2010 IEEE 3rd International Conference on Cloud Computing

Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis

Yağız Onat Yazır[α], Chris Matthews[α], Roozbeh Farahbod[β]
Stephen Neville[γ], Adel Guitouni[β], Sudhakar Ganti[α] and Yvonne Coady[α]

[α]
Dept. of Computer Science
University of Victoria
Victoria, BC, Canada

[β]
C2 Decision Support System Section
Defense R&D Canada Valcartier
&
Faculty of Business
University of Victoria
Victoria, BC, Canada

[γ]
Dept. of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada

***Abstract*—In computing clouds, it is desirable to avoid wasting resources as a result of under-utilization and to avoid lengthy response times as a result of over-utilization. In this paper, we propose a new approach for dynamic autonomous resource management in computing clouds. The main contribution of this work is two-fold. First, we adopt a distributed architecture where resource management is decomposed into independent tasks, each of which is performed by Autonomous Node Agents that are tightly coupled with the physical machines in a data center. Second, the Autonomous Node Agents carry out configurations in parallel through Multiple Criteria Decision Analysis using the PROMETHEE method. Simulation results show that the proposed approach is promising in terms of scalability, feasibility and flexibility.**

I. Introduction

For clients, cloud computing provides an abstraction of the underlying platform which saves them the costs of designing, building and maintaining a data center to host their Application Environments (AE). By leveraging system virtualization in a data center, multiple Virtual Machines (VM), which are the components of AEs, can be consolidated on one Physical Machine (PM). A VM is loosely coupled with the PM it runs on; as a result, not only can a VM be started on any PM, but also, it can be migrated to other PMs.

Currently, migrations can be performed in seconds to minutes depending on the size, activity and bandwidth of the VM and PMs. Migration makes it possible to dynamically adjust data center utilization and tune the resources allocated to AEs. Furthermore, these adjustments can be automated through formally defined strategies in order to continuously manage the resources in a data center with less human intervention. We refer to this task as the process of *dynamic and autonomous resource management*.

In former research, this process is generally performed in a centralized manner, focusing on the use of utility functions to declare the preferences of AEs over a range of resource levels. Each AE then communicates its preferences to a global arbiter which computes a near-optimal distribution of resources among AEs [1]. In this paper, we propose a new approach to the same problem in the context of computing clouds. Our contribution is two-fold. First, we adopt a distributed architecture where resource management is decomposed into independent tasks and each task is performed by Autonomous Node Agents (NA) that are tightly coupled with the PMs in a data center. Second, NAs carry out configurations through Multiple Criteria Decision Analysis (MCDA) using the PROMETHEE method [2].

The rest of this paper is organized as follows. In Section II, we provide a description of the problem focusing on the primary goals and constraints. Section III outlines the former related research. Section IV provides details of the design and methodology of our approach. In Section V, we explain the experimental setup and provide a detailed assessment of our approach. Finally, Section VI summarizes our conclusions and outlines our future directions.

II. Problem Description

In a data center, the primary goal of a dynamic autonomous resource management process is to avoid wasting resources as a result of under-utilization. Such a process should also aim to avoid high response times as a result of over-utilization which may result in violation of the service level agreements (SLA) between the clients and the provider. Furthermore, it needs to be carried out continuously due to the time variant nature of the workloads of AEs.

978-0-7695-4130-3/10 $26.00 © 2010 IEEE
DOI 10.1109/CLOUD.2010.66

At a high level, this process can be decomposed into two distinct, and inter-dependent phases. The first phase consists of defining a mapping between the AEs' service level requirements and resource level requirements. Resource level requirements are generally derived from SLAs based on certain parameters such as response time, throughput, etc; whereas, resource level requirements are often outlined as CPU usage, memory, bandwidth, etc. As the workload of an AE changes in time, this mapping is used to determine the amount of resources that should be assigned to each component—encapsulated in VMs—in order to satisfy the terms outlined in the SLA. This phase also requires performance modeling and demand forecasting for AEs. The accuracy of the output from this first phase has direct effects on the accuracy of the configuration produced in the second phase.

The second phase involves the computation and application of a new configuration by distributing the resources in a data center among the VMs that represent AEs. The configuration is computed based on the output of the mappings produced in the first phase. Maintaining this configuration is a resource allocation problem and is generally defined as a Knapsack Problem [1] or as a specific variant of it, namely Vector Bin Packing Problem [3], both of which are known to be NP-Hard. This phase consists of selecting a suitable configuration from a solution space with respect to a set of criteria. The criteria are used to define the quality of the solution in terms of certain requirements such as satisfying SLAs, overall data center utilization, and the overhead of applying an alternative configuration. The methods to be adopted in this phase need to be flexible so that the providers can easily redefine the configuration goals by adding new criteria or tuning the importance assigned to them.

In the second phase, certain constraints and limitations need to be taken into consideration. Two of these are the time-spent during the selection of a new configuration, and the feasibility of it. Due to the time variance in workloads, a new configuration must be computed in a reasonable amount of time so that it is not stale under the current conditions. The selected configuration must also be feasible in terms of the number migrations necessary. The number of migrations that can be performed in a data center still has limits with the current technologies. In summary, the second phase must provide fast-to-compute, feasible, and reasonably good configurations that do not over-utilize or under-utilize the data center.

In this paper, we focus on the second phase of the process and leave the problem of defining accurate resource mappings, performance modeling or demand forecasting for AEs outside the scope of this work. We believe that these problems can be decoupled, and solutions that address the first phase should be investigated in a separate work.

In the next section, we briefly investigate the previous work in the field and detail how our approach differ from them.

III. Related Work

Since first proposed in the seminal work of Walsh et al. [1], a general two-layer architecture that uses utility functions has been commonly adopted in the context of dynamic and autonomous resource management. This architecture consists of *local agents*—also called local decision modules, or application agents, etc.— that are tightly coupled with each AE, and a *global arbiter*—also called a global decision module—that computes and applies configurations on behalf of the entire data center in a centralized manner.

The main responsibility of the local agents is to calculate utilities. Given the current or forecasted workloads and the range of resources, the utility function calculates a utility between 0 and 1. Once the local agents calculate the utilities for each AE, the results are communicated to the global arbiter.

The global arbiter is responsible for computing a near-optimal configuration of resources given the utilities declared by the local agents. The new configurations are applied by assigning new resources to the AEs in the data center. The main idea is to assign enough resources to each AE without violating SLAs while not exceeding the total resources in the data center. The global arbiter computes new configurations either at the end of fixed control intervals or in an event-triggered manner where events are considered to be current or anticipated SLA violations.

Studies outlined in Bennani et al. [4], Chess et al. [5], Tesauro [6], Tesauro et al. [7], [8], and Das et al. [9] adopted this approach for non-virtualized data centers. They have also focused on performance modeling of AEs, leveraging forecasting methods based on different analytical models. While Bennani and Menasce [4] mainly focused on a queuing theoretic approach to the performance modeling problem, Tesauro [6],Tesauro et al. [7], [8] considered a pure decompositional reinforcement learning approach and a hybridization with the queuing theoretic approach. Furthermore, in Chess et al. and Das et al. [5], [9], this same architecture and utility model is used to build a commercialized computing system, called Unity.

Later research has focused on virtualized data centers where the components of AEs are encapsulated in VMs. Starting with the work of Almeida et al. [10], data center utilization has been explicitly considered as a major criterion in dynamic and autonomous resource management. In addition, this work outlined data center utilization as a major factor in both short-term and long-term resource planning. Other research outlined in Khanna et al. [11], Bobroff et al. [12], Wood et al. [13], Wang et al. [14], Kochut [15], Hermenier et al. [3], and Van and Tran [16] have also adopted a centralized configuration as outlined in [1]. In some studies the cost of migrations—in terms of overhead—during configurations was taken into account [13], [11], [14], [3], [16]. Hermenier et al. [3] focused on reducing the number of migrations through constraint solving. The general idea is to find a set of possible configurations and electing the most suitable one that maximizes global utility and minimizes the number of migrations. This same method is also adopted in the follow-up work of Van and Tran [16].

Although these methods are very efficient in relatively small data centers, we believe that they can potentially suffer from scalability, feasibility and flexibility issues.

The scalability issues may arise mainly due to centralized control and aim for near-optimal configurations. As the size of a data center increase the computation of a near-optimal configuration becomes more complex and time consuming. The time taken during such computations may result in stale configurations since the environment is time variant. The feasibility issues are also related to the aim for near-optimal configurations. Such configurations often require a total re-assignment of resources in the data center. This may result in a very high number of migrations rendering the application of the configuration impossible in a large data center. In addition, usage of utility functions may raise flexibility issues. In a large data center with many AEs, defining a common utility function to effectively represent all AEs' preferences over resources can be very complex. Moreover, the addition of new criteria requires a redefinition of the utility function to be used.

In this paper, we propose a new approach to the resource allocation problem through an architecture that distributes the responsibility of configuration among NAs. Each NAs is a unit that is tightly coupled with a single PM in the data center, which configures its resources through MCDA only when imposed by the local conditions. The distribution of responsibility makes our approach inherently scalable. This limits the solution space to the local views NAs, which results in fast and up-to-date solutions regardless of the size of the data center. Since our approach does not aim for a global re-configuration of the resources in the entire data center, the number of migrations per configuration is substantially less than the global solutions making our approach more feasible given current technology. Finally, NAs use the PROMETHEE method which gives us the flexibility particularly in terms of adding new criteria to the assessment of configurations along with the ability to easily tune the weights of criteria.

In the next section, we provide a detailed overview of this new approach in terms of the design of the overall architecture and how PROMETHEE method is used by NAs.

IV. System Architecture

The system is designed as a distributed network of NAs, each capable of accommodating VMs and, when necessary, delegating VMs to other PMs and handing the management over to the corresponding NAs. We assume that the network maintains global awareness of the resource availability of PMs and their task assignments through an *oracle*. Concretely, the oracle can be implemented as either a distributed or a centralized monitoring system [17].

The system model is described in abstract functional and operational terms based on the Abstract State Machine (ASM) paradigm [18] and the CoreASM open source tool environment [19] for modeling dynamic properties of distributed systems. ASMs are known for their versatility in computational and mathematical modeling of complex distributed systems with an orientation towards practical applications [20]. The description of the model in ASM reveals the underlying design concepts and provides a concise yet precise blueprint for reasoning about the key system properties. Due to space limitations, in this paper we present only certain parts of the abstract model.

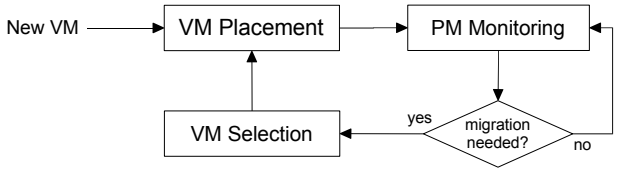

Fig. 1. NA Program

Each NA observes the local resource usages of the PM that it is coupled with. If the local observations reveal an *anomaly* that the resources are over-utilized or under-utilized, it is immediately followed by the configuration phase which is carried out in three steps:

1) Choosing a VM to migrate from the list of VMs that run on the problematic PM.
2) Choosing a PM to migrate the chosen VM to.
3) Migrating the chosen VM to the chosen PM.

The process of dynamically allocating VMs to PMs and maintaining resource distribution among VMs to meet their resource requirements, is modeled as a distributed process carried out by individual NAs in the system. Conceptually, every NA continuously performs a cycle of three activities (see Figure 1): (1) *VMPlacement* where a suitable PM capable of running the given VM is found and the VM is assigned to that PM, (2) *Monitoring* where the NA monitors total resources used by the hosted VMs; (3) *VMSelection* where, if local accommodation is not possible, a VM is selected to be migrated to another PM and the process loops back into placement.

Every NA is modeled as a Distributed ASM (DASM) agent that continuously runs its main program. The following ASM program abstractly captures the behavior of the NAs[1]. In parallel to the three main activities, every NA also continuously responds to communication messages it receives from other NAs.

PM Program

```
PMProgram ≡
  VMPlacement
  Monitoring
  if migrationNeeded then VMSelection
  Communication
```

Since the system is distributed, new VMs can be assigned to any PM in the system. Once entered into the data center, a VM goes through a lifecycle starting with being *unassigned* and ending with being *terminated* which is when its execution is completed (see Figure 2). For any given VM at a given time, one PM is responsible to move the task through its life cycle. We use $\mathcal{N}$ and $\mathcal{V}$ to denote the set of all PMs and VMs in the data center respectively. The current status of VMs are captured by the function $vmStatus : \mathcal{V} \mapsto \text{VMSTATUS}$. Every

[1]Note that according to ASM semantics, the composition of the four activities in this program denotes a parallel execution of these activities

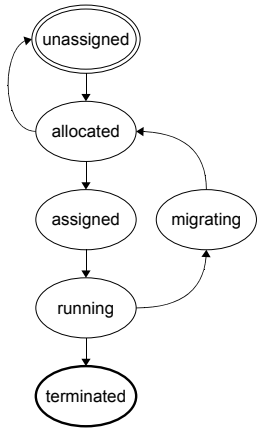

Fig. 2. VM Lifecycle

PM has a pool of VMs that it is responsible for, captured by the function *vmPool* : $\mathcal{N} \mapsto \mathcal{P}(\mathcal{V})$. The set of VMs that are running on a PM is captured by the function *pmAssignment* : $\mathcal{N} \mapsto \mathcal{P}(\mathcal{V})$. Then, we have:

$$\forall p \in \mathcal{N}\ \textit{pmAssignment}(p) \subset \textit{vmPool}(p)$$

The rest of this section focuses on the main activities of the NAs.

A. VM Placement

The VM Placement activity consists of two tasks: (1) Finding suitable PMs and allocating resources for *unassigned* VMs, and (2) Assigning *allocated* VMs to their corresponding PMs. In the Placement activity, an NA looks for the VMs that are either *unassigned* or *allocated*. For these VMs, NA is acting like a broker;

Placement

Placement ≡
ResourceAllocation
VMAssignment

In ResourceAllocation, an NA picks an unassigned VM from its VM pool and tries to find a suitable PM with available resources that can run the VM. At this level, we capture the computation of finding such a PM by the abstract function *placementPM* : $\mathcal{V} \mapsto \mathcal{N}$ that suggests a PM that can run the given VM.

In order to do this, *placementPM* first queries the oracle to retrieve a set of alternative PMs, that have enough resources to host the VM. The oracle returns the set of all of the PMs in the data center that can host a given VM v without exceeding the resource threshold T_r defined per resource dimension r such as CPU, memory and bandwidth:

$$A_{PM} = \{p \in \mathcal{N} | \forall r \in R\ \alpha_r^p + \beta_r^v \leq T_r\}$$

where R is the set of resource dimensions, α_r^p is resource usage of r on PM p, and β_r^p is the resource requirement of VM v for r.

After A_{PM} is provided by the oracle, the next task is to decide on a course of action:

$$\text{Act} = \begin{cases} \textit{noAct} & \text{if } A_{PM} = \emptyset, \\ \textit{select PM} & \text{if } A_{PM} \neq \emptyset. \end{cases}$$

If there is no PM in the data center that can host the VM v, then no action is taken. Otherwise, NA choses the most suitable PM from A_{PM} using MCDA.

Currently, each NA performs MCDA using six criteria to evaluate the quality of each alternative in A_{PM}. We define these criteria as:

g_1 = CPU Usage	g_4 = CPU Variablity
g_2 = Memory Usage	g_5 = Memory Variablity
g_3 = Bandwidth Usage	g_5 = Bandwidth Variablity

The criteria g_1, g_2 and g_3 are used to evaluate the alternatives with respect to the amount of available resources. Whereas g_4, g_5, and g_6 are used to evaluate the alternatives with respect to the recent variation in resource usages. Each g_i is assigned a weight w_i such that $\sum_{i=1}^{6} w_i = 1$.

NAs use the PROMETHEE method [2] to rank each alternative in A_{PM}. The PROMETHEE method is based on pairwise comparisons of the alternatives. That is, it considers the difference between the evaluation of two alternatives over each criterion. The preference of one alternative over another is modulated as a function of this difference. The preferences are quantified as real numbers between 0 and 1 through certain proposed preference functions. Each criterion is associated with a generalized criterion which is represented by the pair $(g_i, P_i(a,b))$, where $P_i(a,b)$ denotes the preference of alternative a over b for criterion g_i. In this paper, we use the *Usual Criterion* as a common generalized criterion for all of the criteria.

Using the generalized criteria, NAs calculate aggregated preference indices to express with what degree an alternative is preferred to another. For $a, b \in A_{PM}$, the aggregate preference indices are defined as:

$$\pi(a,b) \quad = \quad \sum_{i=1}^{6} P_i(a,b) w_i$$

Using these indices, an NA calculates how much each alternative PM outranks the other alternatives and how much it is outranked by the others. For this, positive and negative outranking flows are calculated as:

$$\begin{aligned} \phi^+(a) &= \frac{1}{n-1} \sum_{x \in A_{PM}} \pi(a,x) \\ \phi^-(a) &= \frac{1}{n-1} \sum_{x \in A_{PM}} \pi(x,a) \end{aligned}$$

Finally, the net outranking flow is used to create a complete ranking of the alternatives:

$$\phi(a) = \phi^+(a) - \phi^-(a)$$

The alternatives are ranked based on their net outranking flows, the alternative with highest outranking flow being the best alternative.

Once the ranking is complete, NA sends a resource lock request to the NA of the highest ranked PM to allocate its

resources before actually delegating the VM to it. It also changes the status of the VM to *allocated*. The following ASM rule captures this behavior:

Resource Allocation

ResourceAllocation $\equiv$
  choose $vm \in vmPool(self)$ **with** $vmStatus(vm) = unassigned$ **do**
    let $pm = placementPM(vm)$ **in**
      if $pm \neq undef$ **then**
        RequestResourceLock(pm, vm)
        $potentialPM(vm) := pm$
        $vmStatus(vm) := allocated$
        SetTimer($self, vm$)
      else
        Warning("Cannot find a suitable PM.")

When an NA receives a resource lock request for a given VM, it checks whether it still has the required resources available. If so, it puts a lock on the given resources, sets a timer and sends an acknowledgement to the requesting NA; otherwise, it sends a lock rejection. If it does not receive the VM within a certain time window, it removes the lock and releases the resources.

Upon receiving a lock acknowledgement, the NA changes the status of the VM to *assigned* and sends it to the target NA. As part of the Communication activity, NAs continuously monitor incoming network messages. If an NA receives a VM-assignment message with a VM that it has already allocated resources for, it will release the lock, assigns the resources to the VM, starts the VM and adds it to the set of its assigned VMs.

Process VM Assignment

ProcessVMAssignment(msg) $\equiv$
  let $vm = msgData(msg)$ **in**
    add vm **to** $vmPool(self)$
    if $vm \in allocatedVMs(self)$ **then**
      AssignResourcesToVM(vm)
      ReleaseLock(vm)
      add vm **to** $pmAssignments(self)$
      StartVM(vm) // sets its status to running
      remove vm **from** $allocatedVMs(self)$
    else
      $vmStatus(vm) := unassigned$

Notice that at this level the NA assumes the responsibility of the given VM and if it does not have resources allocated (locked) for that VM (which could be the case for new VMs), it simply sets the VM status to *unassigned* and keeps it in its VM pool so that it can be later processed by the PMPlacement activity.

B. Monitoring

In monitoring, an NA continuously observes the resource usages of the PM that its coupled with. The main idea behind monitoring is to capture any anomalies at a PM with respect to allocation of resources. In this context, we define anomalies as resource usages on a PM reaching above certain thresholds T_r on different resource dimensions r.

The thresholds are defined per each resource dimension r, and represent a limit on resources to keep response times reasonably low. Therefore, monitoring is performed to also capture the likeliness of SLA violations, and trigger necessary actions. As a result, monitoring ensures that the system reacts to anomalies in an event-triggered fashion rather than periodically.

In that sense, the actions that a monitor takes can be outlined as:

$$\text{Act} = \begin{cases} noAct & \text{if } \alpha_r^p \leq T_r, \forall r \in R, \\ triggerVMSelection & \text{if otherwise.} \end{cases}$$

where monitoring continues observing without triggering any reactions when the resources on the PM is below the thresholds on each dimension, or when excess of any threshold is captured it triggers the *VMSelection* and continues observing.

C. VM Selection

Once an anomaly is captured by the monitor, *VM Selection* activity starts. This is where one or more VMs are selected from the VM pool of the problematic PM to be migrated to another PM in order to resolve the anomaly. The selected VMs are used by the VM Placement activity to find a suitable target PM for each of them in the data center.

The selection of VMs are done using PROMETHEE similar to the way it is used in VM Placement. However, the alternative set is formed in a different manner. In VM Selection, an NA does not need to contact the oracle since it does the selection from its local pool of VMs. Therefore, the alternative set is computed based on the PM's local information.

First, VM Selection aims to reduce the number of migrations to resolve the anomaly by trying to include in the alternative set only certain VMs, removal of which will resolve the anomaly in one migration. Then the alternative set A_{VM}^p is defined as:

$$A_{VM}^p = \{v \in \mathcal{V}^p | \forall r \in R \ \alpha_r^p - \beta_r^v \leq T_r\}$$

where $\mathcal{V}^p$ is the set of all VMs running on PM p.

If there are no such VMs present, then A_{VM}^p is equal to $\mathcal{V}^p$. This in essence means that multiple VMs need to be removed in order to resolve the anomaly.

It is important to note here that the reason behind defining alternatives as single VMs instead of combinations of VMs on a PM—power set of $\mathcal{V}^p$—is to avoid the computational complexity that may arise due to searching a potentially large solution space for an ideal combination.

V. Experimental Setup, Results and Validation

In order to test our approach and compare it with other methods, we have built a simulation environment using the C Programming Language. The simulation runs in a stepwise manner. In each iteration, the conditions in the data center evolve with the changing resource requirements of the hosted AEs. The simulator is designed in this way to allow us to compare different methods at identical points in time with the same conditions.

In the simulation environment, VMs are implemented as independent units. These units are the components of AEs, and simulate resource requirements on each resource dimension. The simulated requirements follow a uniform, normal, exponential, Poisson or Pareto distribution. VMs declare resource requirements as random noise signals. We make a distinction between online and batch processes. The length of busy and idle periods of online VMs is simulated in order to generate a system-wide self similar behaviour that is generally observed in online data centers [21]. Further details regarding the generation of resource requirements can be found in our technical report [22].

We have implemented PMs as resource bins that the VMs are placed on. The NAs are implemented as independent units that are tightly coupled with each PM in the data center, which control the dynamic and autonomous resource management on the PMs. NAs are used by the two distributed methods we implemented: (1) Our approach as explained in Section IV, and (2) A Simple Distributed Method (SDM). SDM works by picking the first VM on a problematic PM and migrating it to the first available PM in the data center. Besides NAs, we have also implemented a global arbiter to be used by two centralized approaches. These two approaches use First Fit (FF) and First Fit Decreasing (FFD) methods to perform resource allocations.

The simulator is used to run test scenarios involving varying numbers of VMs and PMs in a data center. Due to space limitations, in this paper, we focus on only a certain number of those scenarios. In each scenario the number of PMs in the data center is determined before run-time. In each simulation run, the data center starts with no VMs and at each iteration new VMs arrive while some of the older VMs terminate and leave the data center. Once a predefined maximum number of VMs is reached, if any of the VMs terminate, they are replaced by new arrivals to keep the number of VMs approximately at the maximum.

Figure 3 shows the number of PMs used by the four methods at each iteration in a data center of 2,500 PMs. In this scenario, the maximum number of VMs is 25,000. Figure 3 shows that the FF and FFD methods use less PMs in order to host the number of VMs at each iteration. Our approach uses the highest number of PMs among the four approaches; however, Figure 4 shows that in the same scenario the number of migrations at each step in the distributed approaches are substantially less than that of FF and FFD. In fact, the number of migrations that are performed by FF and FFD at each iteration and the number of VMs in the data center at the corresponding iterations are very close. The high number of migrations relative to the number of VMs is an important feasibility issue that may generally render the centralized methods impractical in large data centers; mainly due to the global re-allocation of most of the resources in the data center.

In Figure 4, it can be observed that our approach performs the least number of migrations among the four approaches. The difference in the number of migrations performed by SDM and our approach can be observed more clearly in Figure 5. The data in Figures 3 and 5 shows that our approach performs

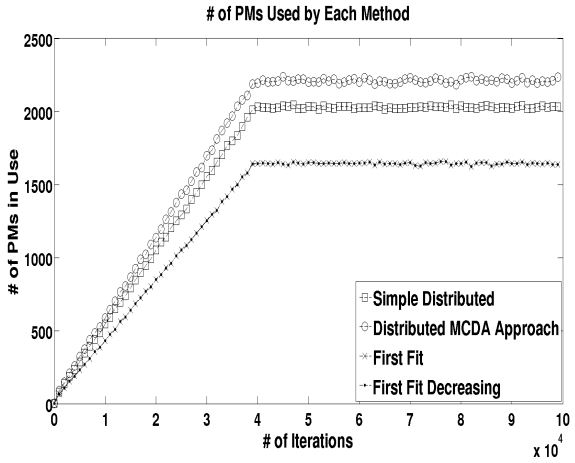

Fig. 3. The number of PMs used by the four methods at each iteration in a data center of 2500 PMs. Note that FF and FFD lines approximately overlap.

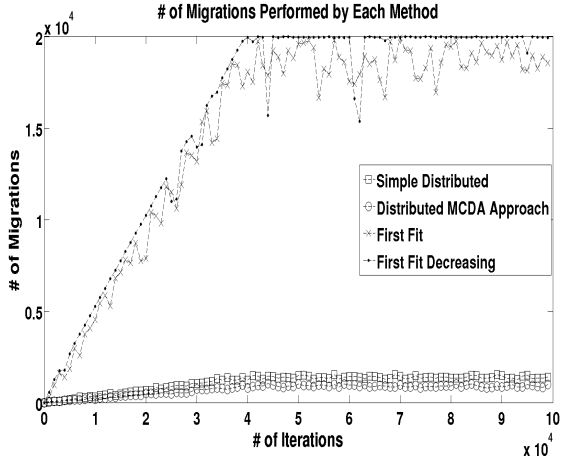

Fig. 4. The number of migrations performed by the four methods at each iteration in a data center of 2500 PMs.

approximately half as many migrations as SDM performs while using slightly more PMs. Therefore, in terms of the number of migrations to apply resource allocations among the four approaches, our approach is the most feasible.

Moreover, because utilization and response times are linked for load dependent resources, the higher utilization that is achieved by the centralized methods may evoke issues with respect to high response times. These higher response times may cause more SLA violations, particularly in the case of online AEs. This means that in a data center of PMs that are constantly on, it may be better not to define high utilization as the only goal of dynamic resource allocation. The CPU utilization in the overall data center is shown in Figures 6 and 7 for FFD and our approach respectively. Both of the methods use $T_r = 60\%$ as a threshold for CPU usage. We can infer from Figure 6 that FFD utilizes the PMs in the data center heavily while possibly producing high response times. Whereas, Figure 7 shows that our approach distributes

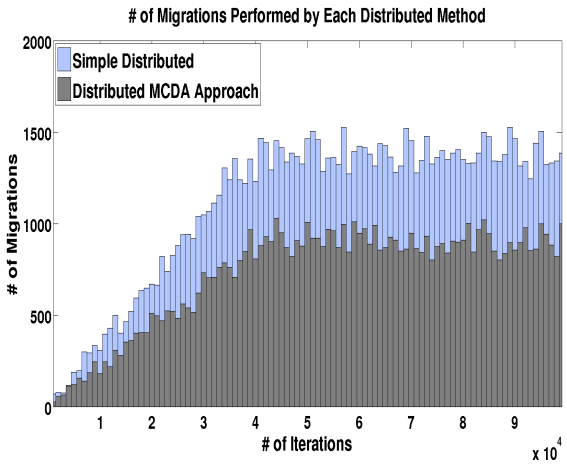

Fig. 5. A closer look at the lower half of Figure 4: the number of migrations performed by SDM and our approach at each iteration in a data center of 2500 PMs.

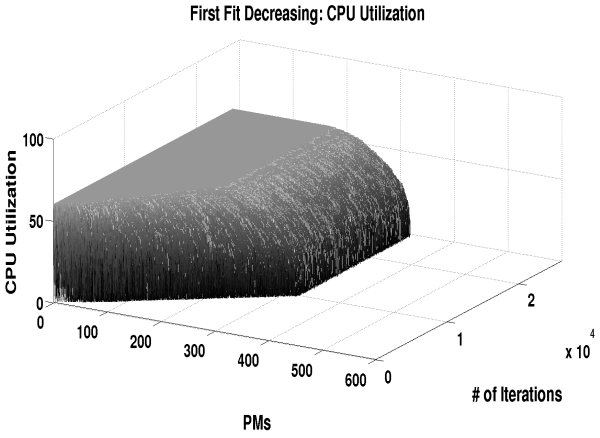

Fig. 6. The overall CPU utilization achieved by FFD throughout a simulation run on a data center of 600 PMs.

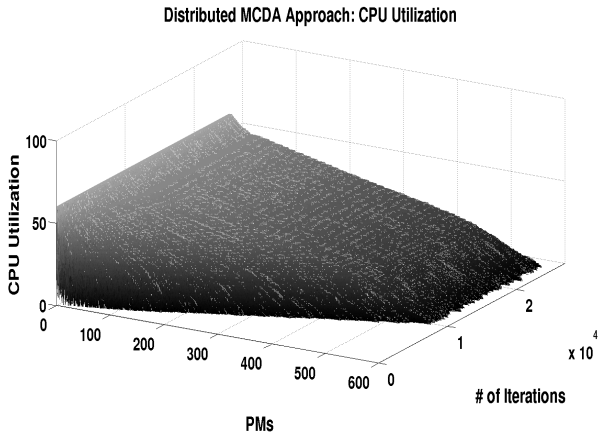

Fig. 7. The overall CPU utilization achieved by our approach throughout a simulation run on a data center of 600 PMs.

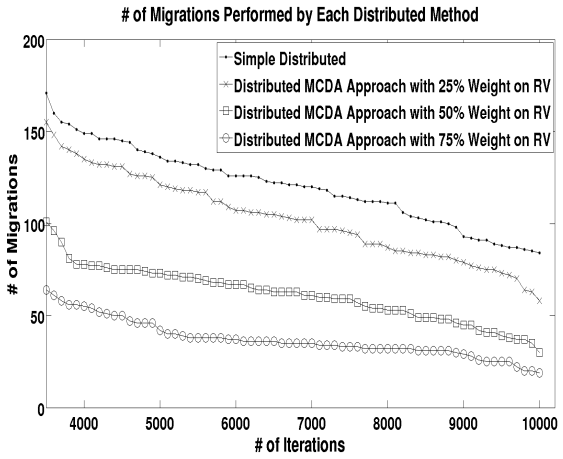

Fig. 8. A view of our approach using different weight assignments to resource variation criteria and SDM in terms of the number of migrations.

the CPU load among more PMs instead of reaching high utilization in the data center. This means that the response times will be shorter which will in turn result in fewer SLA violations with a trade-off of using more PMs. It is also important to note here that FF produce CPU utilizations similar to FFD, and SDM produce CPU utilization similar to our approach. In addition, a similar pattern is observed in bandwidth and memory utilization but is not included in this paper due to space limitations. The details can be found in our associated technical report [22].

Furthermore, our approach provides flexibility in terms of resource management by way of tuning the weights of the criteria. In the simulations explained so far, in our approach equal weights were assigned to each criteria. We have also run simulations by modifying the weights on the criteria. Our main focus is to observe the change in the number of migrations and the number of PMs used as we increase the weight on the resource variation criteria, g_4, g_5 and g_6. Figures 8 and 9 show a comparison of our approach to SDM under varying weights on resource variation criteria in a small data center of 300 PMs. Figure 8 illustrates that as the weights on resource variation criteria are increased, the number of migrations performed by our method at each iteration substantially decreases. In Figure 9, as the weights on resource variation criteria are decreased the number of PMs used by our method is decreased. This shows that our approach provides the level of flexibility that the management objectives can be fine-tuned by easily modifying weights of criteria in order to aim for fewer migrations or higher data center utilization.

VI. Conclusion and Future Work

In this paper, we introduced a new approach for dynamic autonomous resource management in computing clouds. Our approach consists of a distributed architecture of NAs

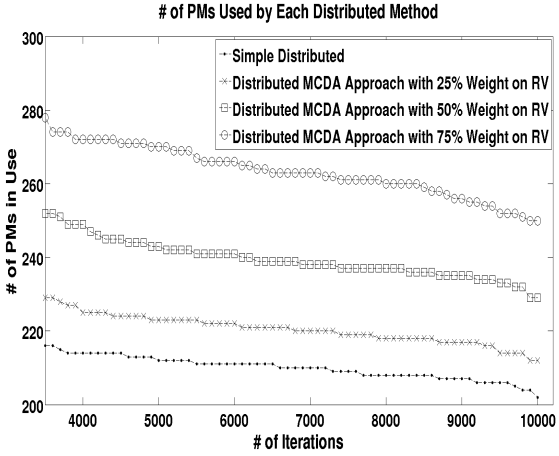

Fig. 9. A view of our approach using different weight assignments to resource variation criteria and SDM in terms of the number of PMs used.

that perform resource configurations using MCDA with the PROMETHEE method. The simulation results show that this approach is promising particularly with respect to scalability, feasibility and flexibility.

Scalability is achieved through a distributed approach that reduces the computational complexity of computing new configurations. Simulation results show that our approach is potentially more feasible in large data centers compared to centralized approaches. In essence, this feasibility is due to the significantly lower number of migrations that are performed in order to apply new configurations. Simulation results show that our method should theoretically trigger less SLA violations by smoothly distributing the overall resource utilization over slightly more PMs in the data center. The flexibility of our approach comes from its ability to easily change the weights of criteria and adding/removing criteria in order to change configuration goals.

In the next stages of this work, our goal is to include new criteria—such as VM activity—to reflect on the overhead of migrations more precisely. We are going to explore further refinements to our use of the PROMETHEE method by incorporating generalized criteria other than the Usual Criterion. In addition, we plan to compare the use of PROMETHEE to other MCDA methods. Finally, we are working on the design and implementation of new modules that will enhance the simulation environment with respect to the measurement of SLA violations.

REFERENCES

[1] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," in *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*. IEEE Computer Society, 2004, pp. 70–77.

[2] B. Mareschal, "Aide a la Decision Multicritere: Developpements Recents des Methodes PROMETHEE," *Cahiers du Centre d'Etudes en Recherche Operationelle*, pp. 175–241, 1987.

[3] F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall, "Entropy: A Consolidation Manager for Clusters," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2009, pp. 41–50.

[4] M. N. Bennani and D. A. Menasce, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, 2005, pp. 229–240.

[5] D. Chess, A. Segal, I. Whalley, and S. White, "Unity: Experiences with a Prototype Autonomic Computing System," in *2004. Proceedings. International Conference on Autonomic Computing*, 2004, pp. 140–147.

[6] G. Tesauro, "Online Resource Allocation Using Decompositional Reinforcement Learning," in *AAAI'05: Proceedings of the 20th National Conference on Artificial Intelligence*. AAAI Press, 2005, pp. 886–891.

[7] G. Tesauro, R. Das, W. Walsh, and J. Kephart, "Utility-Function-Driven Resource Allocation in Autonomic Systems," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 2005, pp. 342–343.

[8] G. Tesauro, N. Jong, R. Das, and M. Bennani, "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," in *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*. IEEE Computer Society, 2006, pp. 65–73.

[9] R. Das, J. Kephart, I. Whalley, and P. Vytas, "Towards Commercialization of Utility-based Resource Allocation," in *ICAC '06: IEEE International Conference on Autonomic Computing*, 2006, pp. 287–290.

[10] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource Management in the Autonomic Service-Oriented Architecture," in *ICAC '06: IEEE International Conference on Autonomic Computing*, 2006, pp. 84–92.

[11] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments," in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 2006, pp. 373–381.

[12] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *IM '07: 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 119–128.

[13] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," in *4th USENIX Symposium on Networked Systems Design and Implementation*, 2007, pp. 229–242.

[14] X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang, "Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center," in *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, 2007, pp. 29–29.

[15] A. Kochut, "On Impact of Dynamic Virtual Machine Reallocation on Data Center Efficiency," 2008, pp. 1 –8.

[16] H. N. Van and F. D. Tran, "Autonomic Virtual Resource Management for Service Hosting Platforms," in *ICES '09: Proceedings of the International Conference on Software Enginnering Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society, 2009, pp. 1–8.

[17] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation And Experience," *Parallel Computing*, vol. 30, p. 2004, 2003.

[18] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[19] R. Farahbod, V. Gervasi, and U. Glässer, "CoreASM: An Extensible ASM Execution Engine," *Fundamenta Informaticae*, pp. 71–103, 2007.

[20] E. Börger, "Construction and Analysis of Ground Models and their Refinements as a Foundation for Validating Computer Based Systems," *Formal Aspects of Computing*, vol. 19, no. 2, pp. 225–241, 2007.

[21] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, 2001.

[22] Y. O. Yazır, C. Matthews, R. Farahbod, A. Guitouni, S. Neville, S. Ganti, and Y. Coady, "Dynamic and Autonomous Resource Management in Computing Clouds through Distributed Multi Criteria Decision Making," University of Victoria, Department of Computer Science, Tech. Rep. DCS-334-IR.