Int. J. Advanced Networking and Applications
Volume: 03, Issue: 02, Pages:1060-1069 (2011)

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

Chandra Mouli Venkata Srinivas Akana
Associate Professor, Department of CSE, AMC Engineering College, Bangalore-83., India.
Email:mouliac@yahoo.co.in
Sundeep Kumar K.
Asst. Professor, Department of CSE, CMR Institute of Technology, Bangalore, India.
Dr. C.Divakar
Professor & Principal, Visakha Institute of Technology, Visakhapatnam, AP.,India
Dr. Ch. Satyanarayana
Associate Professor, Department of CSE, JNTUK, Kakinada.,AP., India.

----------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 technical report, we investigate 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 has 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.

Keywords :Cloud Computing, Distributed architecture, Grid computing, PROMETHEE method, Utility Computing.

Date of Submission : April 04, 2011 Date of Acceptance : May 28, 2011

1. INTRODUCTION

Cloud computing is a popular trend in current computing which attempts to provide cheap and easy access to computational resources. Compared to previous paradigms, cloud computing focuses on treating computational resources as measurable and billable utilities. From the clients' point of view, cloud computing provides an abstraction of the underlying hardware architecture. This abstraction saves them the costs of design, setup and maintenance of a data center to host their Application Environments (AE). Whereas for cloud providers, the arrangement yields an opportunity to profit by hosting many AEs. This economy of scale provides benefits to both parties, but leaves the providers in a position where they must have an efficient and cost effective data center. Infrastructure as a Service (IaaS) cloud computing focuses on providing a computing infrastructure that leverages system virtualization [1] to allow multiple Virtual Machines (VM) to be consolidated on one Physical Machine (PM) where VMs often represent components of AEs. VMs are loosely coupled with the PM they are running on; as a result, not only can a VM be started on any PM, but also, it can be migrated to other PMs in the data center. Migrations can either be accomplished by temporarily suspending the VM and transferring it, or by means of a live migration in which the VM is only stopped for a split second [2]. With the current technologies, migrations can be performed on the order of seconds to minutes depending on the size and activity of the VM to be migrated and the network bandwidth between the two. The ability to migrate VMs 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.

This task as the process of dynamic and autonomous resource management in a data center. In former research, this process is generally carried out through a centralized architecture. The research focuses on the use of utility functions to declare the preferences of AEs over a range of resource levels in terms of utilities. The utility values are then communicated to a global arbiter that computes and performs the resource management on behalf of the entire data center [3].

In this paper, we propose a new approach to the same problem in the context of computing clouds. Our contribution is two-fold. Firstly, 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. Secondly, NAs carry out configurations through Multiple Criteria Decision

Analysis (MCDA) using the PROMETHEEmethod [4], [5].

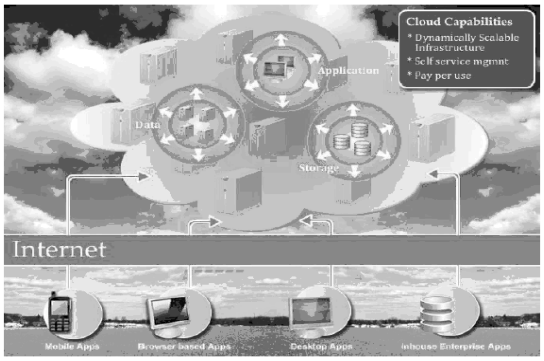

Figure 1. conceptual view of cloud computing.

2. PROBLEM DESCRIPTION

Resource consolidation in data centers can generally be defined as the dynamic and autonomous process of providing mappings between a set of application environments and a pool of shared computational resources. The process is dynamic since the resource usages of the application environments are time variant which stems from two major factors.

- First of all, the amount of computational resources is variable throughout time due to the possibility of addition, removal and temporary unavailability of hardware/ software in a data center (e.g. unexpected failures, scheduled maintenance, etc.).
- Secondly, neither the number of application environments nor their performance level and resource level requirements at an arbitrary instance can be statically determined.

Resource consolidation is also an autonomous process in a sense that its goal is to minimize human intervention during mapping/re-mapping in order to provide a robust self-configuring infrastructure that is more responsive to changing conditions.

The primary actors that benefit from resource consolidation are clients and providers. Clients can simply be defined as the owners of application environments that execute within a data center. From the clients' point of view a data center is merely a pool of resources. One of the expectations of clients is to have a level of abstraction from the low-level operational details of the infrastructure (e.g. servers, switches, topology, software deployment, dynamic allocation of resources, etc.), through which they will be able to define the desired performance— generally referred to as quality of service or service level agreements—of their application environments using high level performance metrics. Based on these high level definitions, clients assume that their application environments are assigned sufficient amounts of computational resources, so that; their performance level requirements are continuously met regardless of the changes in their workload. Moreover, it is also natural for clients to claim a certain degree of control over the quality of their application environments through modification of the high-level performance parameters for various reasons at any point in time (e.g. increasing/decreasing service quality).

The former research in the field can be viewed under the two groups:

(1) Resource consolidation in multiple-server data centers, and

(2) Resource consolidation on a single-server shared centers.

A two-layered architecture of a resource consolidation system for non-virtualized data centers was proposed in the work [3], which is used as a common architecture in a majority of the later research. This architecture consists of a local agent assigned to each application environment, through which the required amount of resources is computed. The information that is generated by the local agents is then communicated to a global arbiter which computes a near-optimal reallocation of resources in the entire data center. This seminal work also integrated this two-layered approach with the usage of utility functions as a measure of desirability of different amount of resources from the application environments' point of view, where the utility values monotonically increase from undesirable amounts of resources to the desirable amounts of resources, between the values of 0 and 1. These utility values are then used to calculate a maximum global utility by the global arbiter under the condition of not exceeding the available resources in a data center. This, in essence, forms the optimization problem which is generally modeled as **Knapsack Problem** where a global near-optimal configuration is computed from the individual preferences of local agents. This same approach which is then merged with forecasting methods based on different analytical models on non-virtualized data centers. While this approach mainly focused on a queuing theoretic approach to the performance modeling
problem, and considered a pure decomposition reinforcement learning approach and their focus on resource consolidation are revealed as:

(1) A purely application environment centric view where data center utilization is not of major concern,

(2) The main goal being the computation of an optimal configuration of resources in entire the data center, and

(3) The absence of the cost of component movements in the formulation of the problem.

Monitoring is simply the phase where information regarding the resource utilization on each computational unit and the resource usage of each application environment in the data center is gathered.

Volume: 03, Issue: 02, Pages:1060-1069 (2011)

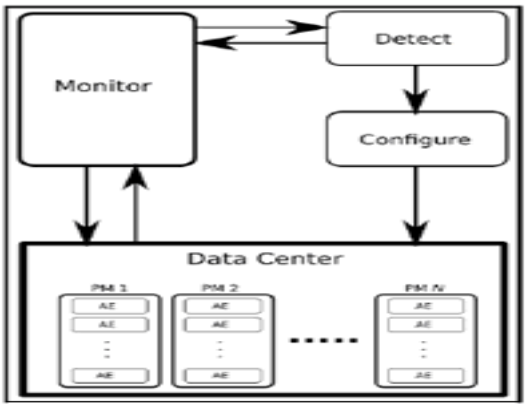

Figure 2. representation of a resource consolidator in terms of the phases of monitor, detect and configure as an observe-detect-react cycle.

A generally employed method is based on periodically sampling data at the end of fixed time intervals. The raw information on resource usage that is collected by the monitoring mechanism is later processed in the detection phase. In the detection phase, the main concern is to capture any anomalies in terms of the violations of performance aggreements, and if there is any, to trigger the configuration phase. In the literature, the detection phase is devised in the two different manners of either treating it as a separate phase.

Fig. 2. The Representation of a Resource Consolidator in terms of the phases of Monitor, Detect and Configure as an Observe-Detect-React cycle. that is entered periodically [3], or by merging it with monitoring to trigger configuration on an event-driven basis where events are often considered to be violations—or predicted violations based on demand and load forecasting— of performance agreements.

Finally, configuration is the phase where the consolidation reacts to the changes in the data center in terms of re-allocating resources for application environments that have undergone changes that have critical affects on their performances. As a result, application environments are re-assigned resources wherever available in the data center, which in turn might require movement of certain components of an application environment between computational units. The nature and effects of such movements are also have attracted a certain amount of interest in the field. In the literature, this configuration phase is generally considered as a Multi-Dimensional Knapsack Problem or as a certain variant of it— Vector Bin Packing Problem, which are NP-Hard. [3].

3. CATEGORIZATION OF RELATED WORK ON RESOURCE CONSOLIDATION IN MULTIPLE SERVER DATA CENTERS- NON-VIRTUALIZED VIRTUALIZED RESOURCE UTILIZATION

In this paper, we propose a new approach to the resource allocation problem through an architecture that distributes the responsibility of configuration among Node Agents (NA). Each NA 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 reconfiguration 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.

4. SYSTEM ARCHITECTURE

We designed the system architecture based on the idea of avoiding the problems of scalability that may emerge as a result of determining and maintaining globally optimal configurations through facilitating a centralized arbiter. We strongly believe that as the data center expands both the computation of optimal configurations and centralizing these computations in a global arbiter might impose serious complexities. As a result of this view, we designed the system as a two-level architecture of

(1) Application agents that are closely coupled with application environments that declare up-to-date resource requirements, and

(2) node agents that are coupled with the computational units in the data center that continuously distribute resources based on the data that application agents declared.

Each application environment in the system is assigned an application agent upon arrival to the data center. The main responsibility of the application agents is to continuously provide the resource requirements that match the performance requirements of the application environments given certain initial performance requirements. Most importantly, the result of these mappings is subject to change as the workloads of application environments vary. Therefore, an application agent should closely monitor the workload of its corresponding application environment and update the resources necessary for it to meet the performance requirements outlined upon arrival to the data center. We also assume, in parallel with the related work, that each application environment will have at least one component during the course of its lifetime, where each component performs a certain task within the objectives of a certain application environments. In our work, each of these components are assumed to be virtual machines to be deployed on computational units. The results of these mappings can be based both on the current workloads and the anticipated future workloads, on a per component basis.

In the second layer of the architecture, however, instead of using a global arbiter to determine the configuration of resources, we assign this responsibility to node agents that are attahed to each computational unit in the data center. Each node agent has the responsibility of monitoring the changes to the resource requirements as declared by the application agents of the corrsponding virtual components that are currently being hosted on the computational unit that the node agent is attached to. Moreover, node agent performs the necessary configurations as the changes in resource requirements of the components impose. These changes in our view are of two natures:

(1) Local re-distribution of the resources as long as the computational can accommodate, and

(2) Moving suitable components to other computational units when the current components cannot be accommodated with the resources.

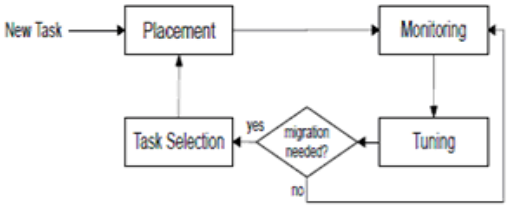

Figure 3. node program.

Fig.3. Node Program the hosting computational unit. It is important to note here that the major focus of this paper is based on the behavior of the node agents during the course of configurations, rather than the modeling of the performance and the mapping function of application environments, for we think that the behaviour of application agents should be investigated in a separate work. Accordingly, in the rest of this paper, we outline the abstract and mathematical model of the node agents.

In this context, we designed the system as a fully distributed network of autonomous node agents each capable of accommodating components—will be called running tasks in the rest of this paper—and, when necessary, delegating tasks to other nodes and handing over the management to the corresponding node agents. We assume that the network maintains a global awareness of the resource availability of nodes and their task assignments. In practice, this awareness can be achieved either by using already established protocols or by having nodes report to a centralized (or a hierarchy) of monitoring units.

The process of dynamically allocating tasks to nodes and maintaining resource distribution among tasks to meet their resource requirements, is modeled as a distributed process carried out by individual node agents in the system. Conceptually, every node—in parallel to running the tasks assigned to it— continuously performs a cycle of four activities (see Figure 3): placement, where a suitable node capable of running the given task is found and the task is assigned to that node; monitoring where the node monitors its tasks and resource requirements as declared by application agents; tuning where the node attempts to adjust its resource assignments locally in respond to changes in task resource requirements and determines if local accommodation is possible; task selection where, if local accommodation is not possible, a task is selected to be migrated to another node and the process loops back into placement. We model every node as a DASM agent that continuously runs its main program. The following ASM program abstractly captures the behavior of the node agents.1. However, in parallel to the four main activities, every node also continuously responds to communication messages it receives from other
nodes.

Figure 4. task lifecycle.

Node Program

```
NodeProgram ≡
  Placement
  MonitoringAndTuning
  if migrationNeeded then TaskSelection
  // the selected task will be migrated in the next cycle
  Communication
```

Since the system is fully distributed, new tasks can be assigned to any node in the system. Once entered into the network, a task goes through a lifecycle starting with being unassigned and ending with being terminated which is when its execution is completed (see Figure 4). For any given task at a given time, one node is responsible to move the task through its lifecycle. The current status of tasks are captured by the function **taskStatus : TASK -> TASK STATUS**

A. Placement

The Placement activity consists of two tasks:

a) finding suitable nodes and allocating resources for unassigned tasks, and
b) assigning allocated tasks to their corresponding nodes.

In the Placement activity, a node looks for the tasks that are either unassigned or allocated. For these tasks, the node is acting like a broker;

In ResourceAllocation, the node picks an unassigned task from its task pool and tries to find a

suitable node with available resources that can perform the task. At this level, we capture the computation of finding such a node by the abstract function **placementNode** : **TASK - > NODE** that given a task suggests node that can run the task. We have,

Resource Allocation

```
ResourceAllocation ≡
  choose t in taskpool(self) with taskStatus(t) = unassigned do
    let n = placementNode(t) in
      if n ≠ undef then
        RequestResourceLock(n, t)
        potentialNode(t) := n
        taskStatus(t) := allocated
        SetTimer(self, t)
      else
        Warning("Cannot find a suitable node.")
```

Placement Node first filters out the nodes in the data center that do not have the resources to host a certain task t. After a list of the nodes that have the necessary resources is provided, the problem is to choose the best node defined on two criteria, resource utilization and rate of change in resource usage. Resource utilization is the criteria that ensures that maximization of resource utilization is taken into account (lower the resources on a node, higher the utilization will be), whereas rate of change in resource usage reflects on the stability of available resource usage on a node (low rate of change points to less likeliness of further migrations, and thus less likeliness of performance drops on the tasks). In order to reflect on both the provider and clients these two criterion can be weighted respectively.

When a node receives a resource lock request for a given task, 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 node; otherwise, it sends a lock rejection. If it does not receive the task within a certain time window, it removes the lock and releases the resources. Upon receiving a lock acknowledgement, the node changes the status of the task to assign and sends the task to the target node. As part of the Communication activity, nodes continuously monitor incoming network messages. If a node receives a task-assignment message with a task that it has alread allocated resources for, it will release the lock, assigns the resources to the task, starts the task and adds it to the set of its assigned tasks.

Process Task Assignment

```
ProcessTaskAssignment(msg) ≡
  // assuming that msg is of type task-assignment
  let task = msgData(m) in
    add task to taskpool(self)
    if task ∈ allocatedTasks(self) then
      AssignResourcesToTask(task)
      ReleaseLock(task)
      add task to nodeAssignments(self)
      StartTask(task)// sets its status to running
      remove task from allocatedTasks(self)
    else
      taskStatus(task) := unassigned
```

B. Monitoring And Tuning

Nodes continuously monitor their resource utilization and their task requirements. There are three main activities under monitoring and tuning: a) for all running tasks, a node monitors
changes in its resource requirements and adjusts resource allocations if needed; if such adjustment is not possible, a task replacement may be triggered; b) a node also monitors its resource utilization and adjust its resource allocation to keep the utilization within a reasonable boundary; c) finally, any resource lock that is timed out (for which a task is not received) needs to be relased. The following ASM rule abstractly captures these three activities:

Monitoring and Tuning

```
MonitoringAndTuning ≡
  forall t in nodeAssignments(self) with taskStatus(t) = running do
    if taskRequirementChanged(t) then
      AccommodateRequirementChange(t)
  if resourceBoundReached(t) then
    Tuning
  ReleaseTimedoutResourceLocks
```

5. PROMETHEE METHOD:

The problem of the selection or the ranking of alternatives submitted to a multicriteria evaluation is not an easy problem. Neither economically nor mathematically usually there is no optimal solution; no alternative is the best one on each criterion. A better quality implies a higher price. The criteria are conflicting. Compromise solutions have to be considered. It starts with general comments on multi-criteria problems.

Max{g1(a),g2(a),gj(a),….,gk(a)a∈ A}

where *a*1 , *a*2 , … *ai* ,… *an* are *n* potential alternatives and *f*1 , *f*2 , … *fj*, … *f*k are *k* evaluation criteria. Each evaluation *fj*(*ai*) must be a real number. Such a matrix can model many real-world applications. In some cases it is an easy task and the matrix is obtained immediately. In other cases it can be a hard problem implying several months of severe consultancy and analysis work, as for

instance when a new production unit must be selected among several possible sites.

6. CONFIGURATION MODEL BASED ON A FORMAL DECISION MAKING METHOD:

I focus on the problem of configuration in terms of moving components of application environments between computational units. This is an action that is performed only when a node agent determines that it is no longer possible to accommodate all of the components that it is currently hosting. Then the problem was defined in an abstract manner in two basic operations:

- Deciding on one or more suitable task(s) remove, and
- Deciding where in the data center to move them to.

The main goal of the first decision—task selection—can be given as the necessity to bring the resource usage on the computational unit within the limits of available resources. However, the choice is not only dependent on these single criteria. It also needs to take into account minimizing the cost of removal and minimizing the probability of further movements of that task from its new host. In this sense, the problem of choice becomes of multiple criteria nature. Then the task selection can be modeled as a multi-criteria decision making problem, where the decision maker is the node agent that detects the need for removal of component(s), and the criteria being maximizing the resource utilization on the computing unit, minimizing the time and the negative effects of the removal, and minimizing the probability of re-removal of the component from its new host. The alternatives can be outlined as the components that can bring the resource usage under acceptable limits (e.g. limits of the host). Furthermore, the decision method to be facilitated should provide means to remove multiple components when necessary without extra the computational burden of re-calculating a best choice.

The main goal of the second decision—node selection— can be given as determining a node that can accommodate the component chosen in the first decision. This, in the first place depends on the amount of resources that a node can provide for the component. Thus the candidate set is defined as the nodes in the data center that can accommodate the chosen components. The resources should be the minimal—but still within limits of resources—on the chosen candidate in order to ensure that the overall utilization will not be low. However, as in the first case it is necessary to define a more comprehensive set of factors that affect this process. The most visible criteria other than the resources in that sense are probability of accommodating the chosen component without the need for further removals from the node, and minimizing the effects of the movement on the currently hosted components. Then these two steps can be considered as multi criteria decision problem as given in:

$$\mathbf{Max\{g1(a),g2(a),gj(a),….,gk(a)a\in A\}}$$

where A is a finite set of possible candidates. $(a_1,a_2,……,a_n)$ and $\mathbf{\{g_1(.),g_2(.),….,g_j(.)……, g_k(.)\}}$ a set of evaluation criteria.

In our case some of the criteria will be maximized while some of them will be minimized based on the step of decision. Then, the decision maker expects to identify a candidate that optimizes all the criteria. In order to deal with this problem, we use the PROMETHEE preference modeling, a specific family of outranking methods. PROMETHEE methods were designed to treat multicriteria problems of type 1 and their associated evaluation table.

The information requested to run PROMETHEE consists of:

1. Information between criteria, and
2. Information within each criteria.

1. **Information between criteria**: relates to the weights of each criteria in the decision problem as a measure of their relative importance. These weights $\{w_j, j=1,2,….,k\}$ are nonnegative numbers, independent from measurement units of the criteria. The higher weight that the more important the criterion during course of decision-making. In general, there is no objection to using normalized weights**:** $\mathbf{\Sigma^k_{j=1}\ w_j=1.}$ Assigning weights to the criteria in our design is the responsibility of the providers in a sense that these weights represent the providers' preferences as to how the resources should be consolidated (e.g. taking maximizing resource utilization, minimizing agreement violation, or a balance between them as the ultimate goal).
2. **Information within criteria**: relates to the comparison of based on how good an alternative with respect to the others based on a per criteria basis. In contrast with Utility Functions, PROMETHEE does not allocate an intrinsic utility to each alternative, neither globally, nor on each criterion. Instead, the preference structure is based on pair wise comparisons. In this case the deviation between the evaluations of alternatives on a particular criterion is considered. These preferences can be considered as real numbers varying between 0 and 1. Then the node agent—the decision maker in general—facilitates a function for each criterion:

$$\mathbf{: P_j(a,b)=F_j[dj(a,b)]\forall a,b\in A} \quad \text{---- (1)}$$

Where, $\mathbf{D_j(a,b)=g_j(a)-g_j(b)}$

And for which

$$\mathbf{0<=p_j(a,b)<=1}$$

In case of a criterion to be maximized, this function provides the preference of a over b for observed deviations between their evaluations on criterion $\mathbf{g_j(.)}$. For criteria to be minimized, the preference function should be reversed or alternatively given by:

$$\mathbf{: P_j(a,b)=F_j\text{-}dj(a,b)\forall a,b\in A} \quad \text{---(2)}$$

The pair $\{g_j(.),P_j(a; b)g$ is called the generalized criterion associated to criterion $g_j(.)$. Such a generalized criterion has to be defined for each criterion. In order to facilitate this identification, six types of particular preference functions have been proposed:

- Usual Criterion,
- U-Shape Criterion,
- V-Shape Criterion,
- Level Criterion,
- V-Shape with Indifference Criterion, and
- Gaussian Criterion.

Let us define the Aggregated Indices and the Outranking Flows as defined in PROMETHEE. For aggregated indices, Let $a,b \in A$

$$: \Pi(a,b)= \Sigma^{k}_{j=1}\, p_j(a,b)wj \quad \text{--- (3)}$$

$$: \Pi(a,b)= \Sigma^{k}_{j=1}\, p_j(b,a)wj \quad \text{--- (4)}$$

Π(a,b) is expressing with what degree a is preferred to b over all the criteria and Πb; a with what degree b is preferred to a. In most cases there are criteria for which a is better than b, and criteria for which b is better than a, consequently Π (a, b) and Π (b,a) are usually positive. The following properties hold

for all(a,b)∈A

Π(a,a)=0

0<=Π(a,b)<=1

0<=Π(b,a)<=1,0 <=Π(a,b)+ Π(b,a)<=1

Π(a,b) 0 implies a weak global preference of a over b;

Π(a,b) 1 implies a strong global preference of a over b;

The outranking flows can be defined as follows. Each alternative a in A face (n -1) other alternatives in A. The, let us define the outranking flows as positive and negative outranking flows, where positive outranking flow of a:

$$: \phi^{+}(a)=1/n\text{-}1\sum_{x\in A}(\Pi(a,x))) \quad \text{--- (5)}$$

$$: \phi^{-}(a)=1/n\text{-}1\sum_{x\in A}(\Pi(x,a))) \quad \text{--- (6)}$$

The positive outranking flow represents how an alternative is outranking all the others, meaning its power or outranking character. Thus, the higher ϕ+(a), the better the alternative. the lower ϕ+(a) better alternative.

7. EXPERIMENTAL SETUP

We have built a simulation environment using the C programming language in order to test our approach and compare it to certain other strategies. The environment simulates the changes and configuration actions in a datacenter in a stepwise manner, where between each step there is a fixed time interval. We have chosen this view in order to be able to compare each strategy to be tested on the simulation environment at any instance of the simulation with identical conditions.

Accordingly, every virtual component declare new resource usages at each step and the strategy that is being used to configure the data center reacts with respect to the new conditions. We define the resource dimensions as CPU usage, memory usage and bandwidth usage.

Layer one consists of modules that represent the virtual components and physical machines. Each virtual component in the simulation environment represents a task that is independent of the others, and each of those tasks are responsible to update their resource requirements at each step. Virtual components in our simulation environment are considered in two groups: **batch and online processes**. Virtual components of batch type update their resource requirements as random noise signals based on certain distributions (e.g. uniform, poisson, exponential, normal, pareto, etc.). Each virtual component update their resource requirements independently on each resource dimension where each dimension may have different distributions. Note that an application environment can be represented by a number of virtual components, and in this case an application environment that represents a batch process is assumed to be represented by one or more of batch-type virtual components. The online type virtual components work in on and off periods. The on periods represent the times when there are requests to be serviced, and the off periods represent idle intervals. During their on periods online virtual components generate resource requirements in a way that is identical to the batch tasks, while during the off periods they do not pose any recource requirements. The distribution of the length of on and periods are modeled to reflect the selfsimilar behaviour seen in online environments, that is, if one of the periods' duration is exponentially (or poisson) distributed the other has a heavy-tailed distribution (e.g. pareto). In the simulation environment each virtual component has a certain life time determined when they arrive. Life time of a virtual component is defined either in terms of a certain number of steps or as unlimited which means that it will reside in the data center forever. The physical machines represent bins that can acommodate one or more virtual components.

In the simulation environment it is assumed that a virtual component cannot span multiple physical machines. Each physical machine keeps a record of the virtual components it hosts and the amount of resources in use by its virtual components. In addition, physical machines have an upper bound for resource usages to ensure that they do not have more virtual machines than they can acommodate and response times are kept at reasonable levels. Finally, they are capable of migrating their virtual components to other physical machines when necessary. Layer two consists of a single module, the global monitor. The global monitor keeps a record of each virtual component and each physical machine in the system. While the virtual components are recorded only as allocated or not allocated without the details of their resource usages, physical machines in the system are recorded with respect to their available resources in each

dimension. The second responsibility of this module is to reply to the queries about the states of the physical machines in the data center. Some of the typical queries that are replied to by the global monitor are the first physical machine with the necessary resources, non-empty physical machine(s) that have a certain level of available resources on each dimension, empty physical machines, etc.

Layer three consists of the strategy modules that represent different configuration methods that are used to consolidate resources. Every strategy module has to assign arriving virtual components to physical machines and re-configure the data center in terms of facilitating necessary movements to keep each virtual component below their upper bounds. The assignments and re-configuration is performed at each step during the simulation. We have implemented four strategy modules with two of them represent a centralized control that aims for a global near-optimal configuration, while the other two represent distributed control with local configuration. By local configuration we mean that the nodes are not concerned with the global consolidation but are focused on holding their resource usages below their resource upper bounds. The two methods we implemented to represent the centralized control and global configuration strategy are the wellknown bin packing First Fit (FF) and First Fit Decreasing (FFD) methods. FF performs configuration at each step by iterating through the list of virtual components that are present in the data center—either allocated or not allocated—and placing each of them on the first physical machine that can accommodate. FFD performs the exact same operation at each step with the difference of first sorting the list of virtual components in a decreasing order, that is, the virtual components that have the greater resource requirements get placed first.

Implementation Requirement

Software Requirement:

- The language chosen for this project is Java swing and softwareused in NetBeans 6.8.
- Operating System: Microsoft windowsXP.

i. Selection of the Platform:

Windows XP provides the most dependable version of windows ever-with the best security and privacy features Windows has ever provided. Overall,Security is improved Windows XP is available in two editors-Windows XP Home Edition for Home Use and Windows XP professional for business of all sizes. Security features in Windows XP Home Edition make it even safer for you to stop and browser on the Internet. Particularly if you use always-on connection such as cable modem and DSL. Window XP Professional includes all of the security capabilities of Windows XP Home Edition, plus other security management features. These important new security features will reduce your IT costs and enhance the security of your business systems. Windows XP Home Edition Security service have been ddesigned to be flexible, and take into account a wide varity if dcurity and privacy

We model every node as a DASM agent that continuously runs its main program. The following ASM program abstractly captures the behavior of the node agents. However, in parallel to the four main activities, every node also continuously responds to communication messages it receives from other nodes. Since the system is fully distributed, new tasks can be assigned to any node in the system. Once entered into the network, a task goes through a lifecycle starting with being unassigned and ending with being terminated which is when its execution is completed.

In our approach also, the physical machines are responsible for local configurations. We adapted PROMETHEE II as a decision making procedure for determining the configurations.
The configuration is performed only in two cases:
(1) Resource upper bound is reached, and
(2) Resource lower bound is reached.

When the upper bound is reached a physical agent tries to move the most suitable virtual component(s) in order to bring the resource usage below the upper bound again.

This is performed using PROMETHEE II as follows. First, the physical machine agent filters a list of virtual components that can bring the resource usages below the upper bound, if this list is non-empty, then it performs the pairwise comparisons in order to sort the alternatives based on their net outranking flows. During these pairwise comparisons, the criteria are resource usages in each dimension and resource usage variabilities in each dimension. The variabilities are determined based on a certain set of most recent data on resource usages. The weight of these criteria is left to the providers preference based on their goals. If the list is empty, then the physical machine agent uses all of the virtual components for the pairwise comparisons and the sorting of the net outranking flows of each alternative. Once the most suitable virtual component is picked, the physical machine agent contacts the global monitor in order to retreive a set of nodes that can acommodate the chosen virtual component. Using the same criteria, the most suitable one is picked from the set of alternative physical machines.

Finally, the machine with resources above the upper bound moves the chosen virtual component to the chosen physical machine. In the case where there is no physical machine that can acommodate the virtual component in the data center, the cycle is repeated for the second best virtual component. If none of the virtual components can be replaced within the data center, a new node is created for the most suitable virtual component. In the case where a physical machines resource usages are below a lower bound, the physical machine agent—as in the simple method—tries to remove its virtual components and turn itself off. During this process, the virtual machines are not

picked based on any criteria, however for the choice of destination the pairwise comparisons are again used.

8. RESULTS AND VALIDATION

The test scenarios based on the number of virtual components to be hosted on a data center and the necessary size of the data center in terms of the physical machines to be used. In that sense, we focus on three distinct data centers with a certain number of virtual components:

- Extra Small scenario represents a data center with 600 physical machines and 5000 virtual components to be hosted, Small scenario represents a data center with 1300 physical machines and 10000 virtual components, and Medium scenario represents a data center with 2500 physical machines and 20000 virtual components.

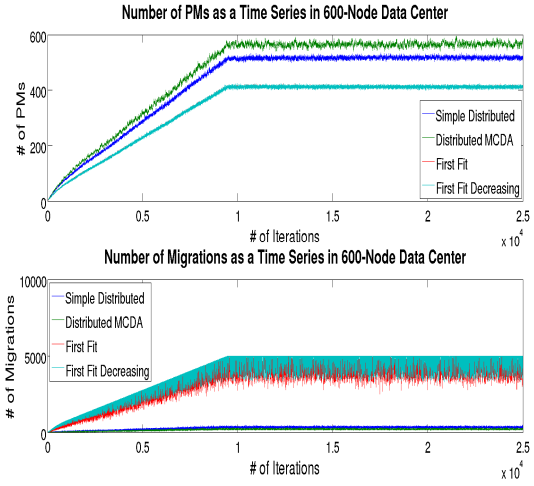

Figure 5. the number of physical machines per algorithm

Fig.5. shows represent the number of physical machines per algorithm that are in use at each step. The bottom figure represents the number of migrations per algorithm that are performed to configure the data center.

Figure represents the number of physical machines per algorithm that are in use at each step. The bottom figure represents the number of migrations per algorithm that are performed to configure the data center. (Number of virtual components, resource requirements per virtual components).note that each scenario starts with 0 virtual components at iteration 0 and iteration by iteration reaches the specified final number of virtual components, and the physical machines are not utilized unto 100%, instead we set the upper bounds per each method to 60% for feasible response times. When a virtual component with finite lifetime departs from the data center, it is replaced by a fresh virtual component arrival. We have chosen such an implementation to be able to evaluate the methods under a state that can be called steady in terms of the number of virtual components.

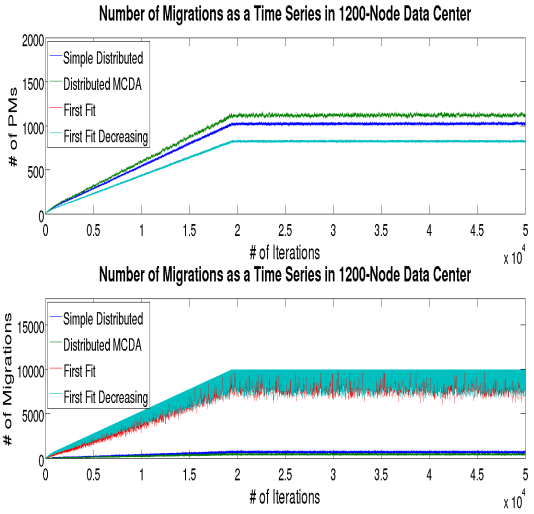

Figure 6. the number of physical machines per algorithm

Fig.6 shows represent the number of physical machines per algorithm that are in use at each step. The bottom figure represents the number of migrations per algorithm that are performed to configure the data center.

It can clearly be seen in these figures that the centralized control methods (FF and FFD) need a number of physical machines substantially less than the distributed (simple method and our approach) approaches. This means that the configuration in the centralized cases result in higher utilization of the data center. However, it is also Cleary seen in the migration measures that the centralized methods are performing an unreasonable number of migrations to reach that utilization level. In the distributed cases, however, the number of migrations that are performed for configuration is much lower with a tradeoff.

Fig.7. shows represent the number of physical machines per algorithm that are in use at each step. The bottom figure represents the number of migrations per algorithm that are performed to configure the data center.

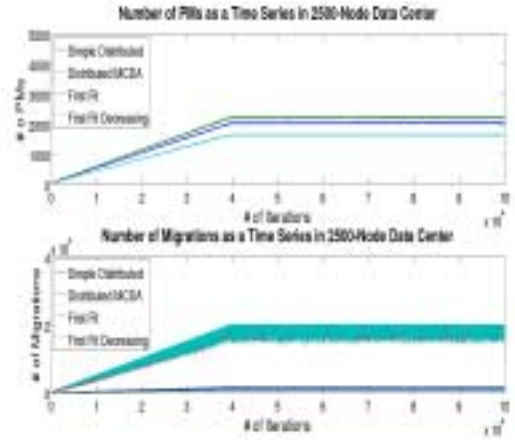

Figure 7. the number of physical machines per algorithm

9. 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 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: View of overall CPU utilizations of the data center per physical machine when First Fit method is used. View of overall CPU utilizations of the data center per physical machine when First Fit Decreasing method is used. View of overall CPU utilizations of the data center per physical machine when the simple distributed method is used. View of overall CPU utilizations of the data center per physical machine when our distributed method is used.

FUTURE WORK:

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] Gerald J. Popek, *Formal Requirements for Virtualizable Third Generation Architectures* (Infocom)

[2] William E. Walsh, Gerald Tesauro and Jeffrey O. Kephart, *Utility Functions in Autonomic Systems* (Hawthorne, NY 10532, 19 Skyline Drive)

[3] Mohamed N. Bennani and Daniel A. Menasc´e, *Resource Allocation for Autonomic Data Centers using Analytic Performance Models* (George Mason University 4400 University, Infocom)

[4] Mohamed N. Bennani and Daniel A. Menasc´e, *Resource Allocation for Autonomic Data Centers using Analytic Performance Models* (George Mason University 4400 University, Infocom)

[5] Laura Grit, David Irwin and Aydan Yumerefendi, *Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration* (Infocom)

[6] Zhikui Wang, Yuan Chen, Daniel Gmach, Sharad Singhal and Brian J. Watson, *Appraise: Application-Level Performance Management in Virtualized Server Environments* (Infocom)

[7] Bo Peng, Bin Cui and Xiaoming Li, *Implementation Issues of a Cloud Computing Platform* (Infocom).

Authors Biography

Chandra Mouli Venkata Srinivas Akana received his Masters Degree in Computer Science & Engineering from Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India. He is currently working as a Associate Professor in the Department of Computer Science & Engineering, AMC Engineering College, Bangalore, Karnataka, India. His current research is focused on Software Engineering, Ad-hoc Networks, Network Computing & Security, Data mining & Warehousing and Database Management System. He has published several papers in National & International Journals. He has guided to more than 70 academic projects at Postgraduate & under Graduate level. He is a member of the IEEE, CSI & ISTE.

K Sundeep Kumar received the M.Tech (IT) from Punjabi University in 2004, and ME (CSE) from Anna University in 2009. He is working as Asst. Professor in the Department of Computer Science & Engineering, CMR Institute of Technology, Banglaore. He presented more than 10 papers in International and national Conferences. His research interests include OOAD, Software Engineering and Data Warehousing. He is a member in ISTE.

Dr. Divakar C. received his Masters Degree as well as Doctorate in Computer Science & Engineering from Andhra University, Visakhaptnam, and Andhra Pradesh, India. He is currently working as Professor & Principal for Visakha Institute of Technology, Visakhapatnam, Andhra Pradesh, India. He is guiding six Ph.D. research Scholars. His current research is focused on Bio-informatics, Software Engineering, Ad-hoc Networks, Network Computing, Image Processing & Database Management System. He has published several papers in National & International Journals. He has guided several academic projects at Postgraduate & under Graduate level.

Dr. Ch.Satyanarayana received his Masters Degree as well as Doctorate in Computer Science & Engineering. He is currently working as Associate Professor in the Department of Computer Science & Engineering and Controller of Examinations of JNTUK, Kakinada, Andhra Pradesh, India. He is guiding Five Ph.D. research Scholars. His current research is focused on Software Engineering, Ad-hoc Networks, Network Computing, Image Processing, Data Mining & Database Management System. He has published several papers in National & International Journals. He has guided several academic projects at Postgraduate & under Graduate level.