Server Consolidation in Clouds through Gossiping

Moreno Marzolla, Ozalp Babaoglu, Fabio Panzieri
Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura A. Zamboni 7, I-40127 Bologna, Italy
Email: {marzolla, babaoglu, panzieri}@cs.unibo.it

Abstract—**The success of Cloud computing, where computing power is treated as a utility, has resulted in the creation of many large datacenters that are very expensive to build and operate. In particular, the energy bill accounts for a significant fraction of the total operation costs. For this reason a significant attention is being devoted to energy conservation techniques, for example by taking advantage of the built-in power saving features of modern hardware. Cloud computing offers novel opportunities for achieving energy savings: Cloud systems rely on virtualization techniques to allocate computing resources on demand, and modern Virtual Machine (VM) monitors allow live migration of running VMs. Thus, energy conservation can be achieved through *server consolidation*, moving VM instances away from lightly loaded computing nodes so that they become empty and can be switched to low-power mode. In this paper we present V-MAN, a fully decentralized algorithm for consolidating VMs in large Cloud datacenters. V-MAN can operate on any arbitrary initial allocation of VMs on the Cloud, iteratively producing new allocations that quickly converge towards the one maximizing the number of idle hosts. V-MAN uses a simple gossip protocol to achieve efficiency, scalability and robustness to failures. Simulation experiments indicate that, starting from a random allocation, V-MAN produces an almost-optimal VM placement in just a few rounds; the protocol is intrinsically robust and can cope with computing nodes being added to or removed from the Cloud.**

I. Introduction

Many large-scale IT services are relying on Cloud infrastructures to host applications and to process data. Clouds use virtualization techniques to provide computing resources as utilities: users can request CPU power, storage space, or access to applications and only pay for their use as needed. Resources which are no longer needed can be released at any time. From the user's perspective, Cloud computing allows access to resources "on demand", without the need to acquire, provision or maintain them. Furthermore, users only pay for what they actually use, and thus can make optimal utilization of resources [1].

A Cloud service is physically hosted inside big datacenters, containing a large number of computing nodes. The energy requirement of the whole datacenter (for illumination, power supply, cooling and so on) is a significant fraction of the total operating costs [2]. Thus, reducing the energy consumption is becoming an important issue [3], both for economical reasons (reducing costs) but also for making IT services environmentally sustainable.

In this paper we address the problem of reducing the power consumption of Cloud infrastructures by moving VMs on a limited subset of the available (physical) computing resources, so that the remaining (idle) computing nodes can be switched to low power consumption modes. The process of aggregating services running on multiple servers into a reduced number of more powerful servers is known as *server consolidation*. In this paper we use the term *VM consolidation* to denote the consolidation of multiple VMs on a reduced number of physical hosts.

For example, let us consider the situation depicted in Figure 1. We have three hosts, each of which has a quad-core processor which is capable of executing four VMs. The system is currently hosting four VM instances labeled VM_1–VM_4. The allocation shown in Figure 1(a) is not power efficient, because all hosts have low utilization; it is known that most current hardware is power inefficient under light load [2]. Thus, it makes sense to move VMs on fewer, highly utilized servers so that the remaining (empty) servers can be put in power-saving mode. In Figure 1(b) we show the effect of migrating VM_1 and VM_4 to host 2. Hosts 1 and 3 can now be switched to power-saving states, until new VMs need to be allocated.

In this paper we present V-MAN, a fully distributed algorithm for VM consolidation on Cloud systems. V-MAN is based on a simple gossip protocol that does not require any central coordinator or globally shared data structure. V-MAN is completely VM and application agnostic; it does not require any instrumentation of either the VMs or the hosted applications. V-MAN is executed periodically to identify a new arrangement of existing VM instances so that the number of empty servers is maximized. Once the new allocation has been identified, it is possible to migrate VM instances to their final destination using the live migration feature provided by most Virtual Machine monitors (e.g. Xen [1], OpenVZ [2] and VMware [3]). Simulation results show good scalability and high resiliency to failures.

This paper is organized as follows. In Section II we review existing results from the scientific literature. In Section III we formally define the problem we are addressing, and we

[1]http://www.xen.org/
[2]http://wiki.openvz.org/
[3]http://www.vmware.com/products/vmotion/

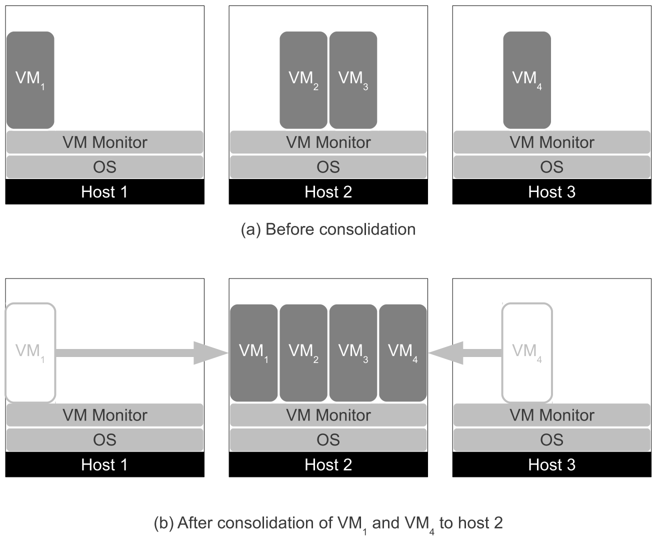

Figure 1. Effect of VM consolidation

describe the V-MAN algorithm. In Section IV we assess the performance of V-MAN by means of numerical experiments. Finally, we report conclusions and outline future research directions in Section V.

II. RELATED WORK

The problem of migrating running processes from one processor to another has been initially considered for balancing workload in distributed multiprocessor systems [4]. Recent advances in virtualization technologies allow entire running VMs to be transferred across different physical hosts [5]. This opportunity is being investigated for different purposes, mostly related with various Quality of Service aspects.

Stage and Setzer [6] describe a network-aware migration scheduler which takes into consideration the workload type of each VM. The migration takes into explicit consideration the network topology and the bandwidth requirements to move VM images within a given deadline. Wood *et al.* [7] describe Sandpiper, a system which automatically identifies performance bottlenecks, identifies a new VM allocation which removes them and finally initiate the required migrations to instantiate the new allocation. Sandpiper is OS and application independent, relying on monitoring disk and network usage inside the Xen VM monitor.

Some recent works considered distributed (hierarchical) approaches for energy management in large datacenters. Bennani and Menasce [8] present a hierarchical approach addressing the problem of dynamically redeploying servers in a continuously varying workload scenario. In this case, servers are grouped according to an application environmental logic, and a so-called *local controller* that is in charge of managing a set of servers. Das *et al.* [9] present a multi-agent system approach to the problem of green performance in data center. As for aforementioned papers, the framework is based on a hierarchy, according to which a resource arbiter assigns resources to the application managers, which in turn become in charge of managing physical servers. Srikantaiah *et al.* [10] study the impact of consolidation of multiple workloads with different resource usage on performance, energy usage, and resource utilization. This is not achieved by migrating applications, but rather by consolidating the workload so that each server receives a "balanced mix" of requests. Finally, Barbagallo *et al.* [11] describe a bio-inspired algorithm based on the scout-worker migration method, in which some entities (the *scouts*) are allowed to move from one physical node to another in order to cooperatively identify a suitable destination for VMs (the *workers*) which are migrated.

V-MAN uses a fully decentralized approach with no shared data structures or central controllers. Also, V-MAN does not require any instrumentation of either VMs or hosted applications, V-MAN does not rely on a small subset of special entities (e.g., the *scouts* of [11]): instead, all servers cooperate to identify a new VM allocation, and this ensures that V-MAN is capable of scaling with the size of the datacenter.

III. SERVER CONSOLIDATION THROUGH GOSSIPING

Problem Formulation: We consider a set of N servers, identified as $1, \dots, N$. Each server has a unique ID (IP address) which can be used to address the node directly. We assume that the Cloud system includes an appropriate communication layer such that any pair of servers can exchange messages. However, we do not require that each server know the ID of all other servers. We allow servers to join and leave the Cloud at any time; this is important because large-scale Cloud services are prone to failures which must be handled gracefully.

Each server can host at most C VMs, although the more general case in which each host has a different capacity can be trivially handled. Cloud users can request allocation of new VMs at any time, provided that the total system capacity CN is not exceeded; similarly, VM instances can be terminated at any time.

A VM allocation $\mathbf{H}$ is an array of N non negative integers $\mathbf{H} = (H_1, \dots, H_N)$ where H_i is the number of VMs running on server i, $H_i \in \{0, \dots, C\}$. We let $M = \sum_{i=1}^{N} H_i$ denote the total number of VMs in allocation $\mathbf{H}$. Since the maximum capacity of the Cloud is CN, it must be $M \leq CN$. We assume that the system can force the migration of a VM running on server i to any other server $j \neq i$, provided that the destination is not full (i.e., before the migration we must have $H_j < C$). V-MAN only cares about the number of VMs running on each server, so there is no need to complicate the notation by also specifying which VMs are running on each node. Also, migration costs are not considered by V-MAN.

Table I
SYMBOLS USED IN THIS PAPER

N	Number of servers
C	Maximum number of VMs which can run on a server
M	Current number of running VM instances
H_i	Number of VMs running on server i, $0 \leq H_i \leq C$
$F_k(\mathbf{H})$	Fraction of servers hosting exactly k VMs
$F_{0,\text{opt}}(\mathbf{H})$	Optimal fraction of empty servers

Given an allocation $\mathbf{H} = (H_1, \dots, H_N)$, we denote with $F_k(\mathbf{H})$ the fraction of hosts containing exactly k VMs: $F_k(\mathbf{H}) = |\{i \ : \ H_i = k\}|/N$ for all $k = 0, \dots, C$. Given a set of N servers with capacity C and $M \leq CN$ VMs, we want to identify a valid allocation $\mathbf{H}$ which satisfies the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & F_0(\mathbf{H}) & (1) \\ \text{subject to} \quad & \sum_{i=1}^{N} H_i = M \\ & H_i \in \{0, \dots, C\} \quad i = 1, \dots, N \end{aligned}$$

Problem (1) requires that the M VMs are allocated so that the number of empty hosts is maximized. This allows unused processors to be switched to non-operating Advanced Configuration and Power Interface (ACPI) states, producing significant power savings. Note that the value of M is not globally known; furthermore, the value of M can be different at different times, as VM instances are started or terminated.

Table I summarizes the symbols used in this paper.

If we assume global system knowledge, the optimization problem (1) has a very simple greedy solution algorithm which consists of allocating C VMs on $\lfloor M/C \rfloor$ servers, so that the maximum fraction of empty servers is $F_{0,\text{opt}}(\mathbf{H}_t) = 1 - \lceil \sum_{i=1}^{N} H_i/C \rceil / N$.

Unfortunately, the solution above can be difficult to implement in practice: as users request the instantiation or termination of VMs, an initial optimal allocation may become no longer optimal after some time.

Some desirable properties of a decentralized protocol for solving the optimization problem (1) are the following [12]: (*i*) Self-organization: the protocol must be able to operate properly even when nodes (servers) leave or join the system, without any manual intervention; (*ii*) Effectiveness: the protocol should produce a good solution as fast as possible; (*iii*) Scalability: the protocol must be efficient even when applied to very large Cloud systems; (*iv*) Robustness: the protocol must tolerate massive failures, which could actually occur in a datacenter (e.g., whole racks losing power, which would result in correlated failures of multiple servers).

Solving the optimization problem through gossip: We now describe V-MAN, a gossip-based algorithm for VM consolidation in Clouds. V-MAN is an iterative algorithm which, starting from an arbitrary initial VM allocation $\mathbf{H}_1$, produces a sequence of allocations $\mathbf{H}_2, \mathbf{H}_3, \dots, \mathbf{H}_t, \dots$ such that $F_0(\mathbf{H}_t) \leq F_0(\mathbf{H}_{t+1})$. Empirical evidence will be given in Section IV that the sequence of allocations $\{\mathbf{H}_t\}_{t=1,\dots}$ converges quickly towards a solution of the optimization problem (1).

V-MAN does not require any special policy for allocating new VMs to servers; furthermore, no special action must be taken when a VM terminates and is deallocated. V-MAN can be executed as a periodic task which "compacts" all running VMs on fewer hosts. V-MAN allows servers to be added or removed while it is running; furthermore, VM instances can be created or shut down while V-MAN operates. Thus, V-MAN can be executed in background without interfering with the normal operation of the Cloud system.

V-MAN maintains a dynamic overlay over the N Cloud servers: each server (node of the overlay) only knows a subset of K hosts which are included in its *local view*. Messages are exchanged only with nodes in the local view. Maintenance of the overlay is crucial for the correct operation of V-MAN: the overlay must be built and maintained with no global knowledge. In particular, each node must not be required to know the identity of all other nodes. Also, the overlay must be maintained even when nodes join and leave the Cloud. The last requirement is of particular importance, because Cloud systems are hosted in large datacenters and the probability of some of the servers failing is high.

The overlay network is built and maintained using a *peer sampling service* implemented as a modified NEWSCAST protocol [13]. The peer sampling service provides each node with peers to exchange information with. The peer sampling service is implemented as follows: each node periodically sends its local view to the K neighbors, and builds a new local view by merging the old one with those received by neighbors. See [13] for more details.

Once the overlay is built, V-MAN works as follows. Let $\mathbf{H} = (H_1, \dots, H_N)$ be the current allocation. Each server i only knows the number H_i of VMs which it is currently hosting. Apart from the peer sampling service, each server executes two threads whose pseudocode is shown in Figure 2. The idea is the following: node i sends the value H_i to each neighbor j in its view. At each interaction, the server with the higher number of running VMs receives the VMs running on the other peer, until the receiving side reaches its maximum capacity C. Note that in this phase no VM is actually migrated: only when a new allocation has been determined with enough accuracy, actual VM migration can take place.

ACTIVETHREAD is the active thread, which is executed every Δ time units. The thread iterates over each neighbor j, to which it sends the current number of VMs running on i, H_i; it then receives an updated value H_i' (which is possibly

```
1: $i \leftarrow$ GetProcID()

2: procedure ActiveThread
3:     loop
4:         Wait $\Delta$
5:         for all $j \in$ GetNeighbors($i$) do
6:             Send $\langle H_i \rangle$ to $j$
7:             Receive $\langle H_i' \rangle$ from $j$
8:             $H_i \leftarrow H_i'$
9:         end for
10:    end loop
11: end procedure

12: procedure PassiveThread
13:    loop
14:        Wait for message $\langle H_j \rangle$ from $j$
15:        if ($H_i > H_j$) then                    ▷ Pull from node $j$
16:            $D \leftarrow \min(H_j, C - H_i)$
17:            Send $\langle H_j - D \rangle$ to $j$
18:            $H_i \leftarrow H_i + D$
19:        else                                     ▷ Push to node $j$
20:            $D \leftarrow \min(H_i, C - H_j)$
21:            Send $\langle H_j + D \rangle$ to $j$
22:            $H_i \leftarrow H_i - D$
23:        end if
24:    end loop
25: end procedure
```

Figure 2. Active and passive threads executed by peer i

different from H_i), and updates H_i accordingly. Updates to the local value of H_i must be done atomically. Also, each server must keep track of the initial location of each VM it receives, so that at the end it cal pull the assigned VMs from their original location.

PassiveThread listens for messages coming from the other peers. Upon receiving H_j from peer j, server i decides whether some VMs should be *pushed* to j (line 19), or *pulled* from node (line 15). VMs are always transferred from the least loaded peer to the most loaded one; the number of VMs to transfer is limited by the residual capacity of the receiving node. We do not address the important issues of deciding *which* specific VMs to migrate; probably it makes sense to transfer those with smaller memory footprint, so that the transferred image is smaller.

IV. Experimental Evaluation

We implemented V-Man using Peersim [14], an open source Java simulator of peer-to-peer systems. The simulation is executed as a sequence of steps (called *cycles*). At each step all nodes execute both V-Man and the peer sampling service which is needed to maintain the unstructured overlay topology. The implementation of the peer sampling service is already provided by the `example.newscast` Peersim package.

Each node maintains a local view of $K = 20$ elements. This means that each node has 20 neighbors, which are different at each cycle as Newscast is executed. All tests have been performed on a network with $N = 10000$ servers with capacity $C = 8$. The initial configuration $\mathbf{H}_1$ is created by putting on each server a random number of VMs uniformly distributed in $[0, C]$. The simulation length is set to 20 steps. For each step $t = 1, 2, \ldots, 20$ we computed $F_k(\mathbf{H}_t)$, $k = 0, \ldots, C$, and produced area plots showing the fraction of empty hosts $F_0(\mathbf{H}_t)$ and the fraction of fully loaded hosts $F_C(\mathbf{H}_t)$. All values have been obtained by averaging the results of 10 independent simulation runs with the same parameters. The fraction of empty hosts at step t obtained by V-Man, $F_0(\mathbf{H}_t)$, has been compared with the optimal fraction $F_{0,\mathrm{opt}}(\mathbf{H}_t)$ computed as the solution of the optimization problem (1).

Experiment #1: Static system: In the first test we investigate the convergence speed of V-Man. To do so, we consider a static system, where the number of VMs is constant, and no servers join or leave the system. At time $t = 1$ the system is initialized by putting a random number of VMs on each server, as already described above. In Figure 3 we show an area plot of the fraction of empty servers ($F_0(\mathbf{H}_t)$, white area at the bottom) and full servers ($F_8(\mathbf{H}_t)$, dark area at the top); the height of the light gray area on each plot denotes the fraction of hosts which are neither full nor empty, and thus can be consolidated. The thick horizontal line shows the value of the optimal number of empty servers $F_{0,\mathrm{opt}}(\mathbf{H}_t)$. We observe that V-Man converges very quickly: after the first iteration, the fraction of empty hosts produced by the V-Man allocation is only slightly lower than the optimal value. A more detailed analysis on the convergence speed is done in the next experiments.

Experiment #2: Impact of the local view size K: The size K of the local view plays an important role, which is analyzed in this experiment. Again, we consider a static system with $N = 10000$ and $C = 8$. We consider different values of K, namely $K = 5, 10, 20$. No servers leave or join the Cloud, and no VM instance is created or destroyed. The optimality of an allocation $\mathbf{H}_t$ is computed as $|F_{0,\mathrm{opt}}(\mathbf{H}_t - F_0(\mathrm{H}_t)|$. Lower values denote that the fraction of empty hosts in $\mathbf{H}_t$ is almost optimal. Results are shown in Figure 4; we observe that larger values of K result in faster convergence of V-Man towards the optimal solution. The reason is that, for large values of K, each server has more neighbors to exchange VMs with.

Experiment #3: Static system with variable number of VMs: We now consider a more realistic scenario in which the number of servers is still fixed at $N = 10000$, but

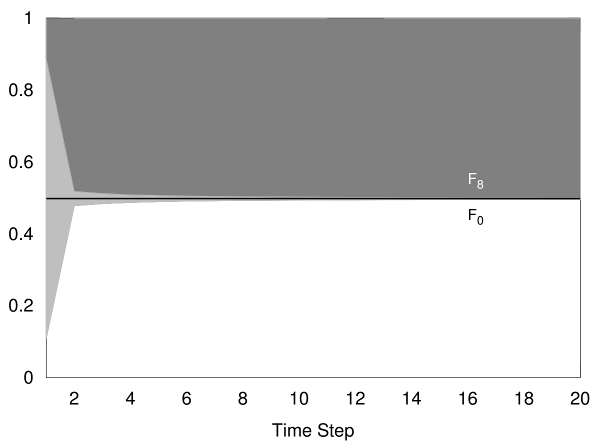

Figure 3. Static system. $N = 10000$, $C = 8$, random initial configuration. Values are averages of 10 independent simulation runs.

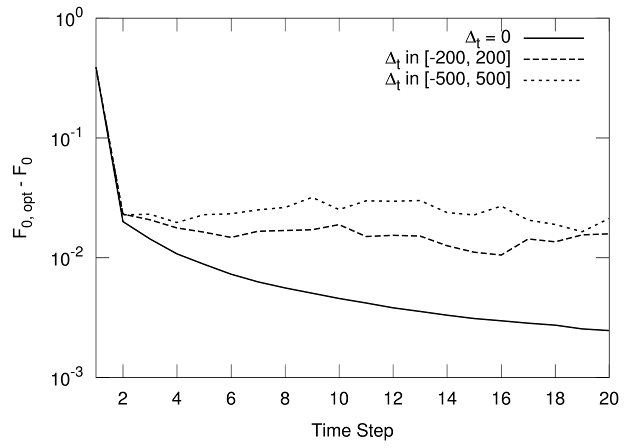

Figure 5. Accuracy of V-MAN with variable number of VMs for each step. Results are averages of 10 independent simulation runs).

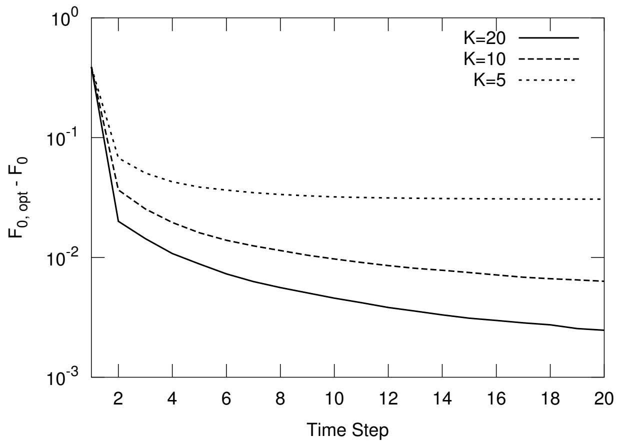

Figure 4. V-MAN accuracy for different values of the local views size K (static system $N = 10000$, $C = 8$, random initial configuration, all values are averages of 10 independent simulation runs).

at each step some VMs are added or removed from the system (*VM churn*). With this we simulate the fact that in real Clouds users can request activation or termination of their VM instances at any time. Specifically, we performed a simulation run in which at each step t we added or removed a number Δ_t of VMs. We consider three situations: $\Delta_t = 0$ (static system), Δ_t uniformly distributed in $[-200, 200]$ and Δ_t uniformly distributed in $[-500, 500]$. The results are shown in Figure 5. In all cases V-MAN is effective in increasing the number of empty hosts; of course, as the variability of the number of VMs increase, V-MAN produces less accurate allocations. However, results are still acceptable, especially if we consider that V-MAN operates without the need to interrupt the normal Cloud operations, and thus does not interfere with the normal system behavior.

The effect of V-MAN can be better appreciated if we stop the protocol and let the churn of VMs drive the Cloud

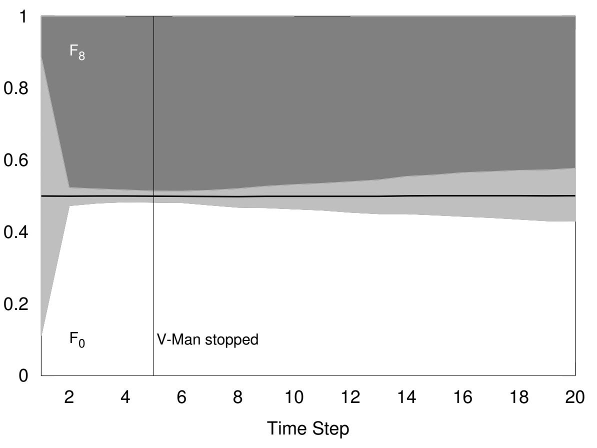

Figure 6. Effect of stopping V-MAN at step $t = 5$.

system towards higher entropy (i.e. the VM allocation gets increasingly irregular). In Figure 6 we executed a simulation with $N = 10000$, $C = 8$ and the VM churn set to $\Delta_t \in [-500, 500]$. V-MAN is stopped at time $t = 5$, after which we clearly observe that the VM allocation deviates from the optimum.

Experiment #4: Dynamic system: In this test we consider a fully dynamic scenario, in which at each step we add and remove VMs, but also add and remove servers from the Cloud. We simulate a major Cloud outage at time $t = 5$, where 1000 random servers are removed from the system; all VMs running on them are lost. Then, 2000 new (empty) servers are added at time $t = 10$. At each step t we add or remove Δ_t VMs from the system, where Δ_t is uniformly distributed in $[-500, 500]$. The result shown in Figure 7 show that V-MAN is resilient to node failures, which is a common features of many gossip-based protocols [15]. Note that the average fraction of empty servers is always a good

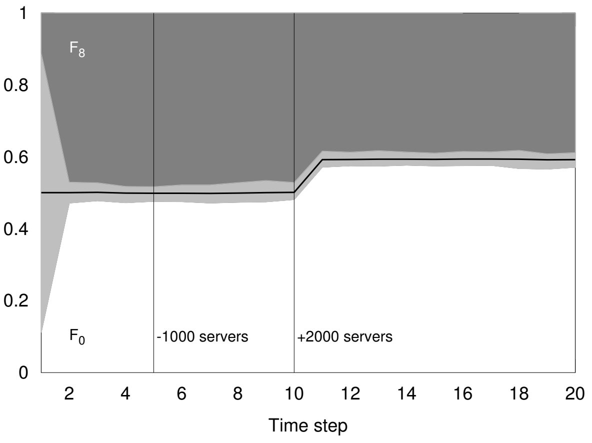

Figure 7. Fully dynamic system. $N = 10000$, $C = 8$. VM churn Δ_t is uniformly distributed in $[-500, 500]$. 1000 random servers are removed at time $t = 5$; 2000 empty servers are added at time $t = 10$.

approximation of the optimal value.

V. Conclusions and Future Works

In this paper we proposed V-Man, a fully decentralized, gossip-based algorithm for consolidating Virtual Machines in a Cloud. V-Man implements a distributed algorithm for identifying a new VM allocation which maximizes the number of empty hosts (i.e. hosts running no VM). Each server exchanges messages with a limited number of peers; these messages are used (*i*) to maintain an unstructured overlay network, and (*ii*) to exchange VMs from lightly loaded nodes to heavily loaded ones.

We implemented V-Man using the Peersim simulator. Results are encouraging, showing that V-Man converges very quickly–less than five rounds of message exchanges are sufficient to produce an almost optimal allocation. Furthermore, V-Man is resilient to server failures, and can tolerate high variability in the number of running VMs.

We plan to extend this work by taking into consideration a more detailed cost model when deciding which VM to migrate. In this paper we have assumed that all VMs are identical, but in practice this could not be the case. For example, if two VMs have vastly different memory footprints, it is cheaper to move the VM with smaller memory image, as the migration operation can be completed in a shorter time interval. We plan to improve V-Man by taking into consideration a more detailed migration cost model. We will also implement V-Man on a real testbed so that a more accurate and realistic performance assessment can be done.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010.

[2] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[3] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[4] D. S. Milojíčić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Comput. Surv.*, vol. 32, pp. 241–299, September 2000.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. NSDI'05*. USENIX Association, 2005, pp. 273–286.

[6] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proc. CLOUD'09*. IEEE Computer Society, 2009, pp. 9–14.

[7] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. NSDI'07*. USENIX Association, Apr. 11–13 2007, pp. 229–242.

[8] M. Bennani and D. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. ICAC 2005*, June 2005, pp. 229–240.

[9] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, and H. Chan, "Autonomic multi-agent management of power and performance in data centers," in *PRoc. AAMAS'08*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 107–114.

[10] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. HotPower'08*. USENIX Association, 2008, pp. 10–10.

[11] D. Barbagallo, E. Di Nitto, D. J. Dubois, and R. Mirandola, "A bio-inspired algorithm for energy optimization in a self-organizing data center," in *Proc. SOAR'09*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 127–151.

[12] M. Jelasity, W. Kowalczyk, and M. van Steen, "Newscast computing," Vrije Universiteit Amsterdam, Technical Report IR-CS-006.03, 2003. [Online]. Available: http://hdl.handle.net/1871/11668

[13] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, August 2007.

[14] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator." [Online]. Available: http://peersim.sf.net

[15] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, pp. 219–252, August 2005.