Energy-Aware Distributed Ant Colony Based Virtual Machine Consolidation in IaaS Clouds

Armel Esnault

ISTIC
Université de Rennes 1
Campus de Beaulieu
35042 RENNES CEDEX

INRIA
Équipe MYRIADS
Campus de Beaulieu
35042 RENNES CEDEX

Energy-Aware Distributed Ant Colony Based Virtual Machine Consolidation in IaaS Clouds

Armel Esnault
supervised by
Eugen Feller and Christine Morin

June 5, 2012

Contents

1 Introduction

Cloud computing is the abstraction of resources in order to offer them as a service for the client. This economy model is based on the pay-as-you-go model in which clients are charged only for what they consume. IaaS is a category of cloud computing where providers rent resources such as computing power, storage capacity or network bandwidth in the form of Virtual Machines (VMs). The goal is to discharge clients of infrastructure management while giving them full control of their operating systems. The challenge for the provider is then to handle all its resources in an energy efficient way while meeting the Service-Level Agreement (SLA) which is a contract between the customers and providers on the Quality of Service (QoS). In such large scale system knowing that they can hosts several thousands of nodes, energy consumption has become a growing problem. Due to economical and environmental reasons cloud providers now must optimize the energy consumption of their data center. One way to optimize energy consumption is to perform VM consolidation which aims at reducing the number of Physical Nodes (PNs) running in the data-center by optimizing the placement of VMs on PNs such that VMs are packed on the fewer PNs as possible. Unused nodes can then be shutdown or put in an energy saving mode such as suspend to ram or suspend to disk. Hence these unused machines consume less energy and generate less heat. Furthermore, if machines in the data-center generate less heat they require less cooling, therefore energy consumption of conditioning system is decreased. This reduces the overall power consumption of the cloud.

Current existing solutions are either centralized which limits scalability either hierarchical which lead to local optimizations or either use ring strucured P2P solution which is hard to maintain and need in worst cases full traversal of the ring to find nodes. We propose a scalable fully distributed energy aware epidemic protocol to optimize the placement of VMs while taking into account the number of migrations.

The remainder of this paper is organized as follows: Section 2 provides a formal problem definition. Section 3 presents the related work. Our approach is presented in Section 4. Experimentals results are detailed in Section 5. Finally, Section 6 depicts conclusions and future works.

2 Problem Statement

The consolidation can be describe in four steps: monitoring,estimation,reconfiguration and actuation. Monitoring is the collection of resources usage in the system for VMs. According to the data collected during the monitoring phase the estimation analysis find estimates needs of VMs and detects hotspots. A hotspot is an host for which at least one resource (e.g. CPU, RAM, Bandwidth) exceeds a threshold defined by SLA for a sustained period. The goal is to move away overloaded VMs according to the SLA by using live migration. Live migration [KKL+07] is a mechanism that allows to move a VM without rebooting the operating system inside the VM. In [CFH+05] authors shown that the service downtime due to migration is low (i.e. 60ms), thus allowing to move VMs with low performance impact. The reconfiguration is the step that given a starting state (which is the mapping of VMs on hosts) to find a new optimized state and a migration plan which correspond to the migration required to move from the starting to the new one. It can be decompose in two phases: the optimization and the planning. Once optimization and planning have been performed a module called Actuator will enforce the plan in order to reach the new state. In this paper we focus on the optimization and planning part of the consolidation.

Virtual Machine Consolidation Problem (VMCP) is given a set of VMs and a set of physical nodes to find the best mapping (which lead bo better utlization of the data center) of association (VM in PN) according to constraints on VMs and resources provided by PNs. Constraints on VMs are the resources needed to perform properly, such as the number of core, the amount of memory or the network I/O. Optimization for VMs consolidation can be reduced to the Multi Dimensional Bin Packing Problem (MDBPP) [Sha04]. The MDBPP is a problem of subset selection in which a set objects that have different volumes must but packed into the minimum amount of bins of different capacity without exceeding the total capacity of each bin. MDBPP is a NP-hard problem and thus optimal algorithms are not scalable in terms of number of nodes (due to exponential cost in time and space of finding an optimal solution) which is compulsory in large scale system such as cloud.

VMCP differs from the classic MDBPP in several points: in BPP the initial state assumes that all bins are empty, in VMCP the initial state contains already a configuration that must be optimized. In VMCP number of object movement (i.e. migration) must be considered and minimized in order to improve performance. Indeed too many migration can lead to network congestion and thus performance degradation. In VMCP heterogeneity of VMs and PNs are considered as well as the number of dimensions on items on bins (e.g. CPU power, RAM size, network bandwidth)

More formally let B the set (of size n) of bins that represent hosts, I the set (of size m) of items that represent VMs and R the set (of size d) of resources. For each bin B_v is associated its capacity vector $\vec{C}_v$. For each item $i \in I$ its capacity time-varying d-dimensional resource demand vectors $\vec{r}_i$. y_v represent the bin allocation variable which is equals to 1 if the bin v is

chosen, and 0 otherwise. The goal of the consolidation process is to minimize the objective function $f(y) = \sum_{v=0}^{n-1} y_v$ which represent the number of bins used while ensuring that the capacity of each bin is not exceeded and guaranteeing that each item is assigned to exactly one bin.

Figure 2 shows an example of VM consolidation process. For the sake of simplicity only RAM utilization is considered in this example. During SLA negotiation VMs reserve an amount of RAM. Then VMs are placed on nodes which have enough space to host them. The monitoring phase will find current amount of RAM used by all VMs and free space on each node. The reconfiguration step optimize the placement of VMs. In this example VM5 can be migrated on node two and VM6 on node one, this allow to reduce the number of node utilized, put unused nodes in a power saving state and thus save energy.

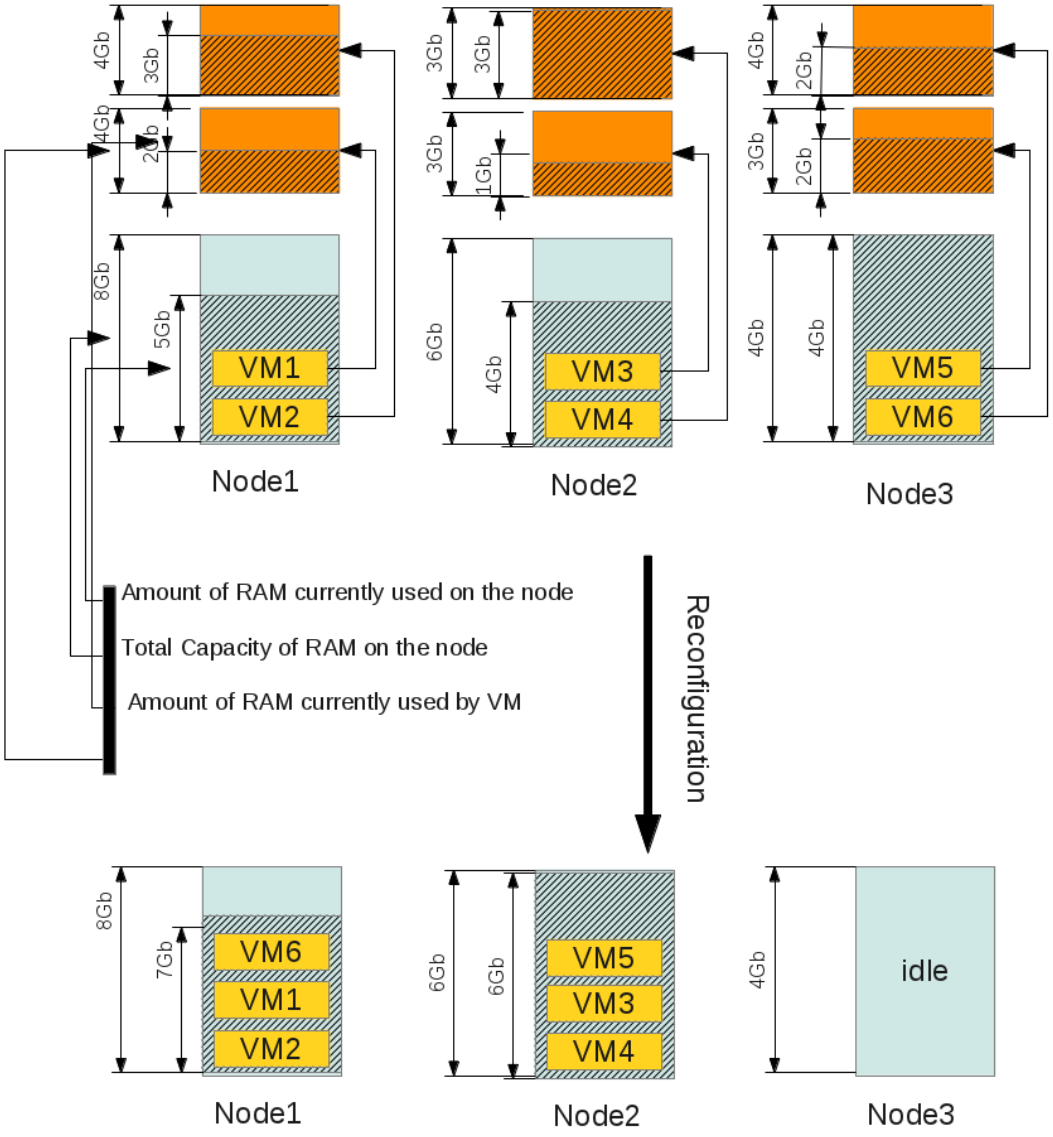


Figure 1: VM Consolidation

3 Related Work

In this section related work in VM consolidation and VMs management framework is presented . First, algorithms for solving the optimization problem and then framework for VMs management.

3.1 VM consolidation algorithms

3.1.1 Greedy

Several greedy algorithm have been proposed to solve the MDBPP such as First Fit Decreasing (FFD) [Yue91] which sort items by their decreasing order then for each item try to pack it in the first bin that can host it. This kind of algorithms does not take into account the current VM placement during the migration plan computation.

Sercon [MO11] is a greedy algorithm that minimize the number of severs used and the number of migrations. Sercon first sorts bins by their decreasing order according of the load of the bins then take the most loaded bin and sort the current items already placed in this bin and try allocate them in others bins by trying the items with the highest demand first. If not all items can be move away the operation is canceled. In this manner only migrations which lead to free bins will be performed.

3.1.2 Metaheuristic

Ant Colony Optimization (ACO) [Dor92] is a method for finding near optimal solutions using a probabilistic technique. It can be used for problems which belong to the NP class, VMCP. ACO algorithms were discovered by observing nature and especially ant colonies. Ants communicate indirectly using their environment. They deposit pheromone which is a chemical substance that control the behavior of ants when they encounter it. This method of communication is called stigmergy. Ants use probabilistic decision for their travel while searching food. They are more likely to choose paths with higher quantity of pheromone. When ants find food, they deposit pheromone on the return in order to induce others ants to follow the path until they reach the food source. Pheromone evaporation correspond to the natural evaporation of chemical substance deposited by ants. It is used to reduce the amount of pheromone over time for keeping only trails that are regularly used. This system aim to promote trails which lead to better solutions. In [FRM11a] ACO has been used to solve VMCP and results show that ACO outperform FFD algorithms.

3.1.3 Mathematical programming

Entropy [HLM+09] utilizes constraint programming (CP) in order to find a better solution than heuristic methods. They use a constraints solver [JRL08] enhanced with a FFD heuristic

to lower the number of nodes used in the cluster. They consider only memory as resource. Entropy use a second algorithm to find the plan of migration. Using CP for finding optimal solution to the BPP leads to exponential complexity and hence this algorithm is not scalable.

3.2 Distributed VM Management Architectures

3.2.1 Centralized

Figure 5.3 show a centralized topology where a master node (M) control slave nodes (S_i) which host VMs. In this architecture VMCP is solved on the master node.

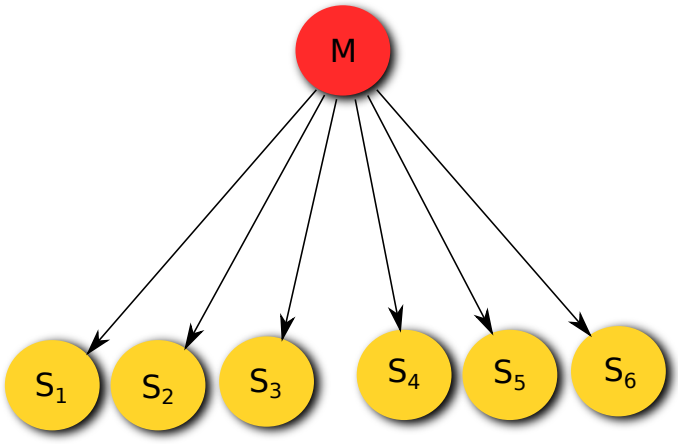


Figure 2: Centralized architecture

Sandpiper [WSVY07] is a system that automates the task of monitoring and detecting hotspots. It use two approaches for monitoring: *black-box* and *grey-box*. The first *black-box* monitors only information that come from the Xen hypervisor such as CPU, disk or network activity without interfering with the VM. *Gray-box* uses a daemon installed on the guest system that look in /proc/ and in others daemon log for gathering memory utilization, service time, request drop rate and incoming request rate. A hotspot detection algorithm tracks SLA violation by analyzing from monitoring information SLA exceeding during a fixed amount of time. However they use a centralized greedy algorithm for optimization of VMs and hence is not scalable.

pMapper [ANV11] is a framework for power-aware application placement in heterogeneous clusters. pMapper is architectured around different components: the performance manager is aware of QoS and SLA and decides actions such as VM re-sizing and idling, the power manager controls power management at hardware level while the migration manager controls VM consolidation using VM migration. For optimization of VM placement they use an algorithm: min Power Parity (mPP). mPP is based on the FFD algorithm for solving the

bin-packing problem. This algorithm is centralized and hence expose a limited scalability and Single Point of Failure (SPOF).

3.2.2 Hierarchical

To address scalability problems of centralized architecture a hierarchical topology has been proposed, figure 3.2.2 shows the hierarchical approaches of Snooze [FRM11b]. Snooze is a scalable, ernegy aware and autonomic VMs management framework for private clouds. A Group Leader (GL) controls several Group Manager (GMs). Each Group Manager has only a partial view of the system. It manages a subset of the Local Controllers (LCs) that hosts VMs.

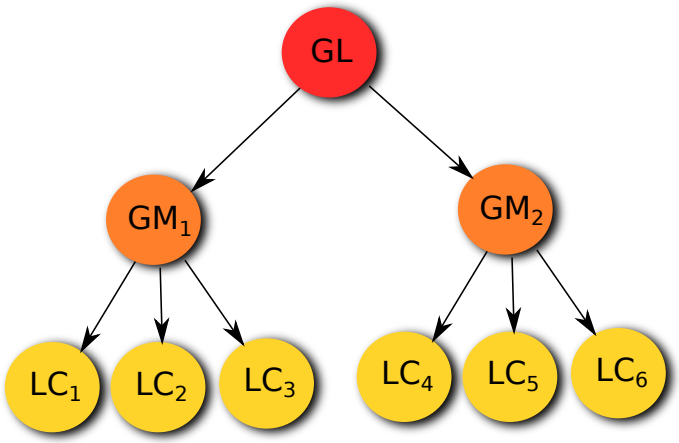


Figure 3: Snooze architecture

Snooze includes a consolidation manager that can integrate any consolidation algorithm but currently uses a centralized ACO-based algorithm. Snooze use self-organization with leader election to repair system when a node fails. Snooze integrate a heartbeat protocol to detect node failures. Snooze VM consolidation is applied by GM on its partial view of the system. Although it allows the scaling on the number of nodes consolidation becomes only local.

3.2.3 Strucured P2P

Distributed VM Scheduler (DVMS) [QLS+12] is a framework for scheduling VMs in a co-operative and dynamic fashion in distributed systems. Their consolidation mechanism is applied inside a strucured P2P network. They use a ring (see Figure 3.2.3) to forward events such as overload or underload situation to the next neighbor $i+1$. If the next neighbor fails to compute a new schedule then the request is forwarded to neighbor $i+2$. The system

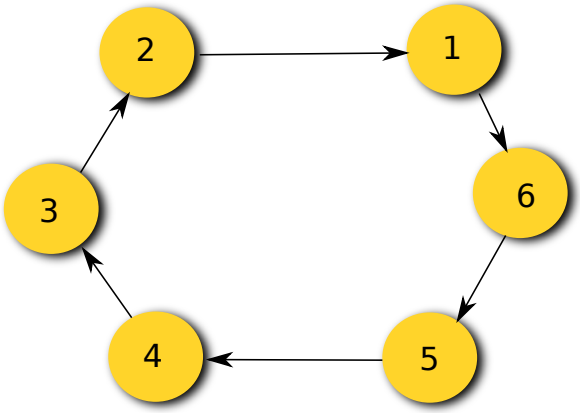
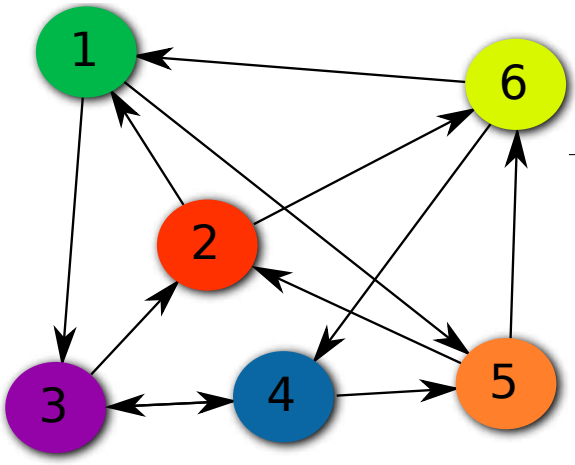


Figure 4: Ring topology

has only been simulated (no real networks communication and concurrency). The solution is currently not fault tolerant and the scalability in terms of number of nodes has not been proven yet.

3.2.4 Unstructured P2P



Node	Neighborhood
1	{3,5}
2	{1,6}
3	{2,4}
4	{3,5}
5	{2,6}
6	{1,4}

Figure 5: Overlay of six nodes and neighborhood size of 2

Figure 3.2.4 shows an example of a untructured P2P overlay. A neighborhood is contructed for each node in the system. The table represents for each node the list of its neighbors. Each node knows only two other neighbors: there is no global knowledge thus allowing the system to scales terms of number of nodes.

[MMP11] propose a solution for assignment and consolidation based on Bernoulli trials. Each node check at random interval over-utlization or under-utilization, in such case a request is broadcasted in the network in order to find new hosts for VMs. The cost of migration is taken into account by using probability function to smooth the number of migration over time. This solution is not fully decentralized: while decision are taken locally a centralized data center mangeer is still needed to coordinate process.

[MBP11] propose a coarse-grained gossip based VM consolidation protocol. Particularly each nodes knows only a subset of its neighborhood peers. Local VM consolidation is applied with each node of the neighborhood. This results in the migration of VMs of the least loaded node to the most loaded one. Because at each point in time the algorithm considers only two nodes the system might need a long time to converge. Moreover, in their model they consider only homogeneous VMs and do not take into account the migration cost. Finally their proposal has only been simulated.

3.3 Conclusion

No solution have been proposed that is scalable, decentralized, self-organized, heterogenous in VMs and PNs and migration cost aware. Centralized architecture expose SPOF, Hierarchical architecture currently perform only local optimization and current decentralized P2P solution have only been simulated.

4 Our Approach: Energy-Aware Epidemic VM Consolidation

We propose a scalable VM consolidation protocol that leverage a P2P gossip based membership management protocol in order to achieve self-organization and decentralization.

4.1 Key Ideas

We use Cyclon [VGVS05] as a membership management protocol. Cyclon is a peer sampling service which create a a fault tolerant and time-varying P2P random overlay in which each node knows only a subset of the nodes called neighborhood. The idea is to use traditional and well known centralized consolidation algorithms (such as FFD, Sercon or ACO) and apply them inside this neighborhood. This approach target eventual optimization. By applying several runs of local optimization with different neighbors the system tends to be optimized. This approach is fully decentralized and does not require any centralized coordination services.

4.2 Overlay Construction

Cyclon membership protocol has be designed for fast information dissemination while dealing with a high churn (i.e. the number of nodes that come in and out in the system). It is based on a periodic and random exchange of neighbors between peers the so called *shuffling operation*. Each peer keeps a local list called of neighbors*cache entries*. A cache entry contains the address of the neighbor (e.g. the IP address and the port number) and the age of the neighbor. The role of the *age* field is to bound the time an entry is chosen for shuffling in order to eliminate earlier dead peer (see [VGVS05] for more details). The shuffling operation is repeated periodically according to a parameter λt. The overlay can be viewed as an undirected connected graph where vertices represent nodes and edges $A \longrightarrow B$ represent the relation *B is a neighbor of A*. Note that in this case the relation is asymmetric (i.e. A is a neighbor of B does not imply that B is neighbor of A). The goal of the asymmetry is to reduce the probability of doing consolidation with the same neighbors shortly after a previous one.

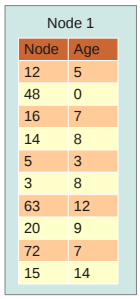
Shuffling operation of a peer P

1. Increment by one the *age* field of all entries of P's cache
2. Select the subset composed of the oldest neighbor Q according to the *age* field and other randomly chosen $l-1$ neighbors with l being a parameter that controls the number of entries that have to be shuffled.

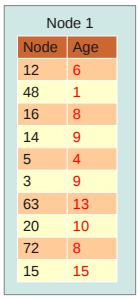
3. Replace in the selected subset the entry of Q by the address of P with *age* 0
4. Send subset to Peer Q
5. Receive from Q a subset of its entries of size i with $0 \leq i \leq l$
6. Delete in the subset received from Q any existing entries pointing at P and already contained in P's cache
7. Update P cache by adding all remaining entries. First fill empty slots and then replace entries that have been sent to Q. Any others remaining entries in the subset receive from Q are discarded.

Overlay joining process and isolation recovery As a fully decentralized system Cyclon needs a bootstrapping method for joining the overlay. It is sufficient for a node which wants to enter the system to know another nodes which is already in. Indeed a shuffling operation with a member of the overlay (called *introducer*) is enough to introduce a node in the system. We use a single node which will be contacted by any other node that wants to join the system but our software thanks to its modular architecture can accept any other mechanism such as local network broadcasting or multicast.

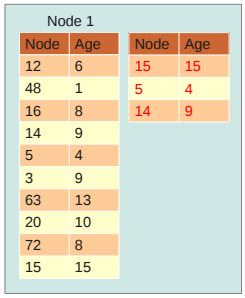
Figure 6 shows an example of a shuffling operation between two nodes. Each node has a cache entries size of 10 nodes. First node one start a shuffling operation and begins by incrementing the age of the cache entries by one. It then selects the oldest neighbor which is the node 15 with an age of 15 and two other randomly chosen neighbors (i.e. five and 14). The entry of the oldest neighbor is replaced by the one of the node one with age zero. The subset is send to the previously deleted entry (i.e. 15). Once node 15 receives this shuffling request it answer by selecting randomly three nodes from its cache entries and sends them as a response to node one. Both nodes will start merging the received subset. Node 15 delete the entry of node five and node one delete the entry of node 12 because they are already contained in their own cache. Then node 15 moves the entry of node one in its empty entry. Finally all entries remaining (i.e. 14) for node 15 and (27, 41) for node one are moved by replacing entries that have been send to the other nodes. In node one entry 41 replace five and 27 replace 15 and in node 15, 14 replaces 27.



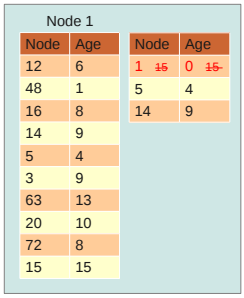
(a) Node one starts a shuffling operation



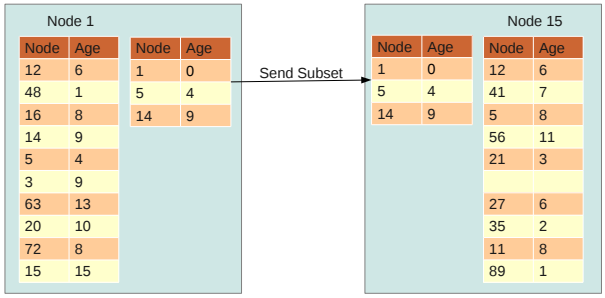
(b) Age of neighbors is incremented by one



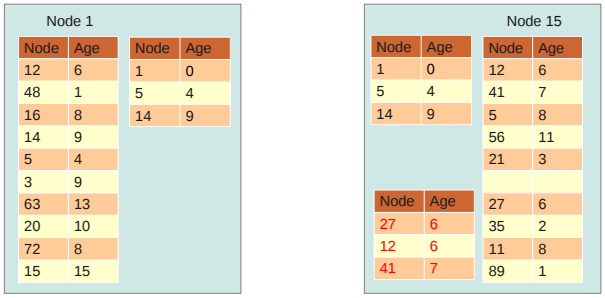
(c) Select the subset composed of the oldest neighbor 15 and two nodes randomly chosen (i.e. 5, 14)



(d) Replace in the subset the entry of 15 with one of age zero

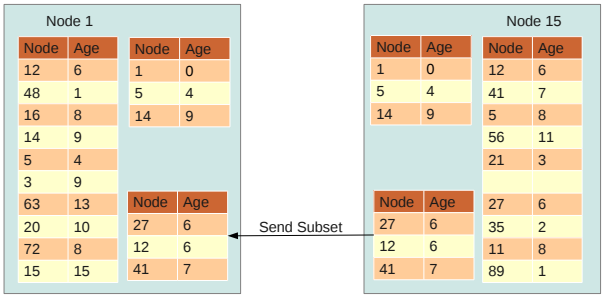


(e) Node one Send subset to node 15

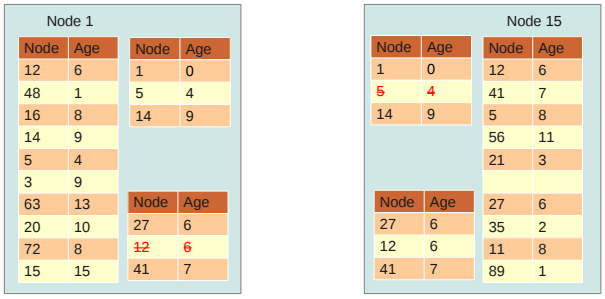


(f) Node 15 select in his subset three neighbors randomly chosen (i.e. 27, 12, 41)

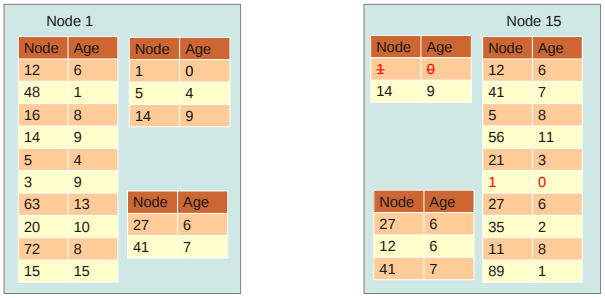
Figure 6: Overlay Construction



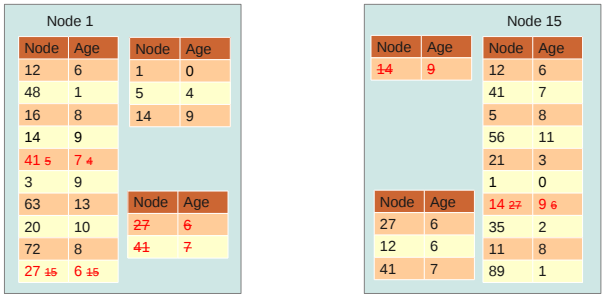
(g) Node 15 sends subset to node one



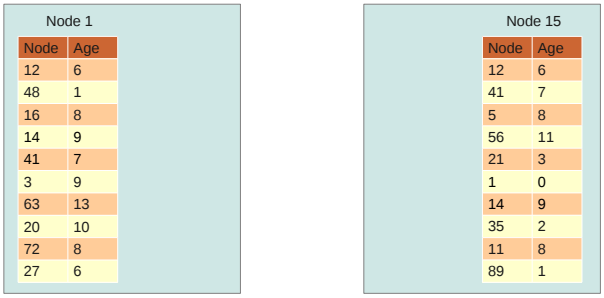
(h) Nodes one and 15 delete in their received subset existing entries pointing on them or already contains in their own cache



(i) Update cache of node one and 15 by adding all remaining entries. First fill empty slot.



(j) Then replace entries that have been sent to the other node



(k) End of shuffling

Figure 6: Overlay Construction

4.3 Consolidation Protocol

According to parameter *consolidationPeriod* each node will trigger periodically a consolidation process within his neighborhood (i.e. the cache entries of the cyclon overlay) for optimizing VMs placements. The consolidation protocol can be described in six steps as follows:

1. First the node N which start a consolidation process tries to acquire a lock for each member (including itself) of its neighborhood. For each PN a lock is associated. The goal of locks is to avoid concurrent access on PN resources in the case there would be in more than one consolidation process. Acquiring lock is a non blocking operation if it is not successful VMs and resource of the neighbors will not be taken into account for consolidation process.

2. For each successful acquisition of the lock the node N ask to the corresponding node its total capacity , VMs currently packed and resource demand vector for each of these VMs.

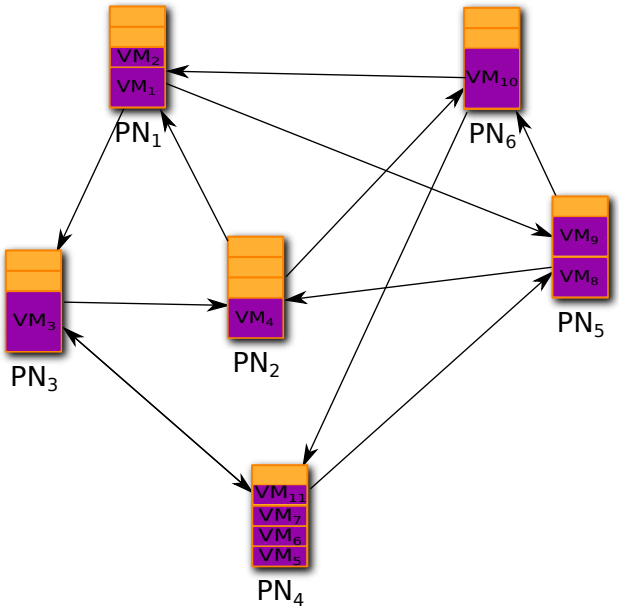
3. An optimization operation is started with all capacity vectors and VMs demands vectors associated as input. The output is a reconfiguration plan where the placement of VMs and the number of migration are optimized. Any kind of classical VMCP algorithm can be use for this operation.

4. An actuation module on the PN which initiates the consolidation enforces the reconfiguration plan by sending migration orders to PNs.

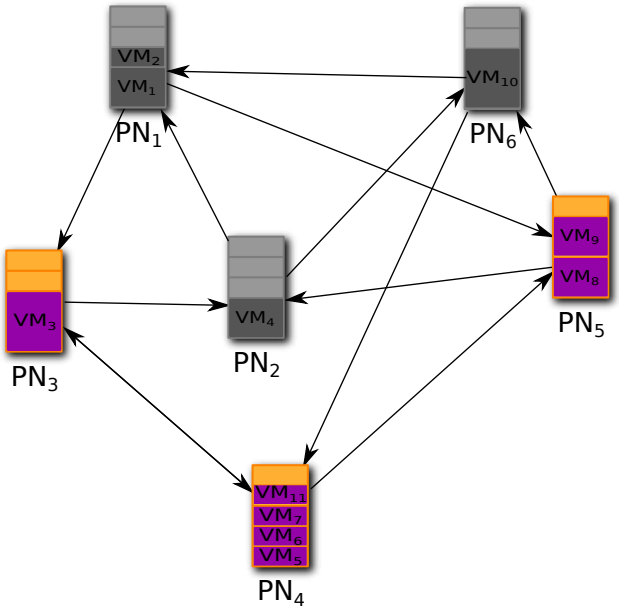
5. All locks are released.

6. Each node which does not host VMs anymore shutdown itself in order to conserve energy without warning any other PNs.

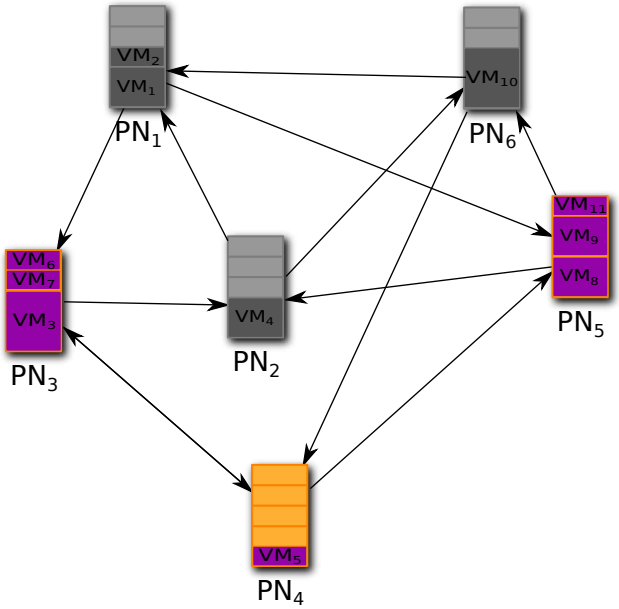
Figure 7 shows the process of reaching a global optimized state using only local optimizations. Figure 7a depicts the initial state of the cloud. There are six nodes $PN_1, \ldots, PN_6$, eleven VMs $VM_1, \ldots, VM_{11}$ that run on PNs. Neighborhood on each node is compose of two nodes. For the sake of simplicity only one resource is considered in this example : number of cores. PNs have the same capacity: five cores. VMs can request one, two or three cores. The total capacity of active nodes in the cloud is 30 slots and the current utilization is 19 slots which correspond to a utilization rate of active PNs of approximately 63%. Figure 7b shows the beginning of a local consolidation process initiated by the node PN_4 with his neighbors PN_3 and PN_5. Figure 7c shows the state after the local consolidation process. VM_{11} has been migrate from PN_4 to PN_5 and, VM_6, VM_7 have been migrated from PN_4 to PN_3. PN_5 to PN_3 are now full and therefore their resources are better utilized than in



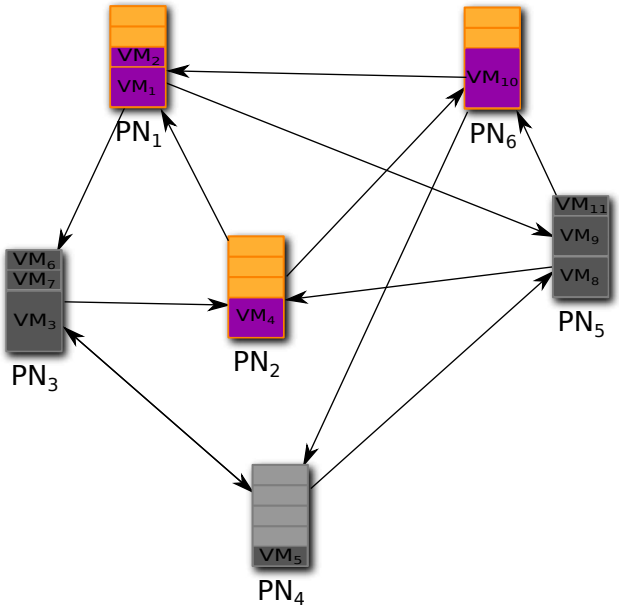
(a) Initial configuration



(b) PN_4 triggers consolidation with PN_3 and PN_5

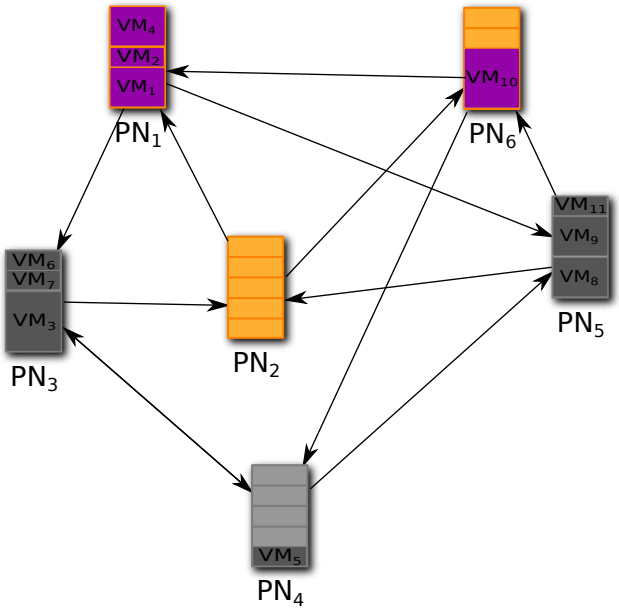


(c) PN_3, PN_4, PN_5 after reconfiguration

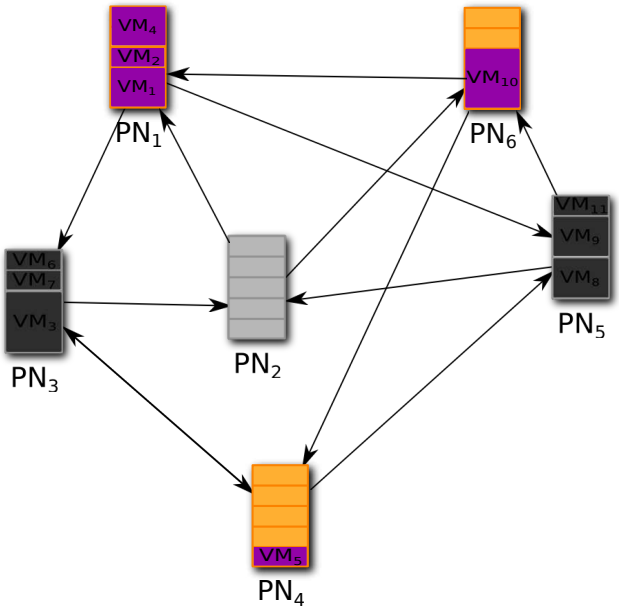


(d) PN_2 triggers consolidation with PN_1 and PN_6

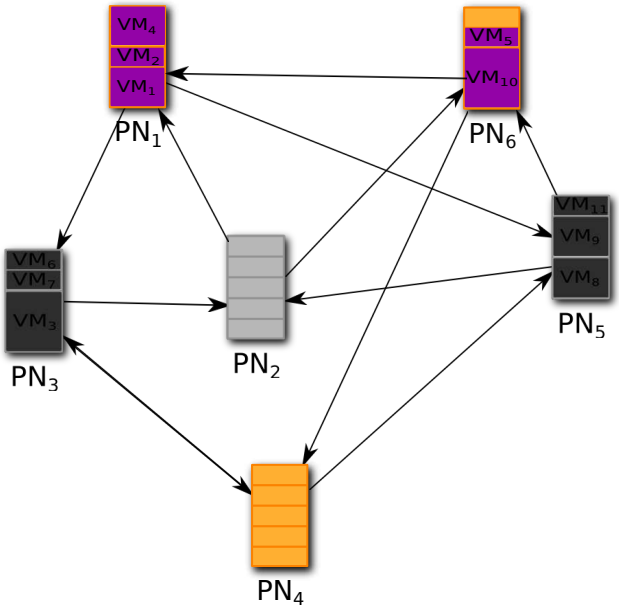
Figure 7: Global Optimization



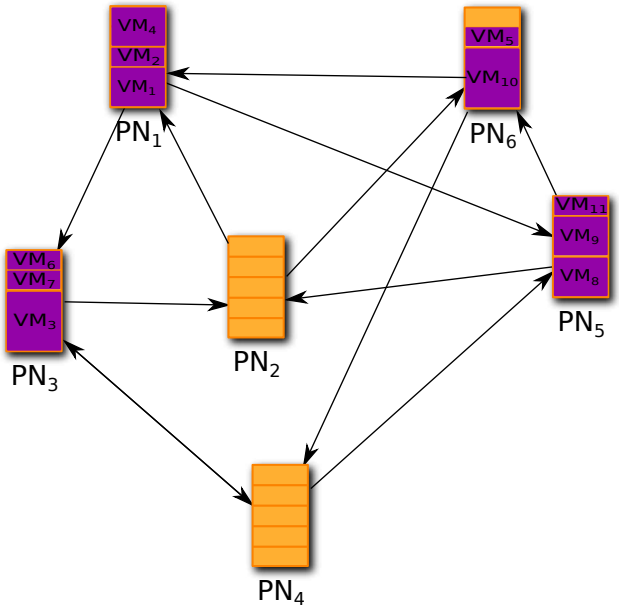
(e) PN_1, PN_2, PN_6 after reconfiguration



(f) PN_6 triggers consolidation with PN_1 and PN_4



(g) PN_1, PN_4, PN_6 after reconfiguration



(h) Global optimization

Figure 7: Global Optimization

the previous configuration. In Figure 7d PN_2 trigger a local optimization with is neighborhood composed by PN_1 and PN_2. After reconfiguration Figure 7e show that VM_4 has been moved from PN_2 to PN_1. PN_1 is now full and PN_2 is empty. PN_6 start in figure 7f another consolidation with PN_1 and PN_4. After this consolidation Figure 7g shows that VM_5 has been moved from PN_4 to PN_6. The node PN_4 is now empty and PN_6 is better used than in the previous state. The final state is shown in Figure 7h where they are two idle nodes than can be transitioned into an energy saving state. The new utilization is now 95% and therefore leads to a better resources utilization in the data center.

While we can use any of traditional centralized algorithms for optimizing the placements of VMs we have designed a new version of the ACO algorithm for VMCP which is migration-cost aware and specifically designed to be integrated into our Consolidation protocol in order to improve the efficiency of the ressource utilization in the data center.

4.4 Migration-cost aware ant colony optimization

We propose a new ACO algorithm which solves the VMCP. It is based on the [FRM11a] algorithm but differs in several points: first we take into account the number of migrations while optimizing the placements of VMs, then we consider the current state for placing VMs (i.e. the current mapping of items to bins).

In ACO the probability for an ant to pack an item i into a bin v is:

$$p_{i,v} = \frac{[\tau_{i,v}]^\alpha \times [\eta_{i,v}]^\beta}{\sum_{u \in N_v} [\tau_{u,v}]^\alpha \times [\nu_{u,v}]^\beta} \tag{1}$$

Where $\tau_{i,v}$ represents the amount of pheromone for packing i into v, $\eta_{i,v}$ the heuristic information. α and β are parameters > 0 to emphasize either more the pheromone or the heuristic information.

We define the migration cost $\xi_{i,v}$ of moving a VM i from host to another as the total number a migration needed to reach the final destination host v.

$$\xi_{i,v} := |Mplan| \tag{2}$$

Where Mplan is the list of migrations. In order to minimize the number of bins used, items which utilize the bins betters are favored

$$\kappa_{i,v} := \frac{1}{|\vec{C_c} - (\vec{b_v} + \vec{r_i})|_1} \tag{3}$$

Where $\vec{b_v}$ is the load vector of the bin v. It is computed as follow:

$$\vec{b_v} := \sum_{i \in B_v} \vec{r_i} \tag{4}$$

Preference of minimizing the number of bins and migration cost are combined to compute the heuristic information according to two parameters e and p in order to emphasize more the number of bins or the cost of migrations. It is equal to zero if by packing item i into v total capacity of v is exceeded.

$$\eta_{i,v} := \begin{cases} \frac{\kappa_{i,v}^{p}}{\xi_{i,v}^{e}} & \text{if } \vec{b}_v + \vec{r}_i \leq \vec{C}_v \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The simulation of pheromone evaporation where $0 \leq p \leq 1$ represent the evaporation parameter where the more p is increased the more evaporation rate is increased.

$$\tau_{i,v} := (1-p) \times \tau_{i,v} + \Delta_{\tau_{i,v}}^{BestSolution}, \forall (i, B_v) \in I \times B \tag{6}$$

Where $\Delta_{\tau_{i,v}}^{BestSolution}$ represent the best (item,bin) association of the iteration.

$$\kappa_{i,v} := \begin{cases} \frac{1}{f(BestSolution)} & \text{if } x_{i,v} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Whereby *BestSolution* is the best solution which correspond to the best migration plan and $x_{i,v}$ is the bin allocation variable which is equall to 1 if item i is assigned to bin v and 0 otherwise. In order to avoid early stagnation which could follow to bad solution, a lower(τ_{min}) and upper(τ_{max}) bound are used for the pheromone value.

Finally in order to compare solutions we define a new objective function which takes into account the number of released bins (i.e bins which do not host any items) and the total number of migrations to reach the new state as well as the variance on the normed load vector of each bins. This last metrics has designed especially for the gossip based consolidation protocol. The goal of promoting these solutions is to create differences in load between bins for *propagating optimization* better for later consolidation between different items and bins.

$$f(Mplan) = \left(n - \sum_{v=0}^{n-1} y_v \right)^{e} \times \left(\frac{1}{|Mplan|} \right)^{p} \times \left(\mathrm{Var}(|\vec{b}_0|_1,\ |\vec{b}_1|_1,\ \ldots,\ |\vec{b}_{n-1}|_1) \right)^{g} \tag{8}$$

Where g is new parameter introduced in order to promote the variance of loads.

Algorithm 1 describes the pseudocode our ACO-based algorithm. First the algorithm takes as input the items and there resource demand vectors, bins and there capacity vectors and current mapping between items and bins. The function *optimize()* call for each ants in each cycle the function *ant_ build_ solution()* which construct solution (i.e. a migration plan of VMs). After each cycle the function *trail_ update_ ()* update the pheromone trail matrix in order to control the behavior of ants. tuple of (VM,PN) (i.e. (item,bin)) that appear in the *CycleBestSolution* will be reinforced in pheromone matrix while others will

Algorithm 1 Ant Colony Optimization

function $optimize(I,B,\vec{r}_i,\vec{C}_v)$
  $BestSolution := nil$
  for all $q \in 0...nCycles-1$ **do**
    $CycleBestSolution := nil$
    for all $a \in 0...nAnts-1$ **do**
      $s := ant\_build\_solution(I, B, \vec{r}_i, \vec{C}_v)$
      Update $CycleBestSolution$ according to the objective function $f(s)$
    end for
    Update $BestSolution$ with $f(CycleBestSolution)$
    $trail\_update()$
  end for
  return $BestSolution$
end function

function $trail\_update(\tau)$
  for all $(i, B_v) \in I \times B$ **do**
    $\tau_{i,v} := (1-p) \times \tau_{i,v} + \Delta^{best}_{\tau_{i,v}}$
    if $\tau_{i,v} > \tau_{max}$ **then**
      $\tau_{i,v} := \tau_{max}$
    end if
    if $\tau_{i,v} < \tau_{min}$ **then**
      $\tau_{i,v} := \tau_{min}$
    end if
  end for
end function

function $ant\_build\_solution(I,B,\vec{r}_i,\vec{C}_v)$
  $Mplan := nil$; $BestPlan := nil$
  while $|Mplan| <$ number of VMs **do**
    Compute $p_{i,v} \quad \forall i \in I, \forall v \in B$
    Choose (i, v) stochastically according to probability: $p_{i,v}$
    $\vec{b}_w := \vec{b}_w - \vec{r}_i$ with w the previous bin where i was packed
    $x_{i,w} = 0$
    Add (i, w, v) to $Mplan$
    $\vec{b}_v := \vec{b}_v + \vec{r}_i$
    $x_{i,v} = 1$
    if $Mplan$ is better than $BestPlan$ according to the $f(Mplan)$ **then**
      $BestPlan := Mplan$
    end if
  end while
  return $BestPlan$
end function

be weakened (i.e. evaporation mechanism) and thus reinforced tuple will be more likely to be chosen by ants in future building of solutions. *ant_ build_ solution()* construct a feasible solution (i.e. a solution which take into account capacity of PNs and estimated demands of VMs). In this function contrary to the previous algorithm in [FRM11a], we take into account the current mapping between VMs and PNs this mean that at the beginning of the function we are already in a feasible state and returning an empty migration plan is a valid solution. Indeed when the placement of VMs are already in an optimal or nearly optimal state doing nothing is the best solution. This function does a local search by trying to place VM according to the Equation 1. For each items i, and bins v the probability to pack i into v is computed . one item and one bin is chosen stochastically according to these probability. Then update the load by decreasing the load of previous bin and increasing the load of the current bin by the estimated demand of the item. The allocation variable is also updated to take into this migration. The migration is then added to migration plan and if this updated migration is better than the previous one (according to the objective function $f()$ defined in Equation 8) the new migration plan is copied in $BestPlan$. On line 30 We limit the time an ants pass in the local search by bounding the number of migration by the number of VMs , this let a chance to move all VMs in one consolidation while limiting the cost of computing the *ant_ build_ solution()* function. Note that a VM can be choosed for migration more than once if this lead to a better placement.

5 Experimental Results

5.1 Prototype implementation

This solution has been emulated. The emulator is implemented in the Python programming language. Each PN is run inside a system process and is bind to an IP address and a port number. Network communication are not simulated and occur in a concurrent way. The prototype implements distributed locks to prevent concurrent access to resources. PN shutdown themselves when they do not host VMs anymore without warning other PNs, hence our implementation is fault tolerant: the peer sampling service eliminates dead nodes (nodes shutdowned) from neighborhood and PNs that do not respond when they are contacted for getting resources during consolidation are not taken into account. If a node crashes during the enforcement of the migration plan, the consolidation is aborted (i.e. we stop any further migration) and VMs on the faulty nodes are lost. VMs are simulated and represented by their resource demand vector as Python objects. Each PN writes events (e.g. migration, consolidation, shutdown) in a local SQLite database during execution. Once an experiment is terminated all databases are collected and merged into a single one for later analysis. The software architecture is modular: components such as (introducers mechanism, algorithm for consolidation, scheduler for shuffling and consolidation) are defined in a configuration file and their replacement do not impact the rest of the software. We have implemented three algorithms: our ACO algorithm, Sercon and FFD as a baseline algorithm. While ACO and Sercon compute a migration plan at the same they optimize the number of nodes used and take into account the number of migrations, FFD is only a simple heuristic for consolidation, so we implemented a simple migration plan algorithm which take the output of FFD and computes a migration plan according of previous placement of each VMs but may violate SLA (i.e. exceeds the total capacity of a node).

5.2 System Setup

Our system was deployed on 42 HP Proliant DL165 G7 servers on the Grid5000 testbed [CDD+05]. All servers have two AMD Opteron 6164 HE 1.7GHz CPU with 12 cores each (for a total number of 1008 cores). Because the consolidation algorithms are not parallel we can simulate one PN per core.

In these experiments we consider three types of resources: memory in Gigabyte, CPU in *computing unit*, I/O in Gigabit/s. Computing unit are inspired from EC2 computing unit in which each unit has the capacity of a one core 1.2GHz Xeon processor in 2007 [Ama]. We consider 6 kind of VMs workload (nano, micro, small, medium, large, xlarge) whose estimated demands is (0.2, 0.5, 0.1), (1, 1, 1), (2, 1, 1), (4, 2, 2), (8, 4, 4), (16, 8, 4) respectively. These value represents differents workload of VMs inspired from EC2 VMs. In

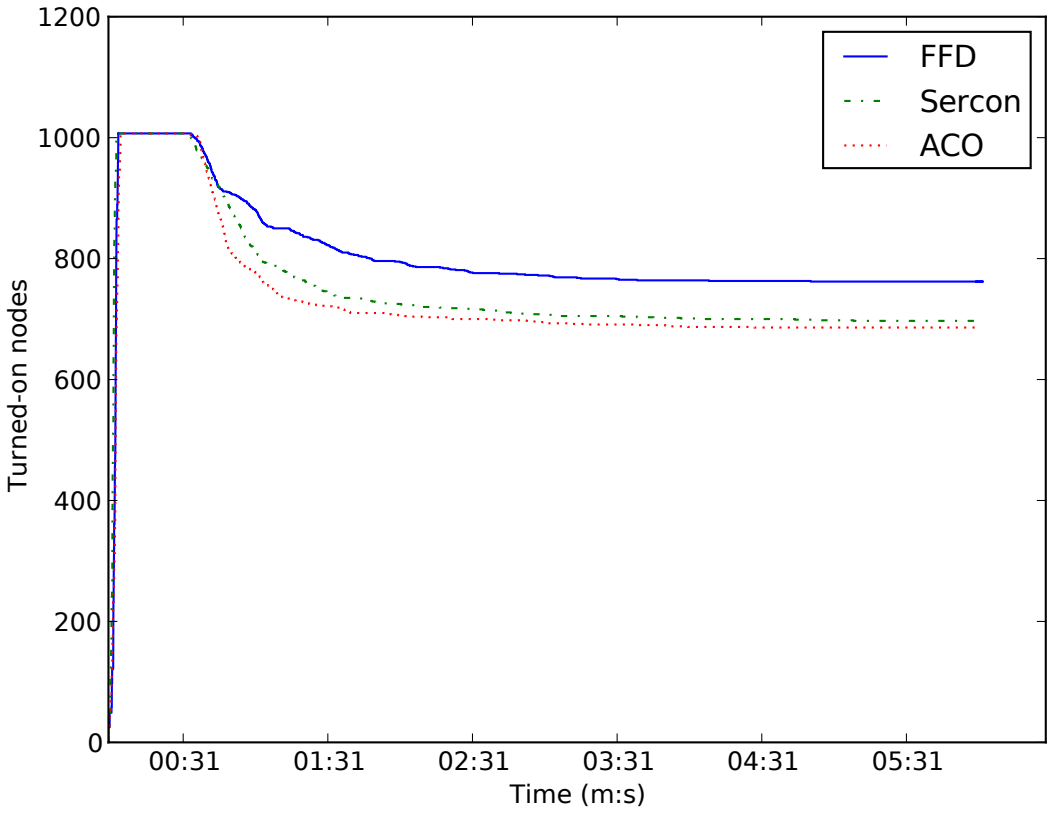


Figure 8: Comparison using different algorithms

these experiments the estimated demand of VMs is static (i.e. does not vary in time). The default configuration run 1008 PNs with 6048 VMs. Each PN has a capacity of (48, 26, 20) and hosts 6 VMs, one of each type and the average utilization of the nodes is 0.64. Each experiment runs for six minutes. Consolidation is triggered by PNs each 30 seconds. The neighborhood max size is fixed to 16 nodes and the shuffling operation is triggered each 10 seconds for each PNs.

5.3 Scalability and Energy Consumption

Figure 5.3 shows the overall result of the experiment with the default parameters for each of the consolidation algorithm (FFD, Sercon, ACO). The number of turned-on nodes in terms of time is represented. First all PNs process are started in parallel. This operation took two seconds before all the 1008 nodes are started. The scheduler of each PNs start consolidation after 30 seconds. Between 00:31 and 01:31 the number turned-on nodes fastly decreased for

all algorithms. At the end of the experiment ACO succeed to release 322 nodes, Sercon 311 and FFD 246. We see that our ACO algorithm outperforms the both greedy algorithms. the experiment shows that for high number of nodes (1008) our P2P solution is able to optimize the overall resource utilization of the cloud.

In order to see the benefit on energy consumption to shutdown unused nodes we estimate energy consumption according to this model described in [FRM11a]:

$$P(u) = (P_{max} - P_{idle}) \times u + P_{idle} \tag{9}$$

P_{idle} and P_{max} fixed to 171 and 218 represent the power consumption of a two CPU dual core Intel Xeon 5148 2.33GHz when the node is idle and fully utilized, respectively measured by using energy consumption sensing units on the testbed.

$$E(B) := \begin{cases} t \times \sum_{v=0}^{n-1} P(\frac{|\vec{b}_v|_1}{d}) & \text{if } |\vec{b}_v|_1 \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Equation 10 gives the global energy consumption of all nodes. Nodes that have been shutdown because they don't host VMs do not consume any energy anymore.

Figure 5.3 shows the estimated energy consumption for the default configuration for each algorithm. When all nodes are started the cloud consumes 202kWh. Using our distributed consolidation protocol we were able to reduce the consumption to 137kWh using ACO algorithm, 140 for Sercon and 153 with FFD. Because no VMs are created or deleted during the experiment the estimated energy consumption depend only on the number of node released thus it is not surprising that the figure has the same shape as the Figure 5.3.

The number of migrations for the different algorithms is presented in Figure 5.3. FFD produces 96494 migrations, Sercon 1872 and ACO 4015. The number of migrations produced by FFD is huge and would lead to performance degradation this is causes by the fact that FFD is not a cost aware algorithm. Sercon produces the lowest number of migrations. Sercon tries to migrate all VMs from one node to other nodes if it can't do so the whole operation is canceled. This means that it migrates only if migrations will lead to release the node. When comparing Figure 5.3 with Figure 5.3 we can see that for Sercon and ACO when there is no nodes released the number of migrations is null while FFD algorithm always performs migrations even if it does not lead to released nodes.

In order to study the impact of using a P2P consolidation protocol we have compared the traditional algorithms (ACO, Sercon, FFD) within our distributed consolidation protocol and within the centralized architecture. Figure 5.3 shows the results for 1008 nodes with 6048VMs. The gain shows the ratio in percent of the number of released nodes on the total number of nodes. ACO in the centralized architecture is too slow to compute a solution for this number of VMs and PNs. We stoped waiting the results after 1 hour. Each of the other algorithms in there P2P distribution is able to reach nearly the same quality of optimization

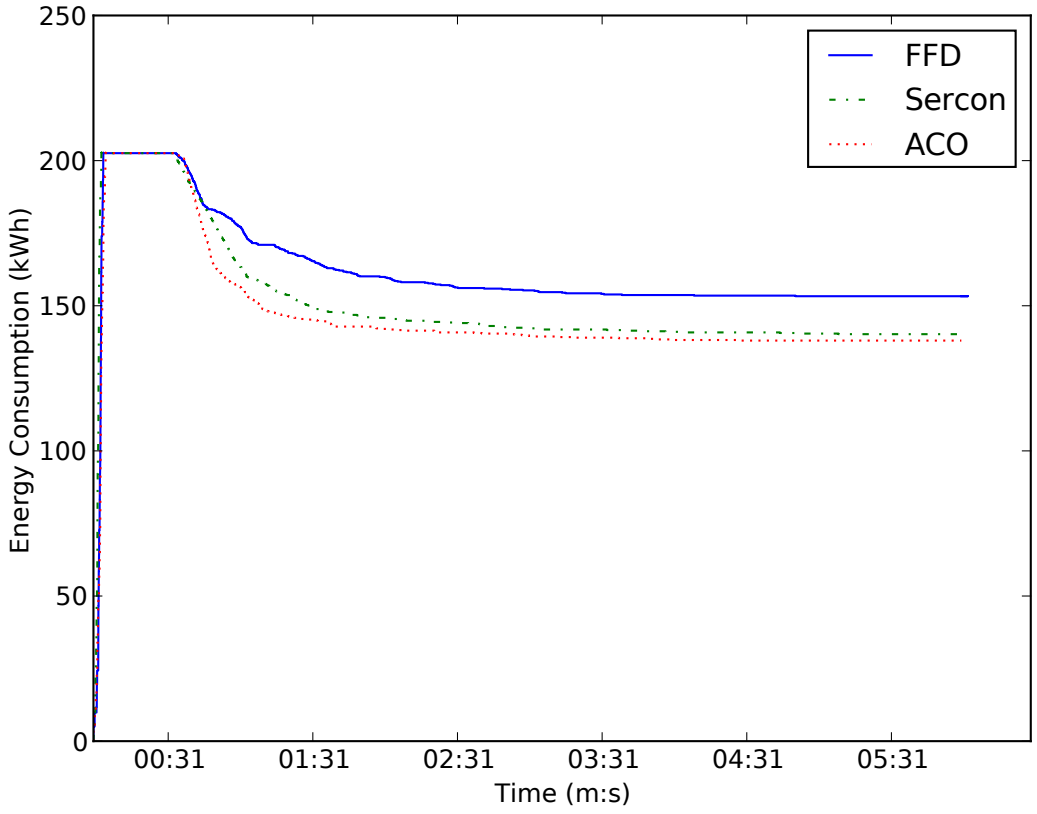


Figure 9: Energy consumption for different algorithms

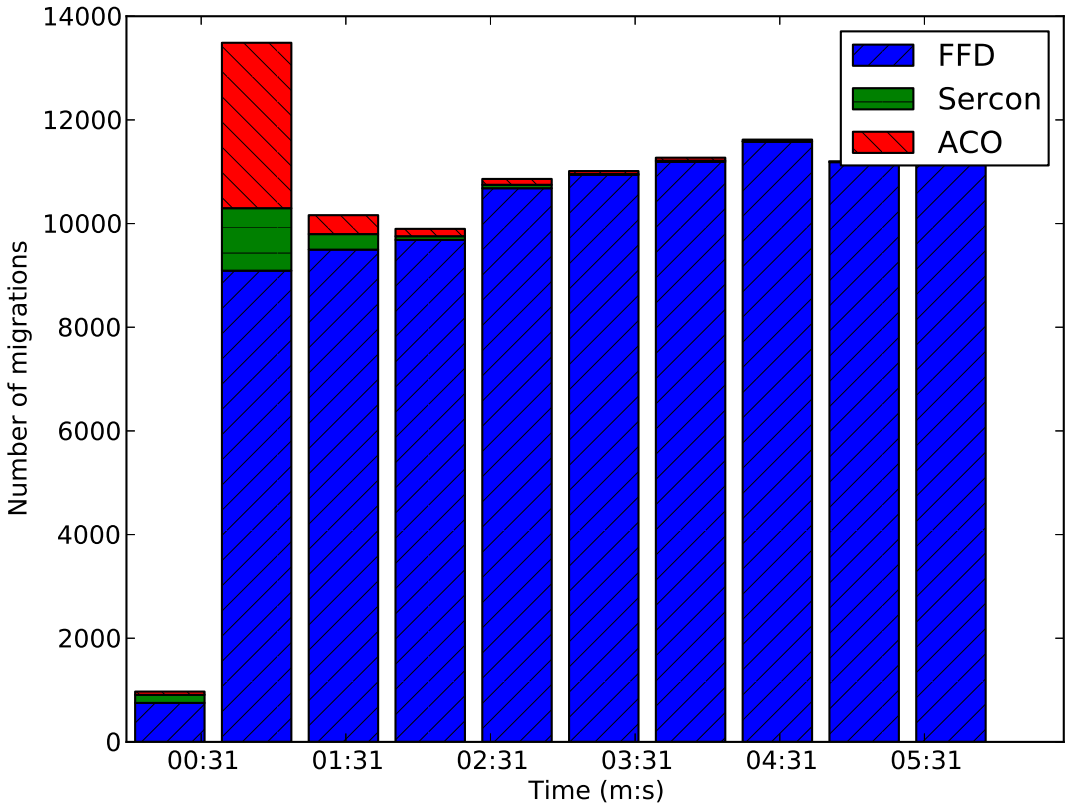


Figure 10: Number of migrations for different algorithms

Distribution	Algorithm	Execution time(s)	Number of migrations	Released nodes	Gain(%)
Centralized	FFD	19.1	6040	249	24.7
	Sercon	23.0	1920	320	31.7
	ACO	-	-	-	-
P2P	FFD	360	96494	246	24.4
	Sercon	360	1872	311	30.8
	ACO	360	4015	322	31.9

Figure 11: Comparison with centralized algorithms

Algorithm	Number of nodes	Number of VMs	Released nodes	Gain (%)
FFD	120	720	29	24.1
	240	1440	58	24.1
	504	3024	124	24.6
	1008	6048	246	24.4
Sercon	120	720	37	30.8
	240	1440	74	30.7
	504	3024	155	30.8
	1008	6048	311	30.8
ACO	120	720	36	30.0
	240	1440	77	32.0
	504	3024	161	31.9
	1008	6048	322	31.9

Figure 12: Scalability

(i.e. number of nodes released) that its centralized equivalent. This demonstrates that the impact of using our P2P system on the quality of the solution is low and with algorithms running several consolidation on subset of PNs we reach the same global optimization than with the global centralized execution. The fact that unused nodes shutdown themselves without warning any other node does not impact the solution and dead nodes (i.e. nodes shutdowned) are eliminated gradually the peer sampling service, this make our system fault tolerant between consolidation.

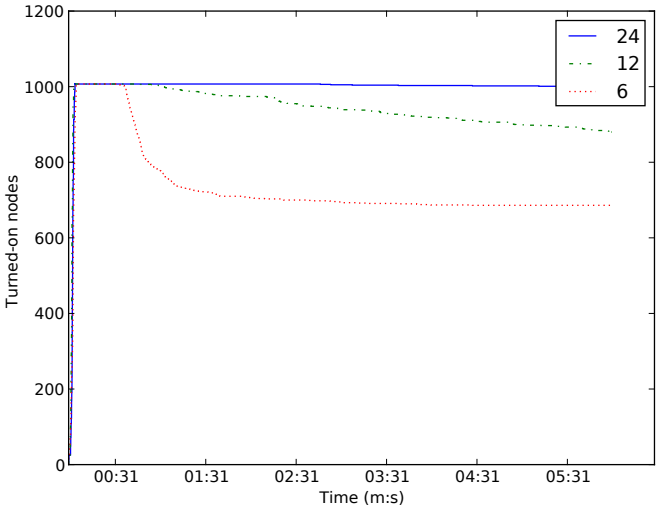
Our P2P solution aims to be scalable on the number of nodes so we run the experiment with clusters of differents size (120, 240, 504, 1008) nodes, each nodes starting with default configuration (6 VMs). Figure 5.3 shows the results of these experiments. ACO does not give the same number of node released because it was too slow to compute a solution. The gain shows the ratio in percent of the number of released nodes on the total number of nodes. Results show that when the cluster is growing the ratio between the number of nodes released on the total number of nodes does not change significantly for any of the algorithm tested. Our solution is therefore scalable in terms of number of nodes.

We test the scalability of our solution in terms of the number of VMs. We compare the default configuration with 1008 PNs with six VMs per node with 2 other initial configurations: with 12 VMs and 24 VMs per nodes. In the 12 VMs per node configuration each node start with three nano, three micro, three small VMs and three medium VMs and the average utilization per nodes is 0.48. In the 24 VMs per nodes configuration each node start with eight nano, eight micro and eight small VMs and the average utilization per nodes is 0.62. Results are presented in Figure 5.3 for each of the algorithms. Because the average utilization is lower in the 12 VMs configuration it not surprising that FFD and Sercon per-

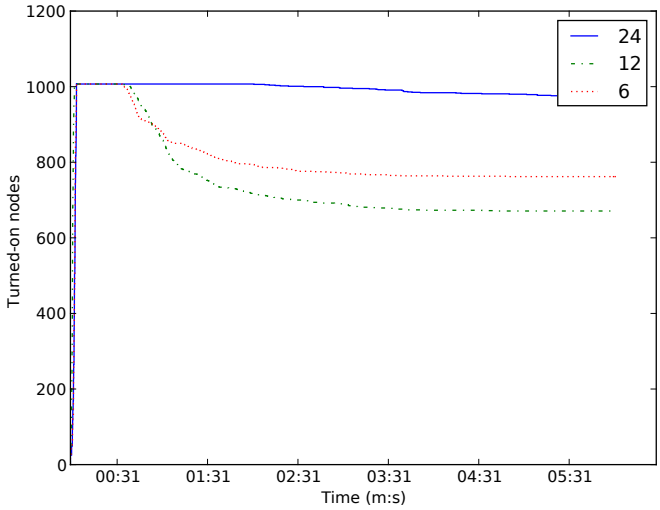
form better than in the six VMs configuration. ACO was too slow to compute a solution in a reasonable amount of time.

Finally we study the impact of the maximum number of peers choosen for consolidation. We compare the reference number of peers choosen during consolidation: 16 with three others values two, four and eight. Figure 5.3 shows results for each of the algorithms. Bigger size of neighborhood gives a faster convergence. The difference between eight and 16 is not very significative for ACO and sercon. With two neighbors per node the convergence time for ACO is slower but reaches at least the same number of realeased node of FFD. Sercon is not able to perform any optimization. This is due to fact that Sercon migrates VMs from one node only if all VMs can be allocated to other nodes.

ACO



FFD



Sercon

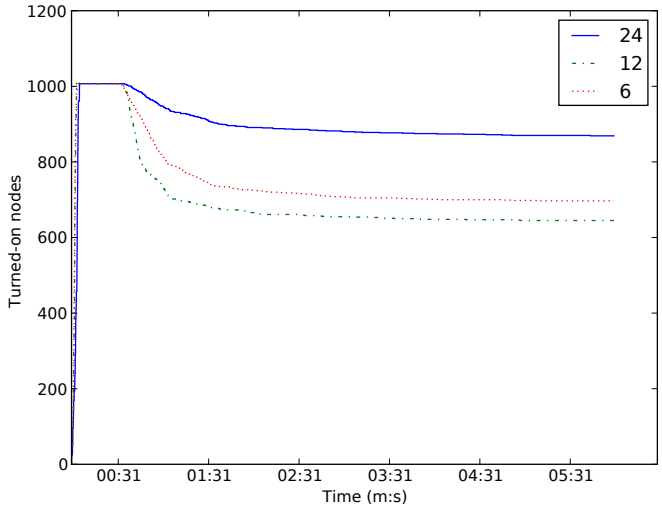
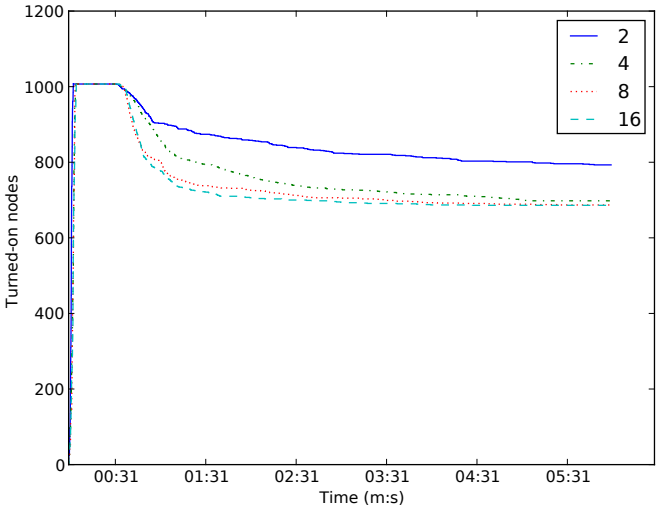
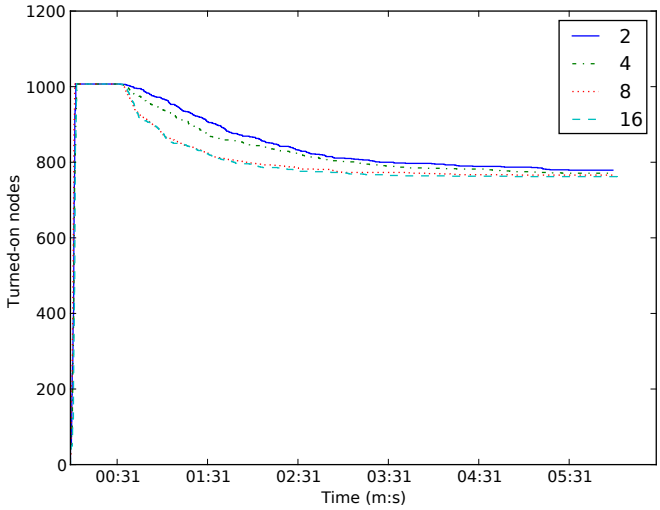


Figure 13: Impact of the number of VMs per nodes

ACO



FFD



Sercon

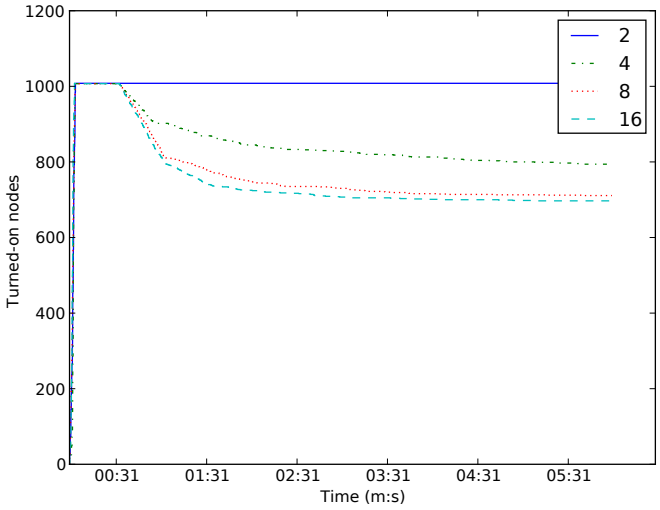


Figure 14: Impact of the number of neighbors

6 Conclusions and Future Work

We have designed a fully distributed gossip based and fault tolerant protocol and a migration cost aware of an ACO algorithm for solving VMCP in IaaS clouds. We maximize utilization of the data center thus allowing energy saving while meeting resource requirements. It leverage a peer sampling with traditional centralized algorithm. We implemented our proposal with real network communication and consolidation computation that occur concurrently, and three consolidation algorithms (FFD, Sercon and ACO). Finally we evaluated our prototype by analyzing trace produced during execution. We studied scalability of our solution as well as the number of nodes released and its impact on energy consumption. We also study the impact of the number of nodes choosen per consolidation. We have shown that using traditional centralized algorithms (e.g. ACO, Sercon) on this protocol makes consolidation scalable on the number of nodes while keeping the same quality of consolidation. Our specially designed Ant colony algorithm gives better results on the numbers of node released than Sercon and FFD. Sercon algorithm has been specifically designed to reduce the number of migrations and create the lowest number of migrations while giving better result than FFD. The number of migrations created by FFD in our distributed protocol make it unusable in our P2P protocol due to the fact that it always create migrations even if they do not lead to node releasing.

In the future we plan to integrate this protocol into a cloud management framework such as Snooze [FRM11b] or OpenStack [Ope] in order to measure the impact of the number migrations and the real energy gains. We have tested fault tolerance between consolidation (nodes shutdown themselves without warning) and we plan to test it during consolidation (e.g. when a node crashes during a migration). Our consolidation protocol is fully decentralized (no SPOF) but the joining process in the P2P overlay is not (utilization of one or more bootstrap nodes). The use of periodic broadcasting according to the network size could be used in order to avoid centralized joining. We have introduced two migration cost-aware algorithm for optimization (i.e. ACO and Sercon) but they take only into account the number of migrations, others metrics could be used in order to improve the estimation of the cost of the migration such as the memory demand of a VM or the current bandwith usage in the network. Sercon and our ACO algorithm compute SLA compliant but sequential migration plans. It would be interesting to allow independent migrations to run in parallel in order to decrease the time of enforcing this migration plan.

References

[Ama] Amazon. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[ANV11] P. Ahuja, A. Neogi, and A. Verma. pmapper: Power and migration cost aware application placement in virtualized systems. *IFIP Lecture Notes in Computer Science (LNCS)*, 5346(5346):243–264, 2011.

[CDD+05] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jégou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, et al. Grid'5000: a large scale, reconfigurable, controlable and monitorable grid platform. 2005.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[Dor92] M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[FRM11a] Eugen Feller, Louis Rilling, and Christine Morin. Energy-Aware Ant Colony Based Workload Placement in Clouds. In *The 12th IEEE/ACM International Conference on Grid Computing (GRID-2011)*, Lyon, France, September 2011.

[FRM11b] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. Rapport de recherche RR-7833, INRIA, December 2011.

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[JRL08] N. Jussien, G. Rochart, and X. Lorca. The choco constraint programming solver. In *CPAIOR08 workshop on OpenSource Software for Integer and Contraint Programming OSSICP08*, 2008.

[KKL+07] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[MBP11] M. Marzolla, O. Babaoglu, and F. Panzieri. Server consolidation in clouds through gossiping. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–6. IEEE, 2011.

[MMP11] C. Mastroianni, M. Meo, and G. Papuzzo. Self-economy in cloud data centers: statistical assignment and migration of virtual machines. *Euro-Par 2011 Parallel Processing*, pages 407–418, 2011.

[MO11] Aziz. Murtazaev and Sangyoon. Oh. Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review*, 28(3):212–231, 2011.

[Ope] OpenStack. Open source software for building private and public clouds. `http://openstack.org/projects/compute/`.

[QLS+12] F. Quesnel, A. Lèbre, M. Südholt, et al. Cooperative and reactive scheduling in large-scale virtualized platforms with dvms. *Concurrency and Computation: Practice and Experience*, 1:2, 2012.

[Sha04] P. Shaw. A constraint for bin packing. *Principles and Practice of Constraint Programming–CP 2004*, pages 648–662, 2004.

[VGVS05] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[WSVY07] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. NSDI*, volume 7, pages 11–13, 2007.

[Yue91] M. Yue. A simple proof of the inequality ffd (l)$\leq$ 11/9 opt (l)+ 1,$\forall$l for the ffd bin-packing algorithm. *Acta Mathematicae Applicatae Sinica (English Series)*, 7(4):321–331, 1991.