Dynamic Virtual Machine Placement in Cloud Computing

Chapter · January 2017

DOI: 10.4018/978-1-5225-1721-4.ch006

2 authors:

Arnab Paul
Virginia Polytechnic Institute and State University
7 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Bibhudatta Sahoo
National Institute of Technology Rourkela
190 PUBLICATIONS **646** CITATIONS
SEE PROFILE

Chapter 6
Dynamic Virtual Machine Placement in Cloud Computing

Arnab Kumar Paul
Virginia Tech, USA

Bibhudatta Sahoo
National Institute of Technology Rourkela, India

ABSTRACT

The aim of cloud computing is to enable users to access resources on demand. The number of users is continuously increasing. In order to fulfil their needs, we need more number of physical machines and data centers. The increase in the number of physical machines is directly proportional to the consumption of energy. This gives us one of the major challenges; minimization of energy consumption. One of the most effective ways to minimize the consumption of energy is the optimal virtual machine placement on physical machines. This chapter focuses on finding the solution to the problem of dynamic virtual machine placement for the optimized consumption of energy. An energy consumption model is built which takes into account the states of physical machines and live migration of virtual machines. On top of this, the cloud computing model is built. Unlike centralized approaches towards virtual machine placement which result in many unreachable solutions, a decentralized approach is used in this chapter which provides a list of virtual machine migrations for their optimal placement.

INTRODUCTION

Cloud computing is a technique for helping users to have access to computing resources on demand with very minimal effort in management. It has emerged as a popular computing model in order to support processing of large scale data. There are several cloud service providers, for example, Amazon, Microsoft, Google, IBM and Yahoo which have built platforms for other enterprises and cloud users in order to access the services provided in the cloud.

The areas in which cloud computing is used vary from nuclear physics to geothermal experiments. Thus we need data centers to help provide powerful computation resources for these vital areas. A Data Center (DC) consists of a large number of Physical Machines (PM) arranged on racks and packed densely

DOI: 10.4018/978-1-5225-1721-4.ch006

in order to increase space utilization. In cloud computing, one of the key concepts for data center management is virtualization. The most important advantage of virtualization is the capacity to run many instances of operating system in a single PM thus making maximum use of the hardware capacities of the PM which helps immensely in saving money for energy and hardware costs. These individual instances of operating system are called Virtual Machines (VM). Thus, in cloud computing, users access the computing resources of a DC with the help of VMs.

The scheduling of VMs in DCs and VM scheduling on PMs form an important part in cloud computing now more than ever because of the growing number of users. VM scheduling has a massive effect on the system's performance and throughput. Thus, the two types of placement of VMs include Virtual Machine Placement (VMP) over DCs and VMP over PMs. The effect that VMP provides is directly related to the Quality of Service (QoS) provided by the services in the cloud. The major objective of VMP is to make maximum utilization of a data center, thus making use of a lesser number of DCs. This helps in achieving more availability and flexibility of DCs, while reducing the costs incurred during operating and maintaining hardware.

Anatomy of Cloud Computing

In cloud computing, virtualization is the most important factor in giving dynamic and scalable architectures. Apart from providing the important aspects of resource sharing and scalability, virtualization also contributes to the capability of virtual machine migration between PMs in order to balance the load (Jones, 2010).

Figure 1 depicts the key elements in a single node in the environment of cloud computing. The component of virtualization in a cloud node is given by the Hypervisor, which is also called as the Virtual Machine Monitor (VMM). This layer is responsible for providing the interface of executing many instances of operating systems in one PM. Thus, hypervisors create virtual machine objects which provide encapsulation for operating system, applications and configuration. Another important aspect in a cloud node known as device emulation is given either in the hypervisor or as a VM. VM management takes place both in local PMs as well as in global DCs.

Thus in order to form a DC, the cloud node shown in Figure 1 is multiplied over a network provided with the management orchestration over the complete infrastructure:

- **Hypervisors:** This is the initial level of a PM (a single cloud node). It provides the virtual operating platform, thus managing the execution of guest operating systems, known as VMs. The VMs share the virtualized hardware resources of the cloud node. One of the best examples of hypervisors for production environments is the Linux Kernel Virtual Machine (KVM).
- **Device Emulation:** Hypervisors are responsible for providing the platform for sharing virtualized physical resources. But in order to achieve complete virtualization, the whole cloud node must be virtualized. This is the job of a device emulator. QEMU is the example of a complete package combining emulator and hypervisor.
- **Virtual Networking:** As the number of cloud users grow, the number of VMs increases. In order to achieve intensified networking system, virtual machines are required to communicate on a virtualized network rather than on a physical level. This reduces the load on the physical infrastructure of the system. Virtual switches are also introduced for effective communication among the VMs.

Figure 1. Core elements of cloud node

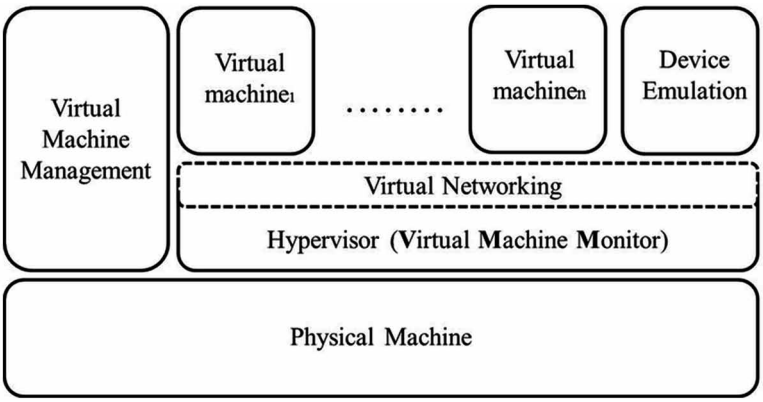
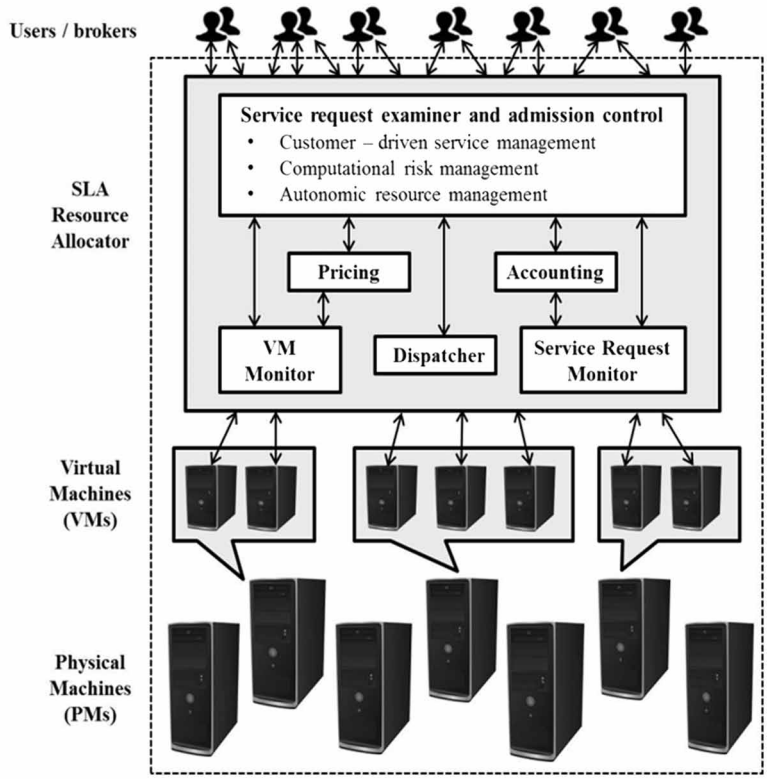


Figure 2. Cloud computing architecture



Cloud Computing Architecture

In the environment of cloud computing, cloud users are dependent on Cloud Service Providers (CSP) for fulfilling their needs. Therefore, certain QoS parameters need to be maintained by CSPs which are mentioned in the Service Level Agreement (SLA). To achieve this, traditional resource management

architecture cannot be used. Thus, we have market oriented architecture of cloud computing which is shown in Figure 2.

The market oriented architecture of cloud computing has the following components.

Users or brokers acting on behalf of the cloud users, submit their job requests to the cloud data center for processing.

The interface between CSP and user is provided by the SLA Resource Allocator. When a user submits a user request, a decision is made by the Service Request Examiner whether to accept or reject the request. It checks the requirements of the QoS parameters. It takes the decision by looking at the information of the resource availability provided by the VM Monitor and workload availability provided by the Service Request Monitor, to ensure no overloading of resources occur. After consideration of these parameters, user requests are assigned to the VMs and a decision is taken regarding the resource requirements of the allotted VMs.

Pricing operation takes decisions regarding the prices to be charged from the cloud users. These prices can be charged on the basis of the user request, or the time of submission, or availability of cloud resources, or simply a fixed rate. This aids in prioritizing the resource allocation process in a cloud DC.

Another important operation is known as Accounting which keeps an account of the actual consumption of resources. This helps in the calculation of the final price to be charged from the cloud user. It also helps in making an informed decision by the service examiner by keeping a tab of the historical resource usage consumption.

VM Monitor keeps a periodic check on the requirements of various resources and the availability of VMs. Dispatcher is accountable for initiating the execution of the user requests on the VMs allotted. Service Request Monitor has the important task of maintaining an account of job requests' execution.

Virtual Machine Placement in Cloud Computing

A two-tiered approach is followed when distributing the VM requests. There are a large number of PMs in a DC. A CSP can have many DCs. The first step is to optimally distribute the VM requests over DCs. Next, the requests are distributed over PMs.

The virtual machine placement can be of the following two types.

- Static virtual machine placement
- Dynamic virtual machine placement

Static Virtual Machine Placement

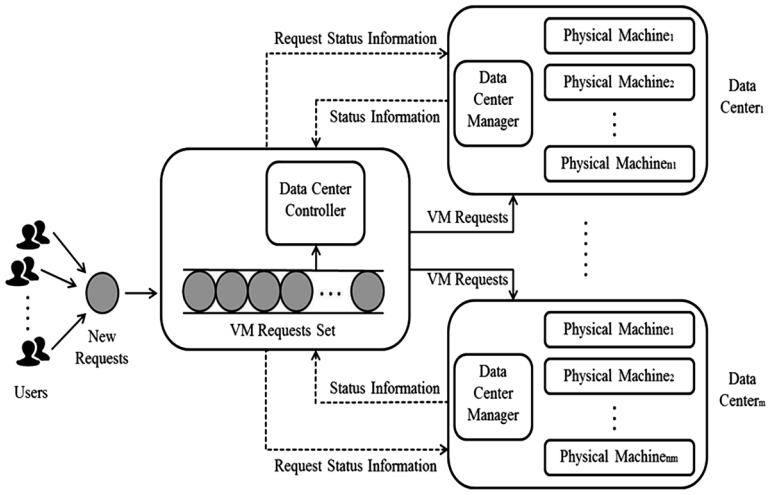
Static VMP is performed when system is in offline mode or during startup of the system. This forms the initial VMP in the environment of cloud computing. There isn't any previous VM mapping. This VMP does not take into consideration the VM states, PM states and the rate of VM requests.

Below is the diagrammatic explanation of the centralized model for static virtual machine placement following the two tier approach. First, VM placement over data centers and then the placement of requests over PMs in a particular data center are shown.

Figure 3 represents the flow of VM Requests from the users till they are distributed over data centers.

Cloud users use the services provided by the cloud service provider and issue requests. These requests are in the form of virtual machine requests since every request will be completed on a virtual machine

Figure 3. Static virtual machine placement over data centers



on top of a physical machine. The VM requests comprise the VM Requests Set. Every data center has multiple physical machines and a Data Center Manager to control all the PMs. In order to control all the data centers, a Data Center Controller is used.

The data center controller receives the VM requests set. It then requests status information from all the data center managers which have the information of their respective data centers. The VM requests contain information of the resources (CPU, RAM, Network etc.) needed in order to complete the request. The data center managers send information of the available resources to the controller. The data center controller optimally distributes VM requests to the data center managers following a heuristic algorithm. This approach to distribute VM requests is a Centralized Approach, where the decision to schedule requests depends on the central controller.

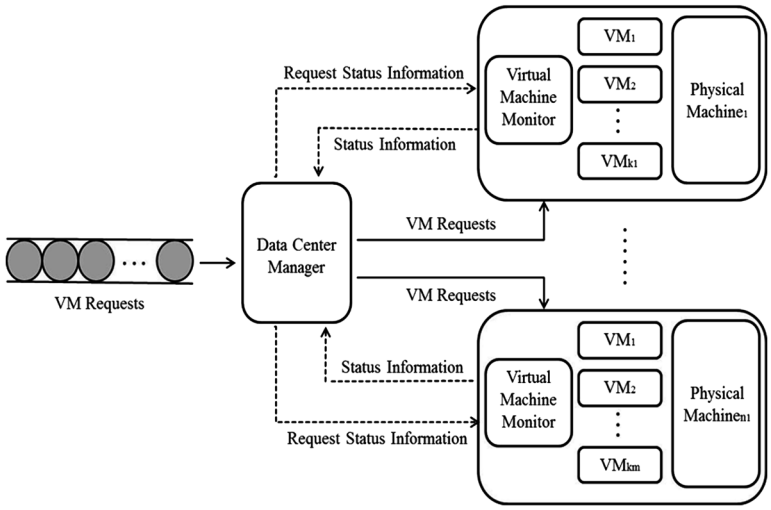
Figure 4 depicts the static placement of virtual machine requests in physical machines in a single data center.

In a data center, there are multiple physical machines on top of which lies the Virtual Machine Monitor [VMM] (Hypervisor). The VMM has the responsibility for virtualization in PMs. The data center manager sends requests to the VMMs to provide the status information of all the physical machines. The data center manager already has the VM requests' information. Upon arrival of the status information from the VMMs, the DC manager places the VM requests to the individual physical machines in order to process them and sends back the response to the cloud users. The placement of VM requests over physical machines is also a centralized approach as the decision is taken by a central authority, in this case the data center manager.

Dynamic Virtual Machine Placement

If an existing mapping of VMs onto PMs is present, we go for dynamic placement of virtual machines. The main goal of dynamic VM placement is to achieve optimum solutions from the already available mapping of VMs at minimal cost. The optimality parameters may vary from minimization of the response

Figure 4. Static virtual machine placement over physical machines



time to the minimization of energy consumption or a combination of multiple parameters. The rate of arrival of user requests as well as the states of both VMs and PMs needs to be considered while taking a decision. Below is an example of dynamic virtual machine placement.

Figure 5 shows the initial solution (i) for the dynamic VM placement problem. Seven VMs are placed over four PMs. PM_4 is in off state since no VM is running on it. Assuming that all the PMs have same amount of resources, the VMs should be dynamically distributed over different PMs in order to reduce energy consumption.

One such solution is shown in Figure 6. In order to reach solution s_1 from (i), live migrations need to be performed. The list of migrations is: VM_5 from PM_2 to PM_3, migration of VM_1 from PM_1 to PM_2 and finally migrate VM_4 from PM_2 to PM_1.

Dynamic VMP is explained in detail later in the chapter.

Figure 5. Initial solution i

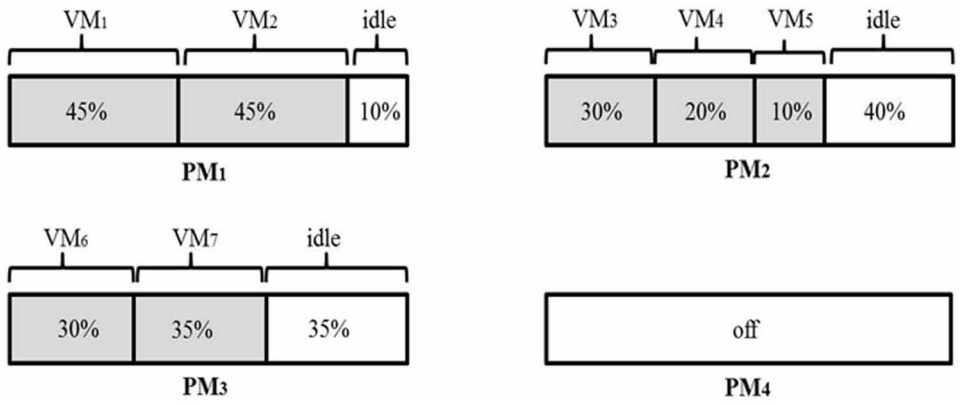
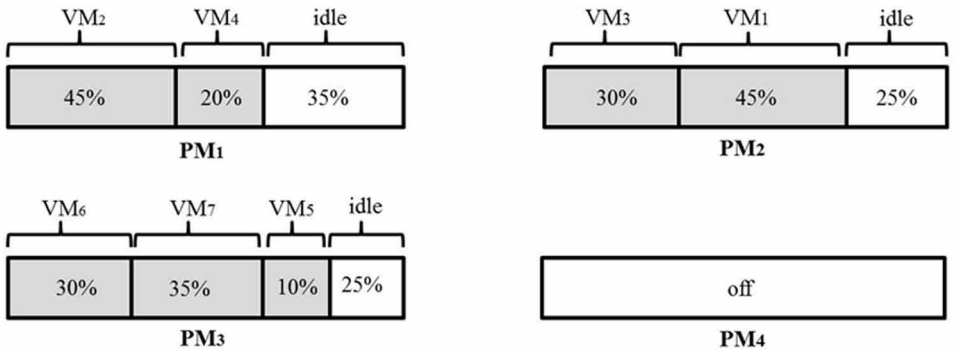


Figure 6. Target solution s_1



Problem Statement

The dynamic virtual machine placement problem has been solved by many researchers. But they haven't considered the live migrations and thus may provide unreachable solutions to the problem. The primary objective of the research in this chapter is to model a cloud computing framework such that the dynamic virtual placement problem can be solved while providing the live VM migrations to be executed, in order to reach the solution.

This can be elaborated as follows:

- The motive is to model the dynamic placement problem in such a way that the states of virtual machines and physical machines are taken into account. Also the list of executable live VM migrations must be provided so that the optimal solution can be obtained.
- The problem if solved using centralized evolutionary techniques has a risk of providing unreachable solutions. Thus, decentralized approach using congestion game model must be used in order to provide the solutions to the dynamic virtual machine placement problem.
- In real world scenario, physical machines may act selfishly and not cooperate to achieve an optimal solution to the problem. Thus both cooperative and non-cooperative approaches should be discussed in order to reach to an optimal solution.

The solution to the dynamic VM placement problem is to be achieved taking into consideration the following constraints:

- **Capacity Constraint**: This condition ensures that the total resource requirements of all the VMs running on a specific PM should be less than or equal to the total resource availability of that particular PM.
- **Placement Constraint**: This constraint checks the criteria that a virtual machine should run on only a single physical machine.
- **SLA Constraint**: The cloud model to be incorporated in the real world should follow QoS parameters. Thus the service level agreement constraint keeps a tab on the QoS parameters enlisted in the agreement and ensures that they are not violated when providing the optimal solution.

BACKGROUND

Dynamic VM Placement in Cloud Computing

Dynamic placement of VMs places VMs based on an existing mapping which is in contrast to the static placement of VMs which starts with no mapping. Dynamic VM placement aims to reach optimal solutions from the existing mapping at minimum cost. This would not shut down or stop the already running VMs, thus the placement solution should provide the list of live migrations to be executed in order to reach the optimal state from the existing state. The whole process is in contrast to the static placement of VMs where VMs can be stopped and restarted which increases the energy consumption and thus degrades the complete system performance. In order to execute dynamic placement of VMs, states of physical machines should also be considered.

Dynamic VM Placement problem can be illustrated in Figures 5, 6, and 7. Figure 5 shows the initial solution (i) for the dynamic VM placement problem. Distribution of seven VMs over four PMs is shown. PM_4 is in off state since no VM is running on it. Assuming that all the PMs have same amount of resources, the VMs should be dynamically distributed over different PMs in order to reduce energy consumption. One such solution is shown in Figure 6. The list of migrations has already been given in a previous section. We can reach another solution s_2 from (i) as shown in Figure 7. Since there isn't much space in PM_1 and PM_2 for the direct interchange of VM_1 and VM_3, there needs to be an involvement of PM_3. Thus the migrations needed for solution s_2 to be reached are: the extra migration of VM_3 from PM_2 to PM_3, VM_1's migration from PM_1 to PM_2 and migrating VM_3 from PM_3 to PM_1. But in order to reach s_2 from (i) will result in more cost than to reach solution s_1 from (i) because of the migrations involving larger VMs. VM_1 and VM_3 could have been stopped and restarted in order to facilitate direct interchange but that would involve more cost. Also the involvement of PM_4 for the interchange of VM_1 and VM_3 would result in higher costs since it would then consider the cost of state change for PM_4.

Dynamic virtual machine placement problem is NP-hard. There have been existing researches where attempts have been made to map the dynamic VM placement problem to bin-packing problem which led to the development of evolutionary algorithms like, genetic algorithms or particle swarm optimization algorithm. As explained before, many problems arise when VMs have to be placed dynamically. While the evolutionary algorithms like genetic algorithms tries to provide optimal solutions to the problem, there may be solutions which are unreachable.

Figure 7. Target solution s_2

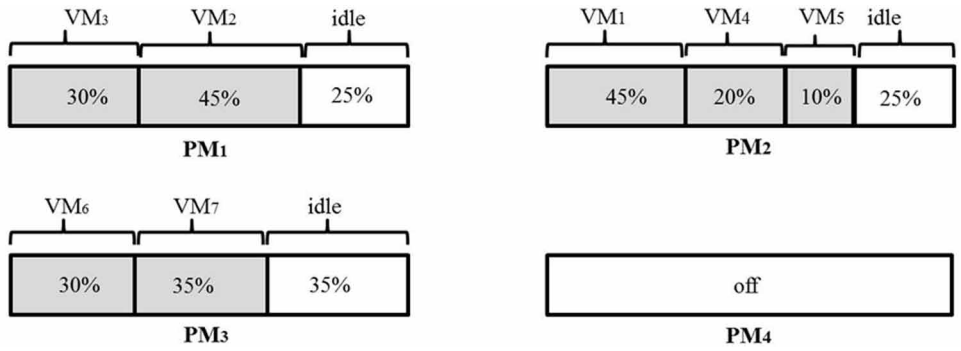


Figure 8 shows an example of unreachable solution to the dynamic VM placement problem. Figure 8(a) shows the placement of 4 VMs onto 2 PMs which form the initial solution sol_0. The optimal solution 'sol' is shown in Figure 8(b) which brings down the energy consumption by a considerable amount. But the solution is unreachable since there is no executable migration route from (a) to (b). In order to achieve the solution, a third PM has to be switched on to handle the migrations which will again consume more energy. Thus, in case of evolutionary algorithms, judgment has to be made whether the optimal solution is reachable or not.

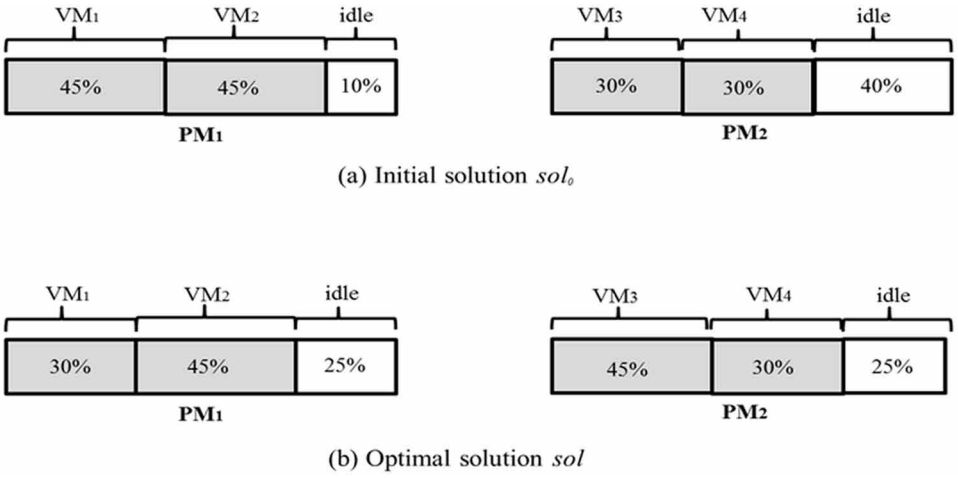
In this chapter, decentralized decision making is used with the help of congestion game theory in order to find solutions to the dynamic VM placement problem. In Congestion Games, a group of players is modeled to share a resource set. Every player selects a subset of resources from the resource set in order to maximize the payoff (Milchtaich, 1996). Here, PMs will be modeled as players which will select a subset of VMs from the VM set in order to minimize the consumption of energy. Nash Equilibrium is attained after a finite number of iterations. This intelligent algorithm uses the initial mapping to obtain the optimum solution and also generates the list of executable VM live migrations to reach the optimal state. Thus no solution is unreachable.

Game Theory

Turocy et al. (2001) defines game theory as the formal study of cooperation and conflict. Hotz (2006) describes a game as a set of players and their possibilities to play the game by following some rules (strategies). The players can be individuals, agents or organizations. The main subject of game theory is the situation where the result is mattered not only by the decision of a single player but others as well.

The earliest instance of formal game theoretic analysis is Antoine Cournot's study of duopoly in 1838. In 1921, a formal theory of games was suggested by Emile Borel. This was further studied upon by John von Neumann's theory of parlor games in 1928. The basic terminology and setup of game theory was established in Theory of Games and Economic Behavior by von Neumann and Oskar Morgenstern in 1944. John Nash in 1950 showed that the games with finite number of players always have a point of

Figure 8. Unreachable solution



equilibrium where the players choose their best strategy taking into consideration the strategies of other players. This equilibrium is known as Nash Equilibrium, named after John Nash. This pivotal point in non-cooperative game theory has been used in various fields from sociology, biology to computer science.

There are mainly two types of game scenarios:

- Cooperative Game
- Non-Cooperative Game

Cooperative Game

Xhafa et al. (2010) defines cooperative game as the game scenario where players form coalitions in advance to discuss their actions. Cooperative game theory investigates such coalition games by studying how successfully a coalition divides its proceeds.

The applications of cooperative game theory are mainly seen in cases related to political science or international relations. In cloud computing, cooperative game theory is used when data centers are managed by a single service provider or the providers form a coalition such that the strategies of virtual machine placement are known by all the physical machines and respective actions are taken to maximize the payoffs of all the SPs.

Non-Cooperative Game

The term non-cooperative implies that this type of game models the process of players who are making choices thinking about their own interest.

The non-cooperative game theory relates to realistic cloud computing where the service providers do not form coalitions and a decision is made considering the selfish actions of individual providers who want to maximize their profit.

Games in Normal Form

The representation of a game in normal form or strategic form is given by Jackson (2011):

- The set of players $N = \{1 \ldots . n\}$.
- Player i has a set of actions a_i which are normally referred to as pure strategies.
- The set of all pure strategies is denoted by $a = (a_1 \ldots . a_n)$.
- Player i has a payoff represented as the function of the action vectors is denoted by $u_i: A \rightarrow I R$, where u_i (a) is i's payoff if a is the strategy taken.

Normal form games are often represented in the form of a table. The most common example is prisoner's dilemma represented in Figure 9.

In prisoner's dilemma, the number of players is two with each having two pure strategies, where $a_i = \{C, D\}$; C is for cooperate and D is for defect. C indicates the payoff to the row player (player 1) as a function of the pair of actions, while D is the payoff to the column player (player 2). The game is

Figure 9. A prisoner's dilemma game

		Player 2 C	Player 2 D
Player 1	C	-1, -1	-3, 0
	D	0, -3	-2, -2

explained as follows. Both the players are caught committing a crime and are investigated in different cells in a police station. The prosecutor comes to each of them and tells each of them:

If you provide a confession and accept to testify against the other; and if the partner doesn't confess; you will be set free. If both of you accept committing the crime, you will be both sent to jail for two years. If you do not accept committing the crime but your partner does, you will be given a sentence of three years imprisonment. If none of you confess; only one year punishment will be given due to the lack of evidence.

So the payoff matrix shows the imprisonment time in years. The term cooperate means that the partners cooperate with each other. The term defect means that you are accepting the crime and agreeing to testify, and so you are breaking the agreement which you two have.

Nash Equilibrium

If a set of strategies for the players constitute a Nash Equilibrium, it means that none of the players can benefit by altering his/her strategy unilaterally.

A strategy a_i is a best reply, also known as a best response, of player i to a set of strategies $a_{-i} \in a_{-i}$ for the other players if

$$u_i(a_i, a_{-i}) \geq u_i(a'_i, a_{-i}) \; ; \forall \; a'_i \tag{1}$$

A profile of strategies $a \in A$ is a pure strategy Nash equilibrium if a_i is a best reply to a_i for each i. That is, a is a Nash equilibrium if

$$u_i(a_i, a_{-i}) \geq u_i(a'_i, a_{-i}) \; ; \forall \; i, a'_i \tag{2}$$

A pure strategy Nash equilibrium only states that the action taken by each agent be the best against the actual equilibrium actions taken by the other players, and not necessarily against all possible actions of the other players.

In prisoner's dilemma, Nash Equilibrium occurs if both player 1 and 2 cooperate (C). If any of the players change his/her strategy, no player can benefit.

RELATED WORK

Dynamic VM Placement Problem

Hyser et al. (2008) studied the virtual machine placement scenario by developing an autonomic controller which dynamically maps the virtual machines onto the physical machines by following the users' policies. This work also differentiated the static VM placement and dynamic VM placement. The algorithms dealing with static VM placement start with no initial mapping. There isn't any need for these algorithms to consider the intermediate steps or the number of moves taken to reach the final state. This is in sharp contrast to the algorithms dealing with dynamic VM placement problem which must find optimal solutions from an initial mapping.

In 2007, Wood et al. (2007) presented a system called Sandpiper, which was used for automatic monitoring and detection of system hotspots. It also provided a new mapping of VMs onto physical hosts and initiated the migrations too. Algorithms for hotspot detection focus on signaling a need for migration of VMs whenever SLA violations are detected either implicitly or explicitly. In the same year, Bobroff et al. (2007) implemented an algorithm based on first-fit approximation to find solution to the problem of dynamic VM placement having the goal of price minimization. The problem was mapped as a bin packing one where the minimum number of PMs to be needed for the VMs was calculated and then a remapping was done for VMs onto PMs. The main disadvantage of the algorithm was that, it didn't look for unreachable solutions from the initial mapping.

A two phase process was developed by Hermenier et al. (2009) in order to find solutions to the dynamic VM placement problem. The consolidation manager named Entropy found solutions in two phases. The first phase finds out a placement keeping in mind the constraints, VM set and the CPU requirements. It also provides the likable configuration plan to achieve the desired mapping. The second phase tries to improve the result computed in the first phase. It takes into account a refined set of constraints and tries to minimize the number of migrations required. However very simple parameters are considered in this work and server consolidation is not taken as a factor.

Liao et al (2012) affirmed that dynamic VM placement problem faced three major challenges; multi-dimensional constraints, the initial state and the intermediary steps. They proposed a system called GreenMap which was a VM-based management framework to be able to execute live VM migrations considering the resource consumption of servers and energy consumption. Simulated annealing based heuristic is used for the optimization problem under the constraint of multi-dimensional resource consumption. The two objectives namely, reduction in energy consumption and performance degradation are balanced to give the desired output. But the GreenMap system doesn't address heterogeneous physical hosts which invariably form the real life server clusters.

Cooperative Game Theory

Wei et al. (2010) uses cooperative game theory in order to find solution of the QOS constrained problem of allocation of resources. Here problem solution is done in 2 steps. First optimal problem solution is done by every participant independently without resource multiplexing. Next a mechanism is designed by considering strategies which are multiplexed, of all the players. Existence of Nash Equilibrium is shown if resource allocation problem has feasible solutions. This paper creates a useful analytical tool for the solution of optimal scheduling problem.

No consideration for multi-tier architecture of web services and lack of emphasis on stable placement of applications are the drawbacks in Wei et al. (2010). These disadvantages are removed by Lee et al. (2010) where an evolutionary approach of game theory is designed for stable and adaptive placement of applications. This paper implements Nuage which uses an evolutionary game theoretic approach in order for stable adaptive deployment of applications. The main goal of this work is to deploy *N* applications on *M* hosts with the goal that applications adapt their locations and allocation of resources is done on the basis of resource and workload availability. In this work only CPU time share is considered for assignment of resources to each VM.

General colocation game and process colocation game were introduced by Londono et al. (2009) in order to distribute resources to infrastructure providers. This work considers a cooperative game theoretic framework where the objectives for resource management were to maximize resource utilization and minimize total cost of the allocated resources. The work also proves that achieving nash equilibrium is NP-complete. The main drawback of this work is that although it provides best response computation, it is an expensive framework.

Cooperative VM management for multi-organization environment in cloud computing was done by Niyato et al. (2011). Three types of resources are considered for VM management; private cloud, on-demand plans and reservation of public CSP. The algorithm works in two steps. First, an optimization model is formulated for cooperative organizations and then, an optimal VM allocation problem is solved in order to minimize the total cost. This work doesn't consider the stochastic nature of demand.

Resource allocation in Horizontal Dynamic Cloud Federation Platform (HDCF) environment is studied by Hassan et al. (2011).The work presents a cost effective and scalable solution to the resource allocation problem using game theory. It studies both cooperative and non-cooperative resource allocation games as well as both decentralized and centralized algorithms are presented in order to find optimal solutions. The major drawback in this framework is that it doesn't take into consideration the dynamic nature of clouds where hundreds of clouds leave and join the federation in a dynamic manner.

Mao et al. (2013) worked upon the problem of cloud service deployment by modeling it as a congestion game. Only cost and quality of resources were considered. Every service acts as a player which chooses a subset of resources for the maximization of his payoff. It is shown that Nash equilibrium is reached in polynomial time. This work can be further implemented in seeking solution to the placement of multiple cooperative heterogeneous components over many cloud services.

New resource allocation game models (CT-RAG, CS-RAG) were introduced for problem solving in cloud computing by Sun et al. (2014). Existence of Nash equilibrium is shown but the work considers static game with no representation of fairness of tasks. The problem of allocation of resources for PMs in cloud computing based on the uncertainty principle of game theory and coalition formulation was studied by Pillai et al. (2014). It is shown that the solution gives higher request satisfaction and better resource utilization.

Non-Cooperative Game Theory

Teng et al. (2010) introduced a novel Bayesian Nash Equilibrium allocation algorithm in order to find solution to the resource management problem in cloud computing. It analyzes the bid proportion

model where users are assigned resources in proportion to their bids. The allocation solutions reach Nash equilibrium by gambling in stages which is further simplified by considering that competition among users is for the same job type consisting of a sequential order of same type of jobs. In this model, the shortcoming is that bidders fix the resource price and problems of response delay are not addressed. In the same year, Sun et al. in (2010) used continuous double auction and Nash equilibrium for the allocation of resources in cloud based on the M/M/1 queueing system. Sun et al. (2010) took performance QOS and economic QOS as optimization objectives. The work in Sun et al. (2010) greatly improves the Quality of Service in terms of performance and economy. It is fair to both resources and users by considering execution cost of 'pay per use' services and the average time of execution of user jobs.

The problem of service provisioning was studied by Ardagna et al. (2011). IaaS providers host applications of SaaS providers such that every SaaS complies with the SLA (revenue and penalty) requirements. SaaS providers maximize revenues while cost of resources by IaaS is minimized. IaaS providers maximize revenue by providing virtualized resources. The following assumptions for SaaS providers are taken in this work.

- A single web service application is hosted on a single VM.
- Same web service application implemented by many VMs can run in parallel.
- All the virtual machines are uniform in terms of CPU and RAM capacity.

IaaS providers offer flat VMs, on-demand VMs and on-spot VMs for which SaaS providers pay. The service provisioning problem is formulated and solved using game theory.

The main drawbacks in this work are listed as follows.

- Validation of the solution is missing by experiments in real world environment.
- The heuristic solutions adopted by IaaS and SaaS providers are not compared.

These drawbacks in 2011 were worked upon by Ardagna et al. (2013).

Kong et al. (2011) worked upon resource allocation among selfish virtual machines by applying stochastic approximation methods. It thwarted non-cooperative behavior of the VMs. The proposed approach is not implemented in a real world virtualized cloud system. A cooperative game theoretic resource splitting solution in cloud computing was formulated by Lu et al. (2012). Here multiple cloud providers cooperate with one another in order to form a cloud federation which ensures greater profit since provider's capability to serve for public cloud users is enhanced. A game theoretic policy is formed to aid SPs in the decision making process. The work's performance is not gauged for many cloud environments.

The co-scheduling problem addressed by Dhillon et al. (2013) involves a central authority which provides solution to constrained optimization problem with an objective function. This works provides an alternate game theoretic perspective using stable matching theory which reduces the Stable Roommates Problem (SRP) to Stable Marriages Problem (SMP). Chen et al. (2014) studied the use of game theory for live migration prediction over cloud computing. It is shown that game theory improves Gilbert-Elliot model for the prediction of the probability on dirty page.

PROBLEM FORMULATION

Energy Consumption Model

The major part of energy consumption in a data center is produced by PMs which are running. There are other causes of energy consumption such as cooling apparatus but they form a much lesser percentage. Here we focus on the energy consumption or PMs which is divided into three sections.

Energy Consumption in Different States

Existing research indicates that energy consumption can be minimized by the adjustment of the states of PMs. Here 4 states of a PM are considered:

- *off* - This state consumes no energy.
- *ready* - PM is on but there are no active VMs on it. This state wastes energy consumption if PM is kept in this state for a long time. But if PM is switched off, much energy and time will be wasted to turn the PM back on to ready state.
- *idle* - This state exists as a result of a trade-off between off and ready states. PMs in this state consume lesser energy than PMs in ready state. Also lesser time and energy will be required to turn a PM on from idle state to ready state.
- *running* - This state consumes the maximum energy.

The states and the transitions allowed are shown in Figure 10.

It is observed that energy consumption varies according to load changes in a PM Xu et al. (2010). Thus energy consumption of a PM depends on the PM's utilization.

Thus the energy consumption for a PM can be modeled as:

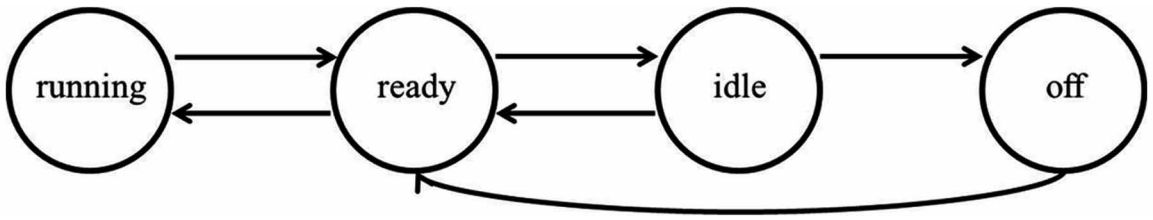
$$EC\ (t) = F\ E \times t + x \times L(t)^{y} \tag{3}$$

In equation 3, EC(t) represents the energy consumed in time period *t,* FE is the energy consumption that is fixed per unit time, L(t) is the load of PM in time *t*, and *x* and *y* are the energy adjusting coefficients.

The above equation can be modified for modeling energy consumption of the i^{th} PM in a particular state, since the fixed energy consumption of a PM in every state is different.

$$EC_{i}\ (t) = F\ E_{i}\ (P\ M\ S_{i}) \times t + x_{i} \times L_{i}\ (t)^{y}i \tag{4}$$

Figure 10. States and their transitions



Equation 4 calculates the energy consumption of PM_i in time period *t*. FE_i (PMS_i) is the fixed energy consumption per unit time of PM_i in state PMS_i. $L_i(t)$ is the load of PM_i for time period *t*, while x_i is the energy coefficient of the i^{th} PM and y_i is the relationship between load and energy of PM_i.

Energy Consumption During State Switch

Following rules are maintained while calculating energy consumption during switching of states of a PM.

- Energy consumption during switching between running and ready states can be ignored.
- State switching between off and ready states is slower than switches between idle and ready states.
- Switching off a PM costs lesser than keeping the PM in idle state.

Thus we use an array ECS(P X 4) to store the energy consumption of state switches since it is fixed for a particular state transition for a specific PM. The energy consumption during state switching of PM_i according to the array ECS can be seen in Figure 11.

Energy Consumption During Live VM Migrations

The consumption of energy during live virtual machine migrations are of three parts;

- Energy consumption by the source PM for preparation for migration.
- Energy consumption by the target PM for reception and rebuilding of the migrated VM.
- Energy consumption by other physical equipment during VM migration.

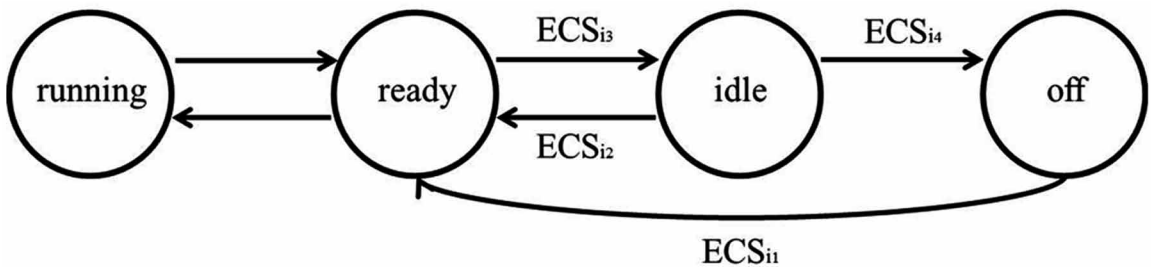
We maintain three arrays in order for energy consumption during live VM migration.

EVS (P X V) denotes the energy consumption by source PM during migration. EVS_{ij} is the energy consumed by PM_i for moving VM_j out.

EVT (P X V) denotes the energy consumption by target PMs for receiving the migrated VMs. Thus, EVT*ij* is the energy consumed by PM_i for receiving the migrated VM_j.

EV X (P X P X V) is a three dimensional array which denotes the energy consumption during migration. Thus EVX_{ijk} is the energy consumed during migration of VM_k from PM_i to PM_j.

Figure 11. Energy consumption of state switching



Thus the energy consumption during live VM migrations can be modeled as follows:

$$EVSij = \alpha i \times SZj + \beta i \tag{5}$$

$$EVTkj = \alpha k \times SZj + \beta k \tag{6}$$

$$^{EVX}kj = {}^{\gamma}ik \times {}^{SZ}j \tag{7}$$

where, α_i, β_i, α_k, β_k and $^{\gamma}_{ik}$ are energy adjusting coefficients and SZ_j is the size of VM_j

The Overall Energy Consumption Model

The energy consumption due to the switching of states during the entire migration process can be simplified. On comparison of the array *PMS* before and after the whole migration process, an array *PMST(4 X P)* can be derived, which represents the state switching of all PMs. Thus,

- $PMST_{1i} = 1$ means PM_i need state transition from off to ready.
- $PMST_{2i} = 1$ means PM_i need state transition from idle to ready.
- $PMST_{3i} = 1$ means PM_i need state transition from ready to idle.
- $PMST_{4i} = 1$ means PM_i need state transition from idle to off.

Thus the total energy consumption by state switching of PMs (EPS) for a particular solution (M) of the VM placement problem can be calculated as:

$$EPS(M) = \sum_{i=1}^{P} \sum_{k=1}^{4} (ECS_{ik} \times PMST_{ki}) \tag{8}$$

In order to calculate the total energy consumption during live migration, we create 3 arrays by comparing the array M before and after migration.

Array *VMLM* (P X P X V) is created to show the live migrations for all the VMs. Thus, $VMLM_{ikj} = 1$ means that VM_j's migration from PM_i to PM_k is needed.

Similarly 2 other arrays VMS(P X V) and VMT(P X V) are created to show the sources of migrations and their targets respectively. Thus $VMS_{ij} = 1$ means that VM_j is migrated from PM_i and $VMT_{kj} = 1$ means that VM_j is migrated to PM_k.

Thus the total energy consumption during migration of VMs (EVM) for a particular solution (M) can be calculated as:

$$EVM(M) = \sum_{i=1}^{P}\sum_{j=1}^{V}(EVS_{ij} \times VMS_{ij}) + \sum_{k=1}^{P}\sum_{j=1}^{V}(EVT_{kj} \times VMT_{kj}) + \sum_{i=1}^{P}\sum_{k=1}^{P}\sum_{j=1}^{V}(EVX_{ikj} \times VMLM_{ikj}) \quad (9)$$

The total energy consumption by all the PMs in different states (EPM) for a particular solution (M) can be given as:

$$EPM(M) = \sum_{i=1}^{P} EC_i(t) \quad (10)$$

Thus the total energy consumption by the VM placement problem (TEC) from initial solution M' to solution M can be denoted as:

$$TEC(M) = EPS(M) + EVM(M) + EPM(M) - EPM(M') \quad (11)$$

where, EPM(M') is the energy consumption of all PMs under initial solution M'.

System Model

With increasing dependence on Cloud Service Providers (CSP) by consumers for computing needs, there has been an enhanced requirement of a specific level of QoS which has to be followed by the CSPs. CSPs aim to meet the QoS parameters which are specified in the negotiated Service Level Agreement (SLA). Thus, an improved cloud architecture has to be developed which will cater to the needs of the consumers as well as the CSPs. The problem model that has been used in this chapter is depicted in Figure 12.

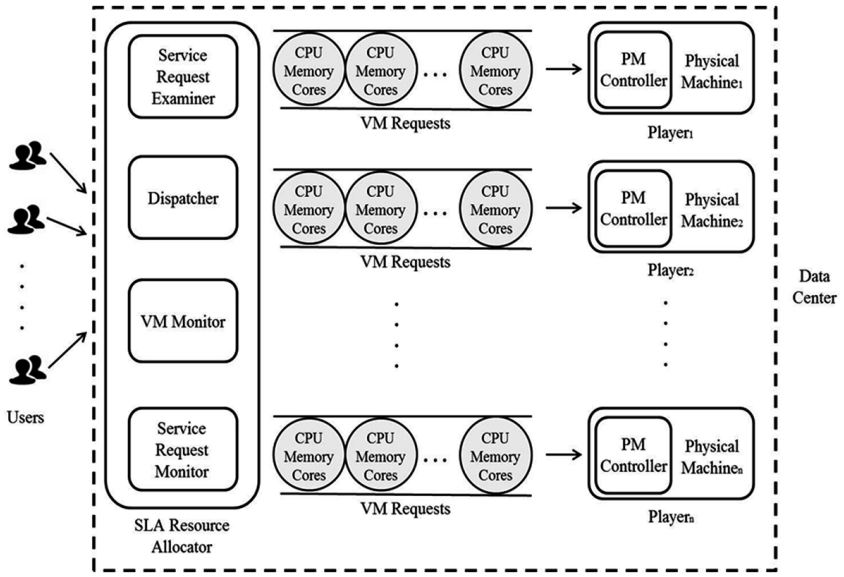
The cloud architecture being referred here has many entities.

- Users or brokers
- SLA Resource Allocator
- Service Request Examiner
- VM Monitor
- Service Request Monitor

VM requirements are given in the form of a vector $x.(\overrightarrow{cpu}) + y.(\overrightarrow{mem}) + z.(\overrightarrow{cores})$, where $\overrightarrow{cpu}$ is the CPU requirement in GHz, $\overrightarrow{mem}$ is the memory requirement in GB and $\overrightarrow{cores}$ is the requirement for the number of cores. x, y and z are the multiplying factors constraint to x + y + z = 1.

All Physical Machines (PMs) have a PM Controller which accepts the VM Request queue sent from the dispatcher. The PM Controllers are as players for the cooperative game theory which take the decision on the dynamic VM placement problem in order to minimize energy consumption.

Figure 12. Proposed system model



Problem Statement

Figure 12 refers to the system model which is used in order to find the solution to dynamic VM placement problem.

There are *n* Physical Machines (PMs). Every PM has a state associated with it. The states included are: 1 = off, 2 = idle, 3 = ready and 4 = running. Thus we use an array PMS (1 X P) to specify the current state of a PM. For every PM, there are 3 resources, namely CPU (GHz), Memory (GB) and number of cores. Thus we use an array PMR (3 X n) to specify the amount of resources that every PM has. Thus PMR_{ij} specifies the amount of $resource_i$ that PM_j has. The dynamic VM placement problem in this chapter is solved by Cooperative Game Theory where the players are the physical machines. Thus the resource status information is shared by all the PMs which aids in the problem solving process.

There are V_i number of VMs in PM_i. Let the maximum number of VMs present in one particular PM be maxV.

$$\max V = \max(V_i) \quad ; 1 \le i \le n \tag{12}$$

VM requirements are given in the form of a vector $x.(\overrightarrow{cpu}) + y.(\overrightarrow{mem}) + z.(\overrightarrow{cores})$.

Thus a 3-dimensional array VMR (3 X maxV X n) is created in order to forecast requirement of resources for VMs for a certain period of time. Thus VMR_{ijk} indicates the requirement of i^{th} resource by the j^{th} VM residing in k^{th} PM. For all the V VMs where $V = \sum_{i=1}^{n} V_i$, we require I number of performance indicators. In order to represent the performance requirements, we create an array VMP (I X V), where VMP_{ij} is the requirement of the i^{th} performance indicator for the j^{th} VM. This performance indicator acts as the SLA indicator.

The result of the virtual machine placement problem is stored in a mapping array M (P X V). M_{ij} can hold values 0 or 1. The value 1 indicates that j^{th} VM is placed at the i^{th} PM, while the value 0 shows that j^{th} VM is not running on the i^{th} PM. We also require an array MP (I X V) which indicates the level of performance of all the VMs. MP_{ij} means the j^{th} VM is performing at the i^{th} level for a particular solution to the VM placement problem.

The VM dynamic placement problem based on energy consumption (EC) (VDPPEC) can be formulated as follows:

$$VDPPEC = \min TEC(M) \tag{13}$$

where,

$$TEC(M) = EPS(M) + EVM(M) + EPM(M) - EPM(M') \tag{14}$$

such that,

$$\begin{aligned} &C_{ri} \le PMR_{ri} \\ &(C_{ri} = VMR \times M'; r = 1,2,\ldots,R; i = 1,2,\ldots,P) \end{aligned} \tag{15}$$

$$\sum_{i=1}^{P} M_{ij} = 1 \qquad (j = 1,2,\ldots,V) \tag{16}$$

$$MP_{iv} \ge VMP_{iv} \quad (i = 1,2,\ldots,I;\ v = 1,2,\ldots,V) \tag{17}$$

Equation 13 shows the optimization objective of the VM dynamic placement problem, i.e. minimization of energy consumption (EC) for a particular placement solution (M).

The solution is arrived upon satisfying all the three constraints as shown in equations 15, 16 and 17. Equation 15 is the capacity constraint. M' is the transposition of the mapping matrix (M). Placement constraint is represented in equation 16 and equation 17 shows that the SLA should be met in VM placement solution.

SOLUTION

Cooperative Game Theory

In this section, the algorithm based on congestion game theory for cooperative physical machines has been discussed which is used for solving the dynamic VM placement problem. In order for better convenience for description of the algorithm, the symbols along with their meanings have been listed in Table 1.

Table 1. List of symbols and their meanings

Symbols	Meanings
n	Number of physical machines
V	Number of virtual machines
$\rho\ n$	Initial mapping of V VMs onto n PMs
$\delta\ n$	Optimal mapping of V *VMs* onto n PMs after dynamic VM placement
δ^{α}_{p-1}	A strategy α for PM_{p-1}
ϑ	3 dimensional matrix (n * n * V) showing the live migrations to be executed. Thus, $\vartheta_{ikj} = 1$ means that *VMj's* migration from PM_i to PM_k is needed.
S	Strategy matrix to store the collection of strategies for all the PMs. Thus S_{ij} means the j^{th} strategy for i^{th} PM.
$payoff_i(\ \delta^{\alpha}_i\ )$	Total Energy Consumption if strategy α of PM_i is taken.

The algorithm is executed dynamically whenever any SLA constraint is violated. The status of all the physical machines is shared by all the players. Thus, it is cooperative game theory.

Algorithm 1 (Figure 13) is the calling module. It takes the initial mapping of V virtual machines onto *n* physical machines as input and produces the best strategy or the optimal mapping with minimum energy consumption as the output. Also the migrations needed to reach to the optimal solution are given as output.

The algorithm uses the congestion game model which models a resource set being shared by a group of players. Here every player selects a subset of resources from the resource set in order to maximize his own payoff; here maximizing the payoff means minimizing the energy consumption. Here, the physical machines are treated as players which select a subset of VMs in order to minimize the total consumption of energy. Thus algorithm 1 calls algorithm 2 for each player in order to reach Equilibrium at every step.

Every player, which is each Physical machine, acts as a player and attempts to achieve its own equilibrium by going through algorithm 2.

Algorithm 2 (Figure 14) finds the best strategy for the i^{th} PM. It takes as input the best strategy for (i-1) PMs and gives as output the best strategy of *i* PMs. First the i^{th} PM looks for the best strategy which will consume the least energy. Then all the (i-1) PMs change their strategies accordingly so that equilibrium

Figure 13. Algorithm 1: Equilibrium of 'n' players

Algorithm 1: Equilibrium for n players

Input: n Physical Machines, V Virtual Machines
Initial mapping for n players; $\rho^n = (\rho_0^n, \rho_1^n, \ldots\ldots, \rho_{n-1}^n)$
Result: Best strategy for n players; $\delta^n = (\delta_0^n, \delta_1^n, \ldots\ldots, \delta_{n-1}^n)$
Executable live migrations matrix; ϑ
for $(i = 0; i < n; i++)$ **do**
  Nash_Equilibrium(i);

Figure 14. Algorithm 2: Nash_Equilibrium(p)

Algorithm 2: Nash_Equilibrium(p)

Input: Best strategy for (p-1) Physical Machines
; $\delta^{p-1} = (\delta_0^{p-1}, \delta_1^{p-1}, \ldots\ldots, \delta_{p-2}^{p-1})$

Output: Best strategy for p players; $\delta^p = (\delta_0^p, \delta_1^p, \ldots\ldots, \delta_{p-1}^p)$

$\exists \delta_{p-1}^a \in S_{(p-1)j}$,
$payoff_{p-1}(\delta_{p-1}^a) \leq payoff_{p-1}(\delta_{p-1})$; $\forall \delta_{p-1} \in S_{(p-1)j}$
$\delta_{p-1}^p \leftarrow \delta_{p-1}^a$
$\delta_i^p \leftarrow \delta_i^{p-1}$, $i \in [0, p-2]$
$\delta^p = (\delta_0^p, \delta_1^p, \ldots\ldots, \delta_{p-2}^p, \delta_{p-1}^p)$
$j = 0$;
for $(i = 0; i < (p-1); i = j)$ **do**
  if $(payoff_i(\delta_i^p) \leq payoff_i(\delta_i)$; $\forall \delta_i \in S_{ij})$ **then**
    j++;
  else
    $\exists \delta_i^a \in S_{ij}$,
    $payoff_i(\delta_i^a) \leq payoff_i(\delta_i)$; $\forall \delta_i \in S_{ij}$
    $\delta_i^p \leftarrow \delta_i^a$
    $j = 0$;

is maintained, that is optimal energy consumption takes place. This algorithm thus produces the best strategy for every step for all the *n* physical machines.

There are *n* players (Physical Machines) and *V* virtual machines. So in each step in order to achieve Nash Equilibrium for a player the time complexity will be O(nV). The algorithm for finding out the equilibrium for a particular step has to be repeated for n players. Thus the time complexity to achieve Nash Equilibrium for *n* players will be $O(n^2V)$.

Thus this algorithm obtains the Nash Equilibrium in cooperative congestion game for dynamic VM placement at a particular time frame in polynomial time.

Six sets of experiments are performed and the results are compared with results of Best Fit algorithm. Best Fit Algorithm is a centralized algorithm which aims to place VMs into the PM with the least available space and with no SLA violations, such that the PMs are fully utilized. The steps involved in best fit procedure are defined in Algorithm 3 (Figure 15). The results are shown later.

Non-Cooperative Game Theory

In this section, the algorithm based on non-cooperative theory which has been used to solve the problem of dynamic virtual machine placement has been discussed. The symbols used in the algorithm have been listed in Table 2.

Algorithm 4 (Figure 16) shows the procedure for a PM to select its best placement every time such that in every iteration, the energy consumption by that physical machine is lesser than the energy consumption in the previous iteration for that particular PM. Thus, all the PMs act selfishly in order to optimally dynamically place VMs onto PMs such that energy consumption is minimized.

Practically speaking, in order to compute Nash Equilibrium, some coordination must be present among the players; in this case the physical machines. Thus, the non-cooperative procedure shown in algorithm 5 is devised where players (PMs) are synchronized in such a manner that their individual placement strategies are updated in a round-robin fashion.

Figure 15. Algorithm 3: Best Fit Algorithm

Algorithm 3: Best Fit

Input: p Physical Machines, n Virtual Machines
Result: Best Fit placement strategy for n Virtual Machines
for $(i = 0; i < n; i = i + 1)$ **do**
  Sort p PMs in increasing order of free space (remaining capacity)
  for $(j = 0; j < p; j = j + 1)$ **do**
    if $(Requirement(VM_i) \leq Capacity(PM_j)$ **then**
      Place VM_i into PM_j
      break;

Table 2. List of symbols and their meanings

Symbols	Meanings
i	Physical machine number
itr	Iteration number
$energy_i^{(itr)}$	Total energy of i^{th} PM computed at iteration number *itr*
τ	The accepted tolerance
$\delta p_{-1\alpha}$	A strategy α for PM_{p-1}
en	Tolerance at iteration number *itr*. $\sum_{i=1}^{n} \left\| energy_i^{(itr-1)} - energy_i^{(itr)} \right\|$
command	The instruction given to the neighbor PM. It has two values, *CEASE* and *CARRY_ON*.
SEND((a, itr, command), i)	Send message (a, itr, command) to PM_i
RECEIVE((a, itr, command), i)	Receive message (a, itr, command) from PM_i
VMLM_TEMP	Temporary live VM migration matrix.
M_TEMP	Temporary VM placement matrix.

Figure 16. Algorithm 4: Best_Placement(i)

Algorithm 4: Best_Placement(i)

Input: VMLM_TEMP, M_TEMP, itr
Output: $energy_i^{itr}$
while (1) **do**
  Find new placement strategy for PM_i;
  Calculate $energy_i^{itr}$
  if $(energy_i^{itr} \leq energy_i^{itr-1})$ **then**
    Update M_TEMP, VMLM_TEMP;
    return($energy_i^{itr}$);

Algorithm 5 (Figure 17) is executed for all the physical machines periodically or whenever SLA violations are made. When the execution of the algorithm is over, the players will have reached Nash Equilibrium, thus the strategies remain the same in equilibrium.

In this algorithm, the tolerance is maintained at a very small value, such that in every iteration, the energy consumption is reduced till a point that very small savings in energy is seen in consecutive iterations. This is the point where the algorithm exits. In order to facilitate the coordination among all the physical machines, messages are sent to the next physical machine. The message consists of three arguments; the energy savings made in that particular iteration, the iteration number and the command. If the command is CARRY_ON, it means that equilibrium has not been reached and best placement strategy needs to be computed again. On the other hand, the command CEASE instructs the next physical machine that it can exit the algorithm as equilibrium is reached. In every iteration, the first PM checks the tolerance for the last iteration and accordingly initiates the command sequence for all the PMs.

SIMULATION AND RESULTS

The simulation has been done using an in house simulation using Java.

Figure 17. Algorithm 5: Non-Cooperative Algorithm

Algorithm 5: Non-Cooperative Algorithm

PM i; (i=1, 2, ..., n) executes:

1. Initial :
- itr ← 0;
- $energy_i^{itr} \leftarrow 0$;
- en ← 1;
- sum_energy ← 0;
- command ← $CARRY\_ON$;
- prev = [(i-2) mod n] + 1;
- next = [i mod n] + 1;

2. while (1) do
- **if** *(i = 1) // First PM* **then**
  - prev = n;
  - **if** $(itr \neq 0)$ **then**
    - RECEIVE((en, itr, command), prev);
    - **if** $(en \leq \tau)$ **then**
      - SEND((en, itr, CEASE), next);
      - exit;
  - sum_energy ← 0;
  - itr ← $itr + 1$;
- **else**
  - // Other PMs
  - RECEIVE((sum_energy, itr, command), prev);
  - **if** *(command = CEASE)* **then**
    - **if** $(i \neq n)$ **then**
      - SEND((sum_energy, itr, CEASE), next);
    - exit;
- $energy_i^{itr}$ = Best_Placement(VMLM_TEMP, M_TEMP, itr);
- sum_energy = sum_energy + $\left| energy_i^{(itr-1)} - energy_i^{(itr)} \right|$;
- SEND ((sum_energy, itr, CARRY_ON), next);

Simulation Parameters

The series of experiments have been performed considering the capacities of physical machines as each having 12.8 GHz of processor speed, 8 GB memory and is octa-cored. The capacities of VMs are chosen randomly from a fixed set of values.

The values of x_i and y_i are determined via function fitting graph related to CPU utilization and are fixed at 61.06 and 2 respectively. Thus Equation 4 which calculates the consumption of energy by PMs in different states can be written as:

$$EC_i(t) = F E_i(P M S_i) \times t + 61.06 \times L_i(t)^2 \quad (18)$$

It is considered in the experiments that all the PMs are homogeneous in nature. Thus the parameters' values are same for all the PMs.

In order to calculate the consumption of energy by every PM, we use the energy consumption model described in chapter 2. The values for the parameters are shown in Table 3. To determine the energy consumption during live VM migration, the parameters have their values fixed; $\alpha_i = 0.256$, $\beta_i = 10.0825$, $\alpha_k = 0.256$, $\beta_k = 10.0825$ and $\gamma_{ik} = 0.5$.

Experimental Results

The Convergence of Nash Equilibrium

In order to prove that the non-cooperative algorithm converges to the nash equilibrium, we have performed experiments; the results of which is shown in Figure 18 and Table 4. A system of 20 PMs and 50 VMs is considered.

It is observed that the value of en converges to zero as the number of iterations increases. This proves that we are achieving Nash Equilibrium in polynomial time. From this experiment, for further simulations, we fix the value of τ as 0.003.

Figure 19 shows the variation of the number of iterations needed to achieve a value for sum_energy less than τ (0.003) as the number of PMs increase. For this simulation, the number of VMs has been fixed at 50.

Next, we perform a set of experiments keeping the number of PMs available fixed at 40 and increasing the number of VMs from 10 to 150. All the results obtained from the non-cooperative game theoretic algorithm are compared with the results obtained from cooperative game theoretic algorithm and best fit algorithm. The impact of selfishness is seen in the results.

Table 3. Values for the energy consumption parameters

States	Running	Ready	Idle	Off
Running	$FE_i(\text{running}) = 32.2708$	-	-	-
Ready	-	$FE_i(\text{running}) = 32.2708$	$ECS_{i3} = 3.5$	-
Idle	-	$ECS_{i2} = 7.5$	$FE_i(\text{idle}) = 0$	$ECS_{i4} = 5.5$
Off	-	$ECS_{i1} = 0$	-	$FE_i(\text{off}) = 0$

Figure 18. en vs No. of Iterations

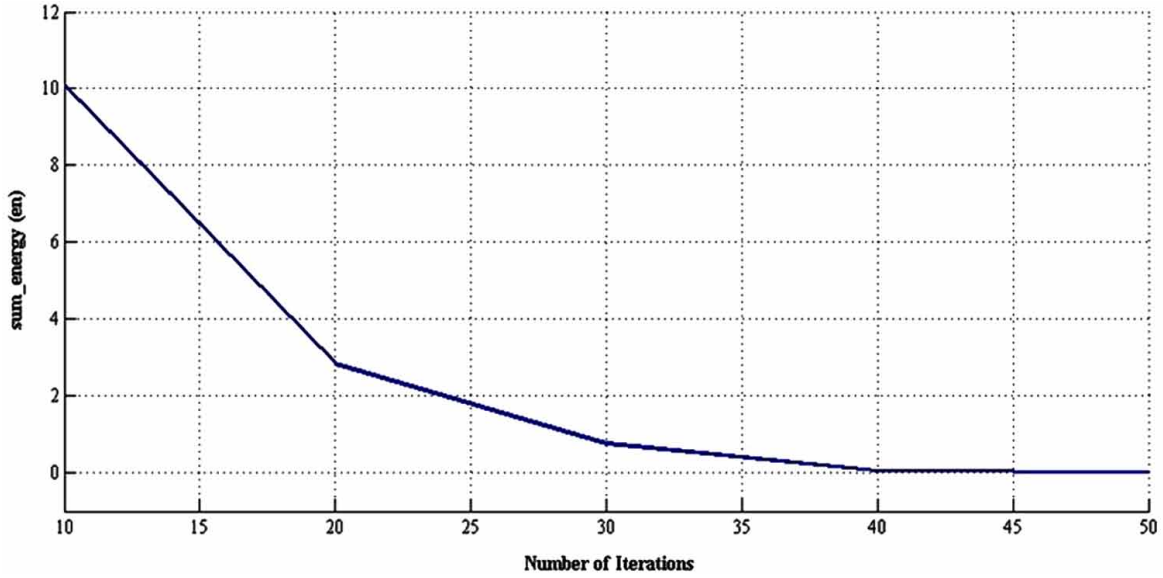
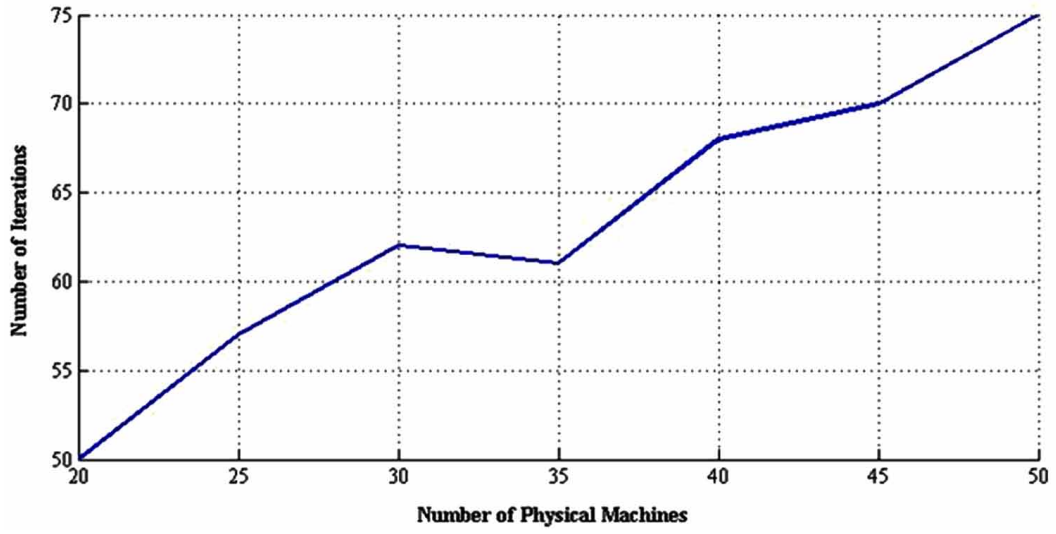


Table 4. sum energy (en) vs No. of Iterations (for 20 PMs, 50 VMs)

Sum Energy (en)	No. of Iterations
10.09	10
2.84	20
0.76	30
0.043	40
0.0027	50

Figure 19. Convergence of non-cooperative algorithm (until en $\leq$ 0.003)



The number of PMs used in order to dynamically place the VMs are projected in Figure 20. The results showing the energy consumption and execution time are shown in Figures 21 and 22 respectively.

From the results, it is observed that number of PMs used and the energy consumption for the non-cooperative game theoretic algorithm are similar to the results obtained from the cooperative game theoretic approach. Both these algorithms give better results than the best fit approach. But due to the non-cooperative nature of the players, that is PMs, the execution time increases for the non-cooperative approach. But the increase in execution time is not drastic so as to discard the algorithm. For minimization of the energy consumption, a slight increase in the execution time can be afforded.

Figure 20. No. of PMs Used vs No. of VMs (Available PMs Fixed at 40)

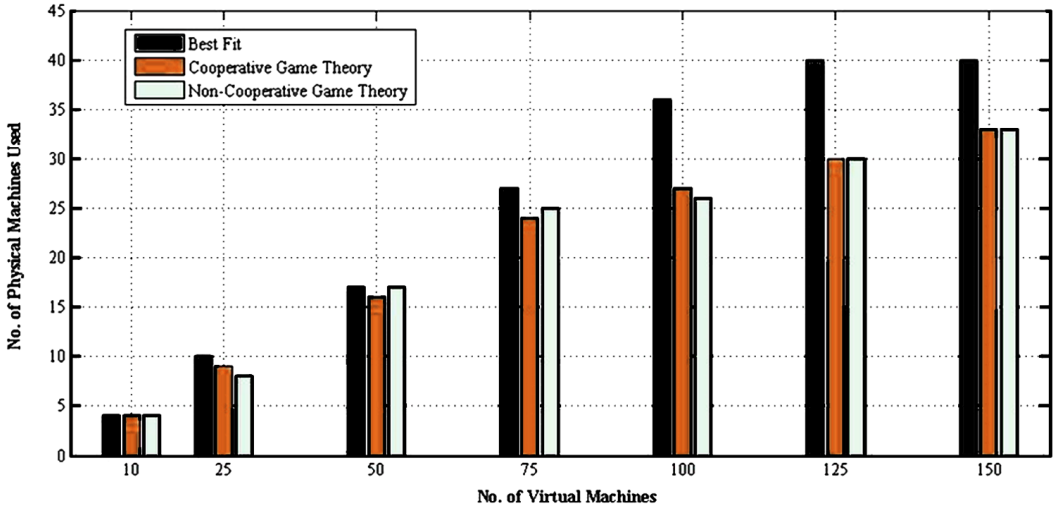


Figure 21. Energy Consumption vs No. of VMs (Available PMs Fixed at 40)

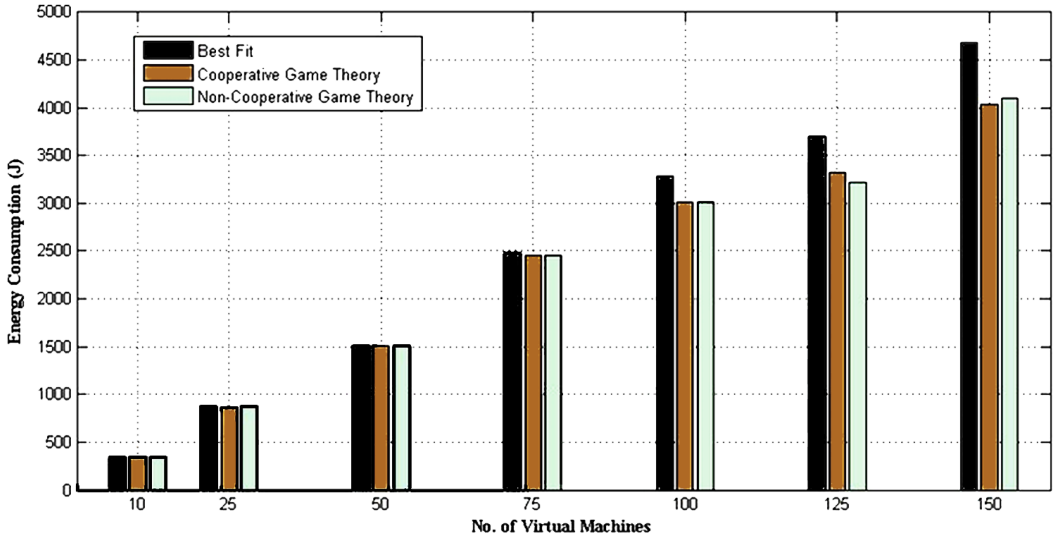
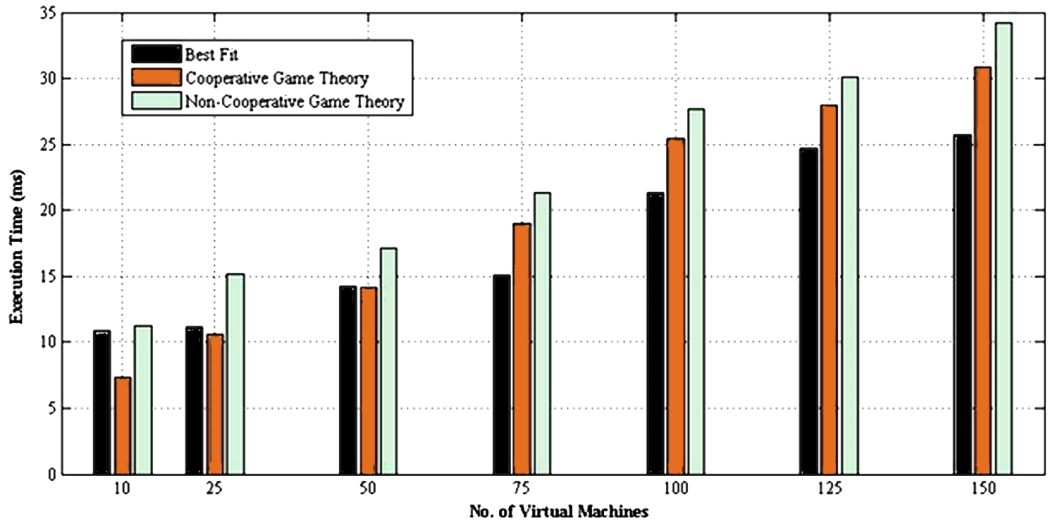


Figure 22. Execution Time vs No. of VMs (Available PMs Fixed at 40)



The next set of experiments are performed for a number of VMs fixed at 100 while the number of available PMs are increased from 35 to 95 in equal intervals of 15. The results of the non-cooperative approach along with the comparison with the cooperative procedure and the best fit algorithm are shown in Figures 23, 24 and 25.

Here too, the results obtained from the non-cooperative algorithm are similar to the results of cooperative approach which are better than the best fit procedure's results. But the minimization of energy consumption via non-cooperative approach comes at a cost of increased execution time.

Figure 23. No. of PMs used vs No. of Available PMs (VMs Fixed at 100)

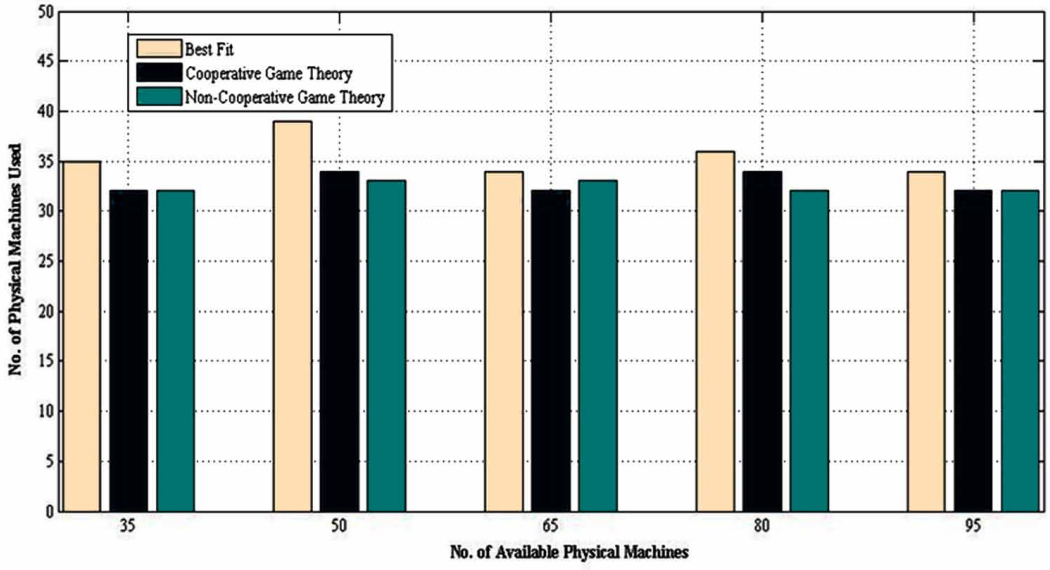


Figure 24. Energy consumption vs No. of available PMs (VMs Fixed at 100)

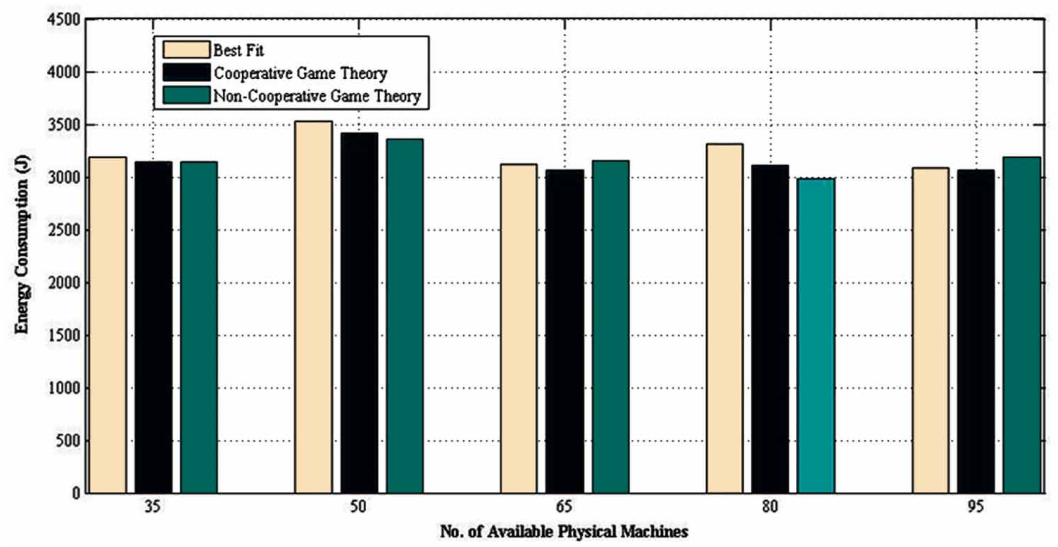
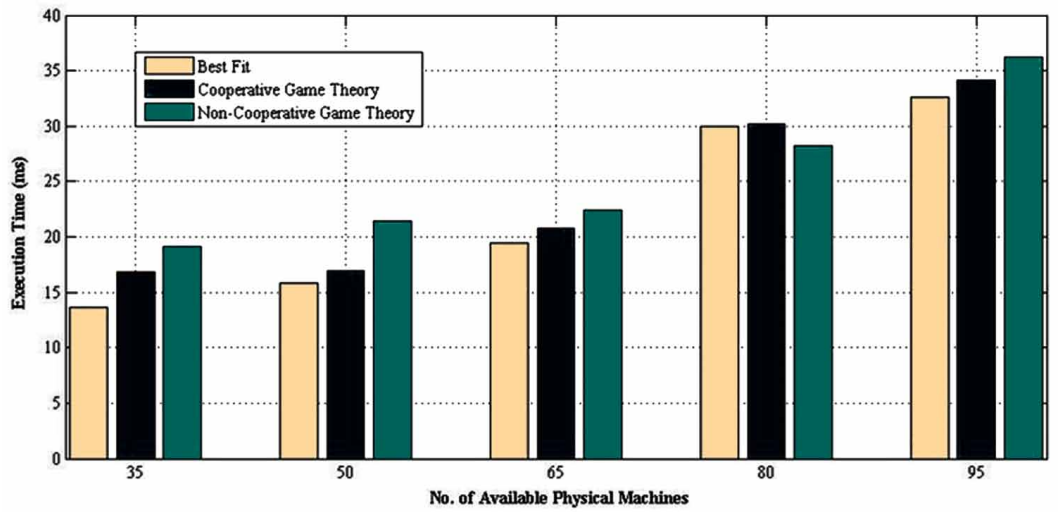


Figure 25. Execution time vs No. of available PMs (VMs Fixed at 100)



Thus the impact of the non-cooperative nature of physical machines affects the execution time to achieve optimal placement of VMs onto PMs while minimizing the energy consumption.

FUTURE RESEARCH DIRECTIONS

While conducting the simulations, fixed values are taken for calculating the energy consumption of physical machines. Also it is assumed that all the physical machines are homogeneous in nature. For

further research, the values of the parameters for calculation of energy consumption should be computed dynamically. Also heterogeneous environment should be considered. In addition to these, energy consumption due to other factors like cooling equipment which have not been considered in this work should be taken into account in future.

CONCLUSION

In this chapter, the problem of dynamic virtual machine placement has been addressed. After a deep analysis of the already existing research work in this area, a decentralized approach using game theory has been proposed here in order to optimize the consumption of energy. First, an energy consumption model is built in view of the complicated process of dynamic placement of virtual machines. Here, three factors have been considered while building the model, namely energy consumption of physical machines in different states, consumption of energy during the state transition of physical machines and during live virtual machine migrations. In this work, both cooperative and selfish nature of physical machines has been considered. Cooperative game theory using the concept of congestion game and non-cooperative game theoretic approaches have been used to propose new algorithms to optimally place virtual machines onto physical machines dynamically. For both the approaches, it is seen that Nash equilibrium is achieved in polynomial time. Both the algorithms guarantee a list of live virtual machine migrations to achieve the target solution from the initial mapping. Simulations are done and the experimental results are compared with the best fit approach. Both cooperative and non-cooperative algorithms help in minimizing the energy consumption, though the non-cooperative game theoretic approach takes a longer execution time to give the optimal result.

REFERENCES

Ardagna, D., Panicucci, B., & Passacantando, M. (2011). A game theoretic formulation of the service provisioning problem in cloud systems. In *Proceedings of the 20th ACM international conference on world wide web*. doi:10.1145/1963405.1963433

Ardagna, D., Panicucci, B., & Passacantando, M. (2013). Generalized nash equilibria for the service provisioning problem in cloud systems. IEEE Transactions on Services Computing, 6(4), 429–442.

Bobroff, N., Kochut, A., & Beaty, K. (2007). Dynamic placement of virtual machines for managing sla violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management, IM'07*. doi:10.1109/INM.2007.374776

Chen, Y.-L., Yang, Y.-C., & Lee, W.-T. (2014). The study of using game theory for live migration prediction over cloud computing. Springer Intelligent Data analysis and its Applications, 2, 417–425. doi:10.1007/978-3-319-07773-4_41

Dhillon, J. S., Purini, S., & Kashyap, S. (2013). Virtual machine coscheduling: A game theoretic approach. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC)*, (pp. 227–234).

Hassan, M. M., Song, B., & Huh, E.-N. (2011). Game-based distributed resource allocation in horizontal dynamic cloud federation platform. In *Algorithms and Architectures for Parallel Processing* (pp. 194–205). Springer. doi:10.1007/978-3-642-24650-0_17

Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., & Lawall, J. (2009). Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. doi:10.1145/1508293.1508300

Hotz, H., (2006). *A short introduction to game theory.* Academic Press.

Hyser, C., Mckee, B., Gardner, R., and Watson, B. J., (2008). *Autonomic virtual machine placement in the data center.* Academic Press.

Jackson, M. O., (2011). *A brief introduction to the basics of game theory.* Academic Press.

Jones, M. T. (2010). *Anatomy of an open source cloud.* Retrieved from http://www.ibm.com/developerworks/opensource/library/oscloud-anatomy

Kong, Z., Xu, C.-Z., & Guo, M. (2011). Mechanism design for stochastic virtual resource allocation in non-cooperative cloud systems. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*. doi:10.1109/CLOUD.2011.82

Lee, C., Suzuki, J., Vasilakos, A., Yamamoto, Y., & Oba, K. (2010). An evolutionary game theoretic approach to adaptive and stable application deployment in clouds. In *Proceedings of the 2nd workshop on Bio-inspired algorithms for distributed systems*. doi:10.1145/1809018.1809025

Liao, X., Jin, H., & Liu, H. (2012). Towards a green cluster through dynamic remapping of virtual machines. *Future Generation Computer Systems*, *28*(2), 469–477. doi:10.1016/j.future.2011.04.013

Londono, J., Bestavros, A., & Teng, S. H. (2009). *Collocation games and their application to distributed resource management. Tech. Rep*. Boston University Computer Science Department.

Lu, Z., Wen, X., & Sun, Y. (2012). A game theory based resource sharing scheme in cloud computing environment. In *2012 IEEE World Congress on Information and Communication Technologies (WICT)*. doi:10.1109/WICT.2012.6409239

Mao, Z., Yang, J., Shang, Y., Liu, C., & Chen, J. (2013). A game theory of cloud service deployment. In *2013 IEEE Ninth World Congress on Services*. doi:10.1109/SERVICES.2013.35

Milchtaich, I. (1996). Congestion games with player-specific payoff functions. *Games and Economic Behavior*, *13*(1), 111–124. doi:10.1006/game.1996.0027

Niyato, D., Zhu, K., & Wang, P. (2011). Cooperative virtual machine management for multi-organization cloud computing environment. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*.

Sun, D., Chang, G., Wang, C., Xiong, Y., & Wang, X. (2010). Efficient nash equilibrium based cloud resource allocation by using a continuous double auction. In *2010 International Conference on Computer Design and Applications (ICCDA)*.

Sun, W., Zhang, D., Zhang, N., Zhang, Q., & Qiu, T. (2014). Group participation game strategy for resource allocation in cloud computing. In *Network and Parallel Computing* (pp. 294–305). Springer. doi:10.1007/978-3-662-44917-2_25

Teng, F., & Magoules, F. (2010). A new game theoretical resource allocation algorithm for cloud computing. In *Advances in Grid and Pervasive Computing* (pp. 321–330). Springer. doi:10.1007/978-3-642-13067-0_35

Turocy, T. L., & Von Stengel, B. (2001). *Game theory: Draft prepared for the encyclopedia of information systems.* Tech. Rep. CDAM Research Report LSE-CDAM-2001-09.

Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, *54*(2), 252–269. doi:10.1007/s11227-009-0318-1

Wood, T., Shenoy, P. J., Venkataramani, A., & Yousif, M. S. (2007). *Black-box and gray-box strategies for virtual machine migration* (Vol. 7, pp. 17–17). NSDI.

Xhafa, F., & Kolodziej, J. (2010). Game-theoretic, market and meta-heuristics approaches for modelling scheduling and resource allocation in grid systems. In *2010 IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. doi:10.1109/3PGCIC.2010.39

Xu, J., & Fortes, J. A. (2010). Multi-objective virtual machine placement in virtualized data center environments. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom) & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. doi:10.1109/GreenCom-CPSCom.2010.137

KEY TERMS AND DEFINITIONS

Cloud Computing: Cloud computing is a technique for helping users to have access to computing resources on demand with very minimal effort in management.

Cooperative Game Theory: Cooperative game as the game scenario where players form coalitions in advance to discuss their actions.

Game Theory: Game theory is the formal study of cooperation and conflict. A game is described as a set of players and their possibilities to play the game by following some rules (strategies).

Non-Cooperative Game Theory: This type of game models the process of players who are making choices thinking about their own interest.

Virtualization: Virtualization is the capacity to run many instances of operating system in a single PM thus making maximum use of the hardware capacities of the physical machine which helps immensely in saving money for energy and hardware costs.

Virtual Machines: These individual instances of operating system are called Virtual Machines (VM).

Virtual Machine Placement: The scheduling of virtual machines in data centers and virtual machine scheduling on physical machines.