbourhood information among the PMs. VMC is periodically applied only within the scope of the neighbourhoods thus allowing the system to scale with increasing number of PMs and VMs as no global system knowledge is required. Moreover, the randomized neighbourhood construction property facilitates the VMC convergence towards a global packing efficiency very similar to a centralized system by *leveraging existing centralized VMC algorithms*. The second contribution has introduced a migration-cost aware ACO-based dynamic VMC algorithm. In contrast to our previously proposed ACO-based algorithm which solely focuses on minimizing the number of PMs, the novel algorithm also attempts to minimize the number of migrations required to move from the current to the newly computed VM placement.

We have implemented a distributed Python-based VMC-enabled Cyclon P2P system emulator and used it to evaluate three state-of-the-art VMC algorithms, namely FFD, Sercon and V-MAN along with our migration-cost aware ACO-based algorithm. The evaluation was conducted on a 42 node cluster of the Grid'5000 testbed which allowed to emulate up to 1008 PMs and 6048 VMs. The results show that the proposed fully decentralized VMC schema achieves a global packing efficiency very close to a centralized topology for all the evaluated algorithms. Moreover, the system remains scalable with increasing numbers of PMs and VMs. Finally, the proposed migration-cost aware ACO-based algorithm outperforms FFD and Sercon in the number of released PMs and requires less migrations than FFD and V-MAN when used in our fully decentralized VMC schema (see Table I).

In the future we plan to study the impact of neighbourhood size and ACO parameters choice on the system performance. We also plan to integrate both contributions in our recently open-sourced holistic energy-efficient VM management framework called Snooze [6]. This will allow taking into account the dynamic VM resource demands as well as to handle underload situations thus allowing to further improve the data center utilization. In this context we will investigate the complementaries between VM resource demands in order to support more accurate VM to PM assignment decisions (e.g. co-locate CPU and data intensive VMs). This will require the extensions of existing VMC algorithms to consider VM placement constraints such as co-location and anti-colocation.

Finally, we will work on improving the neighbourhood construction schema to build neighbourhoods taking into account the physical distance between the PMs. Last but not least, fault tolerance properties of the VMC schema will be evaluated.

VII. Acknowledgements

We would like to express our gratitude to Louis Rilling and Nikos Parlavantzas for their detailed feedbacks. This research is funded by the French *Agence Nationale de la Recherche (ANR)* project EcoGrappe under the contract number ANR-08-SEGI-008-02. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] G. International, "How Clean Is Your Cloud?" 2012. [Online]. Available: http://www.greenpeace.org/international/Global/international/publications/climate/2012/iCoal/HowCleanisYourCloud.pdf

[2] Rackspace, "Hosting reports third quarter," 2011. [Online]. Available: http://ir.rackspace.com/phoenix.zhtml?c=221673&p=irol-newsArticle&ID=1627224&highlight=

[3] VMware, "Distributed Resource Scheduler (DRS)," 2012. [Online]. Available: http://www.vmware.com/products/drs/

[4] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '09, 2009.

[5] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *ACM/IFIP/USENIX 9th International Conference on Middleware*, 2008.

[6] E. Feller, L. Rilling, and C. Morin, "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds," in *12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2012)*, Ottawa, Canada, May 2012.

[7] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-Efficient Management of Virtual Machines in Eucalyptus," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, 2011.

[8] F. Quesnel, A. Lèbre, and M. Suedholt, "Cooperative and reactive scheduling in large-scale virtualized platforms with DVMS," *Concurrency and Computation: Practice and Experience*, 2012.

[9] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, ser. GRID '11, 2011.

[10] A. Murtazaev and S. Oh, "Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing," *IETE Technical Review*, vol. 28 (3), 2011.

[11] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in Clouds through gossiping," in *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, ser. WOWMOM '11, 2011.

[12] S. Voulgaris, D. Gavidia, and M. V. Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, 2005.

[13] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artif. Life*, vol. 5, pp. 137–172, April 1999.

[14] Deneubourg, S. Aron, S. Goss, and J. M. Pasteels, "The self-organizing exploratory pattern of the argentine ant," *Journal of Insect Behavior*, vol. 3, no. 2, pp. 159–168, March 1990.

[15] L. A. Goodman, "The Variance of the Product of K Random Variables," *Journal of the American Statistical Association*, vol. 57, no. 297, 1962.

[16] E. W. Weisstein, "L1-norm," 2012. [Online]. Available: http://mathworld.wolfram.com/L1-Norm.html