Research Journal of Applied Sciences, Engineering and Technology 10(3): 274-287, 2015
DOI: 10.19026/rjaset.10.2488
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: December 14, 2014 Accepted: January 27, 2015 Published: May 30, 2015

Research Article

Optimization of Virtual Machine Placement in Cloud Environment Using Genetic Algorithm

N. Janani, R.D. Shiva Jegan and P. Prakash
Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India

Abstract: The current trend in the computing era is cloud computing, which helps in providing seamless service to the user, but optimizing the utilization of the available resources and an efficient placement of the virtual machines are not so significant in the existing phenomena. Virtual machines are software computers that act as the key feature in providing services to the existing physical machine. VM placement is the process of mapping the virtual machine requests or images to the physical machines, according to the availability of resources in these hosts. Hence in this study, we have studied on various methods in which the placement are being done and have proposed an idea which, treats the available pool of physical resources as each knapsacks, which are solved using genetic algorithm, to get an optimal placement. Starting with this aspect, we enhanced the solution by considering multiple and multidimensional parameters in the virtual machine request, so that the migration of the virtual machines will be reduced and hence power-saving.

Keywords: Bin packing, cloud computing, crossover, genetic algorithm, knapsack problem, migration, mutation, virtual machine placement, VM placement algorithms

INTRODUCTION

Cloud is the most prominent technology that drives the industry now-a-days. Cloud is the back bone of most of the technically advanced organizations because it provides ample services to its users. Cloud can provide any type of services, depending upon the requirement. It is an on-demand self-service that allows a variety of users to access a wide range of service. It is a pay-as-you-go type of a technology that charges users according to the amount of service consumed. Cloud has taken a firm place in this short span of time because of its scalability, that is, it can easily add up or shrink down the services depending upon the customer necessities.

The cloud service consumers or the clients may be heterogeneous, in the sense; they can be thick or thin clients. All that they want is service, from the cloud. The service may be usage of resources for a particular time, for which they will be metered, monitored and charged for. These resources are a pool of physical and virtual resources which are available anywhere, anytime on cloud. Everything in cloud happens dynamically. Not that a single machine is allotted with a particular resource for ever, as in traditional client server models. Any system can access any virtual machine, which is randomly allotted by the cloud and access service.

Hence it is obvious that to access cloud services, we need a virtual machine and to get a VM mapped into a physical host or a node, there must be a proper VM placement algorithm to ensure that the available resources are not wasted and are allocated and distributed across the hosts in an optimized manner, such that the migration will be reduced, there will be less overhead and also there is an decent amount of contribution to minimize the power utilization. So, we try to deploy the concepts genetic algorithm and multiple multidimensional problem over our placement problem and thereby improvise the placement of the virtual machines.

Services provided by cloud: All the service provided by cloud is completely virtualized, as discussed by Mell and Grance (2011). The cloud service consumer can only access a resource, use it and store data in his desired format, but can't own any of the resource. No one will know, from which virtual machine which user has accessed which virtual resource, or where it has been stored. The detailed categories of the services provided by cloud are discussed below.

SAAS: Software as a Service is one of the service provided by cloud. A typical example to understand SAAS is, a user has setup Windows operating system in

Corresponding Author: N. Janani, Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India

his personal device and he is forced to work with Ubuntu just for a day. He need not reconfigure his operating system or setup a new virtual operating system on the top of the existing one. He can access cloud and enjoy his work in Ubuntu, without disturbing the existing environment.

PAAS: Platform as a Service is yet another service provided by cloud. Consider a scenario where in the user is working on a windows operating system but needs an entirely new platform to build up his project just for a week, which costs more for setup and installment. Cloud can help this user by providing the platform that he wants, just for a week, on a rent basis. Since it provides PAAS.

IAAS: Infrastructure as a Service is where in cloud provides an infrastructure itself as a service to the user. For example, on the date of conducting exams where about more than 5000 candidates participate, the exam board may expect a higher bandwidth of network than the usual. During those times cloud can provide IAAS. Not only data centers can be said an infrastructure; having a normal personal digital assistant one can work with super computers too, with the help of cloud.

Without cloud computing, all these cases would have met unwise solutions either to waste the resources or reconfigure the whole environment for a short time show. But cloud comforts with all these services which are purely dedicated for the users, they may feel that they own the service, or the system or the application or the platform, or the infrastructure. But in real, these services are just given to the users based on their request and they cannot own any, says Shiva Jegan *et al*. (2014). The fact is all these are virtualized!

Virtual machines: A virtual machine is a self-contained operating environment, which acts like a separate computer; that allows running any operating system or application with the existing system. To understand easy, a virtual machine can be said a software computer. Cloud, to serve the users with varying requests needs VM's aid. These VMs are placed based on various distribution methods, which help in granting the user requests with a good quality of service. Virtual machines are so dynamic since cloud is highly scalable; it needs to be so quick and smart. As discussed by Creasy (1981), the objectives that are to be carried for an efficient virtual machine distribution are:

- To avoid congested links between the virtual machine residing system and the request raising system.
- Communication time between the VM residing system and the node must be reduced whereas data exchange should be maximized. That is more data must be transferred properly within short span of time.
- An integrated solution is needed to reduce transmission overhead and allow easy VM image update, in case of there are any updates in the VM image distribution.

MATERIALS AND METHODS

Virtual machine placement methods: The process of mapping virtual machines to physical machines is called as virtual machine placement by ensuring to improve power efficiency and resource utilization. Virtual machines must be distributed in an efficient way such that no system or a request starves for the response from cloud. Each system on the cloud network will be distributed with a VM image. The primary goals in placing a virtual machine are:

- Maximize the usage of the available resources.
- Save power by shutting down servers.

If VMs are allowed to migrate from one physical machine to another, then it shows that it is a dynamic placement. If no migration between systems, then it is said to be a static placement of VM.

In our case, we address dynamic placement of the virtual machines. Some of the virtual machine placement algorithms are being discussed below, followed by the drawbacks in using them, as said by Pisinger (1995). Followed by those implementation issues, the solution to this problem is drawn below.

First fit algorithm: This algorithm is an optimal algorithm which can be used locally to place virtual machines. This algorithm works in a greedy manner to optimize the placement process. When a virtual machine request is triggered, then this algorithm responds by allocating the first physical machine available in the queue, provided that the first machine is fully loaded with required resources. Also this algorithm does not waste time in scanning the partially loaded physical machines, as discussed in Gottlieb (2000). Hence reduces the time complexity. Consider the example where in each box is a VM request and the numbers inside them are the resource required for each virtual machine request.

In the scenario dealt in Fig. 1, there are six physical machines that hold maximum ten resources each. According to this algorithm, the first VM request that comes will get place in the first physical machine. This will be followed by the second VM request which first checks the host 1, if it can give place for execution. If not it goes to the second host. This procedure follows for the entire queue. And finally we can see that, the last two hosts are left free.

First fit decreasing algorithm: This algorithm works similar to the first fit algorithm except the fact that, the virtual machines will be sorted in decreasing order of resources. Once when a VM request arrives, then the first fit algorithm will be applied. This method is more

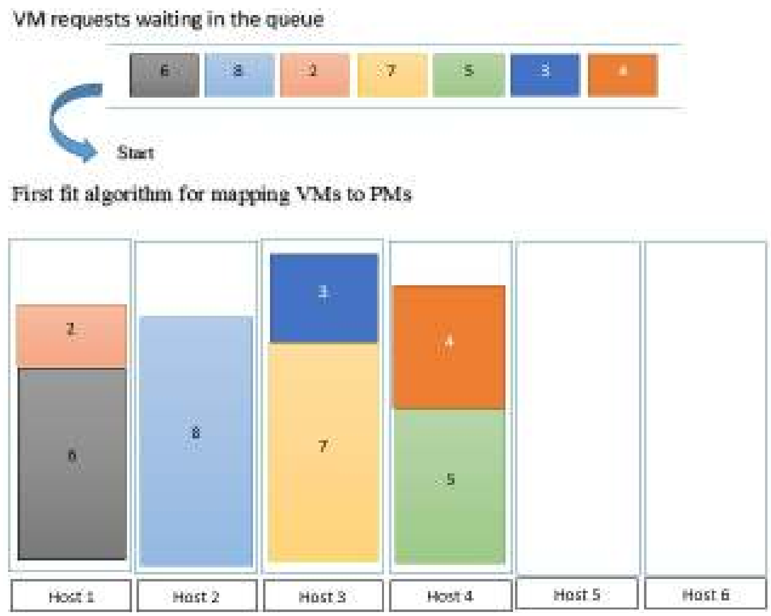


Fig. 1: Mapping VMs to PMs using first fit algorithm

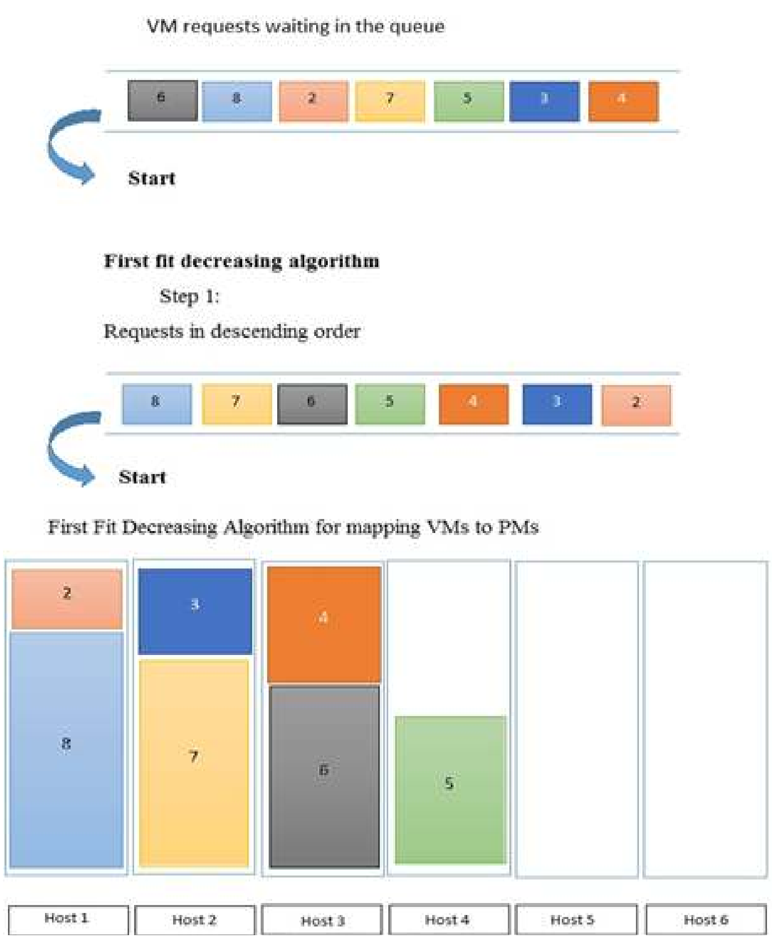


Fig. 2: Mapping VMs to PMs using first fit decreasing algorithm

efficient compared to first fit algorithm since it optimizes the memory allocation. Consider the same example to be solved using first fit decreasing algorithm.

Consider each host can accommodate till 10 units as in Fig. 2. While the placement was by first fit, only one node was fully utilized, but by this method three nodes are fully utilized so that if any new request comes it can occupy the next. Thus it optimizes the allocation. The overall result, in optimizing the resource allocation may seem the same; but the ways in which the VMs are mapped to the hosts vary widely.

Next fit algorithm: Next fit algorithm follows a technique by providing an easier way to allocate resources by making use of a variable called "NEXT" which is initially assigned to null. When a virtual machine request is placed, the search begins from the first node, if the physical machine contains the requirements of the virtual machine; it will be allotted else a new PM will be started. The variable NEXT will go to the next position in the queue, once when a VM is mapped into any of the available PM. This algorithm keeps count on how many PMs are used. Also this algorithm does not look into an already allotted PM. That is the variable NEXT will never go in reverse.

Consider the example in Fig. 3, the maximum capacity that each host can accommodate is 10 units. In first fit method, even after allocating some VM to the host, for the next VM request the already

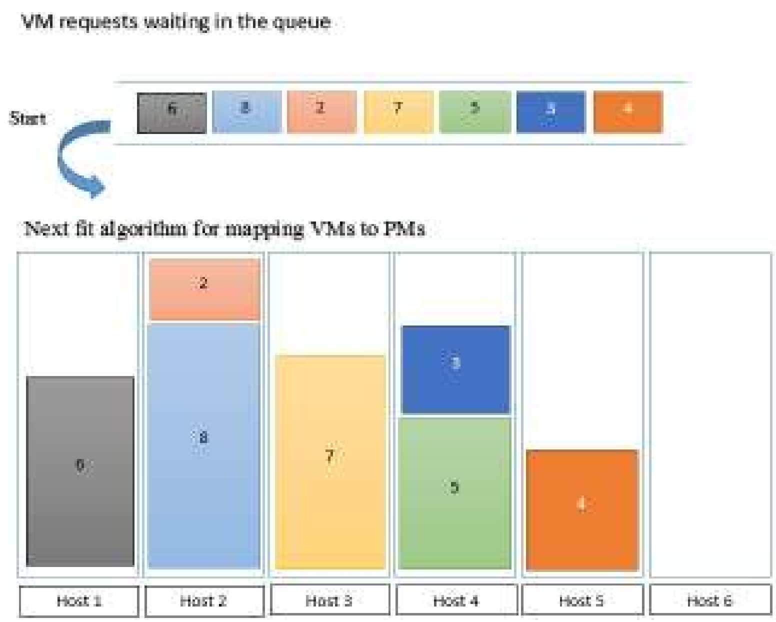


Fig. 3: Mapping VMs to PMs using next fit algorithm

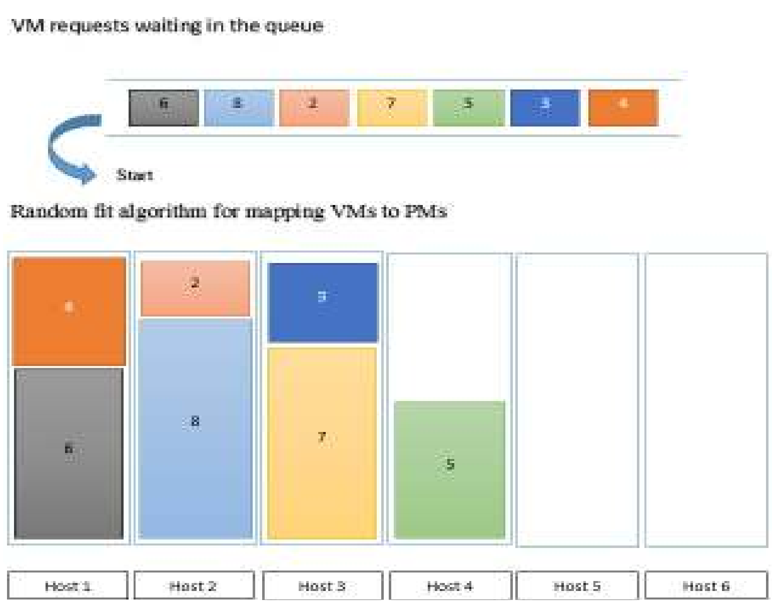


Fig. 4: Mapping VMs to PMs using random fit algorithm

allocated node will also be checked for availability. But in this method, resources once allocated will not be checked again. This optimizes time, but kills memory since all the five hosts that are assigned to VM are underutilized.

Random fit algorithm: Random fit algorithm, as the name indicates this algorithm randomly allocates virtual machine requests to physical machines just by ensuring the resource availability in the physical machine. This algorithm cannot be so successful in all scenarios, since both VMs and PMs are picked at a random, so no proper logic or trace over can be given.

Consider each node can handle 10 units of requests in Fig. 4. Even in this scenario, two physical machines are neither loaded and the placements in the working hosts are also to the maximum. But one cannot judge how the placement is done. Also, it is not always possible to predict or assure that it will give an optimum result. This algorithm can work in the reverse fashion too; making use of all the six physical machines available.

Most full algorithm: This algorithm works on a pretty simple logic that all the physical machines which already allow some VM requests to run on it, will be sorted in ascending order. After this sorting first fit algorithm will take place. Most full algorithm is just an improved version of first fit algorithm.

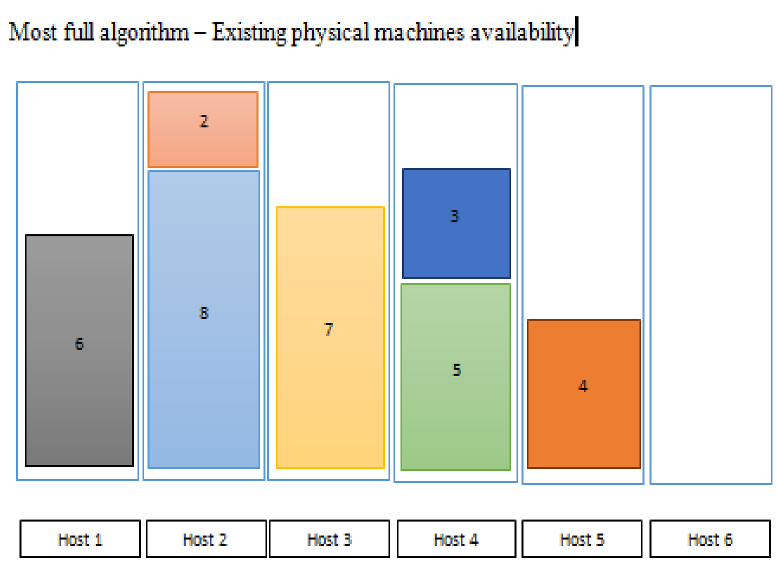
Initially the available PMs are giving place for some VMs. To apply this algorithm, these physical machines must be sorted in descending order. After that the VMs should be sorted in ascending order, after that mapping the VMs to the PMs will take place. In this example, the black color will specify the previous existing physical availability and the colored represent the new sorted VMs which is placed in the existing PMs. Consider each node can handle 10 requests each.

Figure 5a is the example scenario in which most full algorithm is going to be deployed. Available physical machines are sorted in descending in the Fig. 5b, the placement is shown in Fig. 5c.

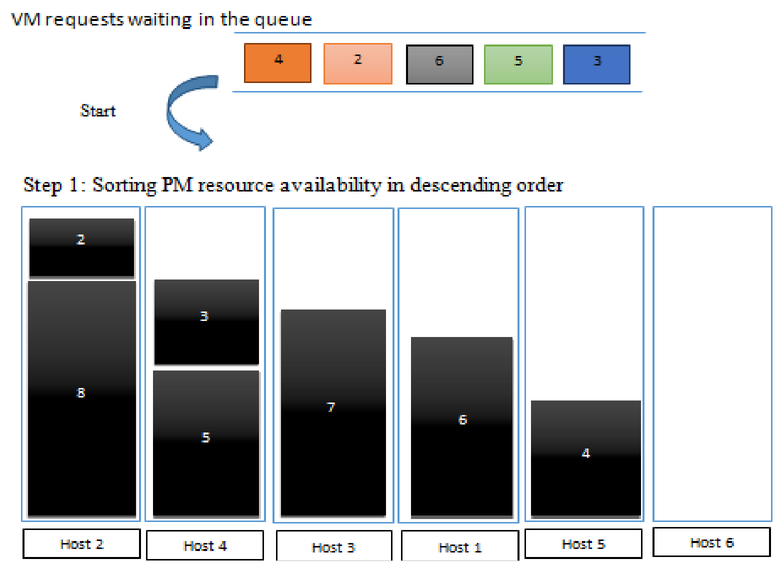
The drawbacks in placing a virtual machine: The VM images are allocated to systems based on any of the criteria depending upon the scenario. The loop hole here is that, what if any of the allotted VM is under-utilized or over utilized. If a VM is idle for a reasonable amount of time then it must be migrated to another system which actually needs it. Also certain VMs get more and more request that its' handling capacity. Here some idle VMs must be transferred and deployed. This process is called virtual machine migration. VM migration helps in improving the efficiency of the cloud, by maintaining proper load balance, discussed by Mohamadi *et al*. (2011).

But the point to be noted here is, VM migration involves extra data involve. For an instance, if a VM image is handling only one request at a system and it needs till ten units of time to complete. Meanwhile another VM is fully overloaded and needs an under-loaded VM to help it. So this former VM is migrated while its process is at the fifth time unit of execution. Here, this migrated VM must keep track of the service it is providing and also take care of the new upcoming process. After migrating if the VM performs to its fullest, then it is valid. But for performing a two time unit job if the VM is migrated on keeping track of the pending five unit request, then it is memory and resource kill, which should be addressed.

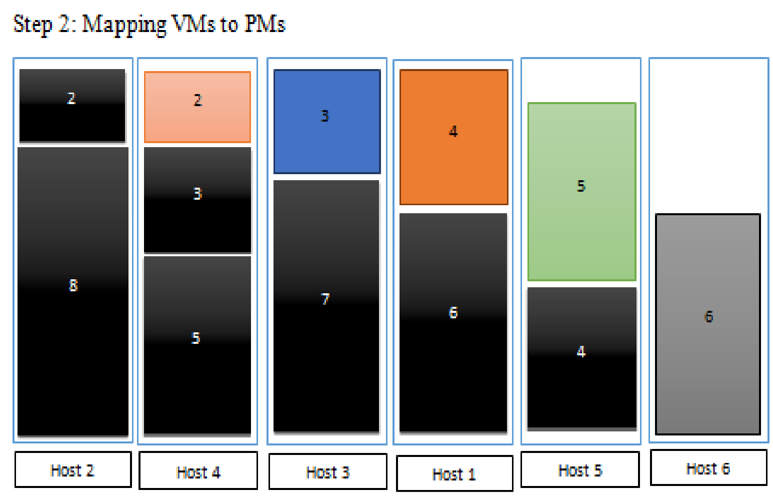
This proves that placing a virtual machine must ensure that the placement made is so efficient so that



(a)



(b)



(c)

Fig. 5: Working of most full algorithm

the resources will be utilized to its maximum. Maximum utilization of resource here means, there is no waste of resource, that is, limited power is consumed, as discussed by Fidanova (2005).

Also all the algorithms discussed above just address a solution only in a single dimension. For placing a virtual machine, the parameters that take predominant roles are the physical machine capacity, the processing speed, RAM size, maximum number of resources that it can accommodate and most importantly how long it will be kept on. All these details play an important role for succeeding in the mission mentioned above. For which an implementation idea comes below.

Knapsack problem: Knapsack problem is a NP Hard problem which demands an optimized solution in dealing with resources. It is a type of the bin packing problem where in the choice is left with a variety of items and a constant volume of container, in which the container must be filled full also ensuring that it provides optimum profit. The items to be filled inside the container must be selected in such a manner that the output is the maximum out of all combinations.

Types of knapsack problem: There are two major types of knapsack problem, as discussed by Song *et al*. (2008), such as.

0/1 knapsack: This is a type of knapsack which obviously focuses on optimizing the profit, yet does not compromise in selecting a part of the item. For example, if there is an item of capacity 5 units and value 15 units and a container of volume 3 units. This type of knapsack will not allow in selecting this item, because 0/1 knapsack either selects the item as a whole (1) or drops it (0), detailed by Vasquez and Hao (2001).

Fractional knapsack: Fractional knapsack is a bit flexible also a complex method which does anything to maximize profit. Consider the same example; there is an item of capacity 5 units and value 15 units and a container of volume 3 units. This type of knapsack will take just take 3 units of the item ensuring a profit of 9 units.

In the 0/1 knapsack, for the same example the profit is nil, but in the other method it is 9 units. And from this, obviously, 0/1 is less complex, but fractional needs more computations in choosing the optimal profit yielding item if n number of items is present.

In our case to map VM requests, obviously, it is not possible to split a request into fragments and then allot them to physical machines. Doing so, just increases the complexity of the placement process since there must be dedicated units to split the resources before sending to a host and another one to merge and consolidate them, as said by Singh *et al*. (2008). Hence we go for the 0/1 knapsack problem, where if we find a host with available resources, we utilize it, or go in search of a next host.

Solving a knapsack problem: Knapsack problem is a standard np-hard problem that can be implemented in various fields. Now the solution to this will be using any of the following methods, as by Fréville and Hanafi (2005).

Greedy method: In greedy method, the overall profit will be focused and to achieve that the sequence or the pattern that gives higher gain will be selected. This method blindly calculates all possible patterns that are feasible and chooses the best out of it.

Divide and conquer: As far as this method is concerned, the given problem will be divided into smaller sub problems, will be consolidated and the result will be found. The sub problems which are divided have no relation to each other. Output of one sub problem does not affect the output of the other.

Dynamic programming: While choosing to solve using dynamic programming, the given problem must be divided into smaller sub problems, whose result will be stored and retrieved for calculating the value of the next sub problem. All the sub problems are inter-related, in a way that outcome of a sub problem depends on the output of the previous sub problem.

There are still more many traditional ways in which a knapsack problem can be solved, but in our case, we choose genetic algorithm to solve, since it gives a more optimal output.

Implementing knapsack problem to solve the virtual machine placement problem, considers the physical machines as bins and virtual machines as objects to be filled in. And follows the steps given below:

- Formulate the patterns of the past demands of VMs at the particular environment.
- Anticipate future demands based on the past details that are obtained from the previous step.
- Map or remap virtual machine to physical machines.

Do repeat this process for quite a regular time interval. The main rule is measure-forecast-remap the placement procedure.

0/1 knapsack demerits: We do get a better result in implementing the knapsack problem in distributing the virtual machines over available hosts, yet what if multiple VMs come into play. Also, since many virtual machines are dealt with we go for implementing multiple knapsack problem, which literally says that one or more bins are involved, as discussed by Vasquez and Vimont (2005).

Also as far as value and volume are considered, normal knapsack itself can work well. But placing a VM deals with various factors like, processing power, memory, storage, network bandwidth, etc., each one contributes a unique dimension. Hence multiple multi-dimensional knapsack problems can be implemented to get more optimized results.

Proposed solution: We are trying to optimize the resource allocation by implementing multiple multidimensional knapsack algorithm so that the chances of migration will be less also, the resources will be utilized to the maximum, as said by Gottlieb (2000). Consider the following example. Its working is detailed as follows, Fig. 6:

- Initially a user tries to access the cloud, where the actual VM request is triggered.
- Likewise, many VM requests will be initiated, which get gathered in a VM queue.
- VM queue, which accommodates the incoming VM requests, must be highly scalable.
- The next step is to have a clear idea about the available physical machines. Though it is dynamic, at least to a reasonable extent, the scenario must be known.
- Then, each VM request is considered a knapsack problem. We name it multiple since, many VM requests are to be addressed at a while.
- These VM requests are with respect to various dimensions like, processing speed, storage, memory, network, etc., hence we name it multidimensional.
- To solve this problem, we apply genetic algorithm, which is one of the evolutionary algorithms that can handle any type of data and give an optimal result.

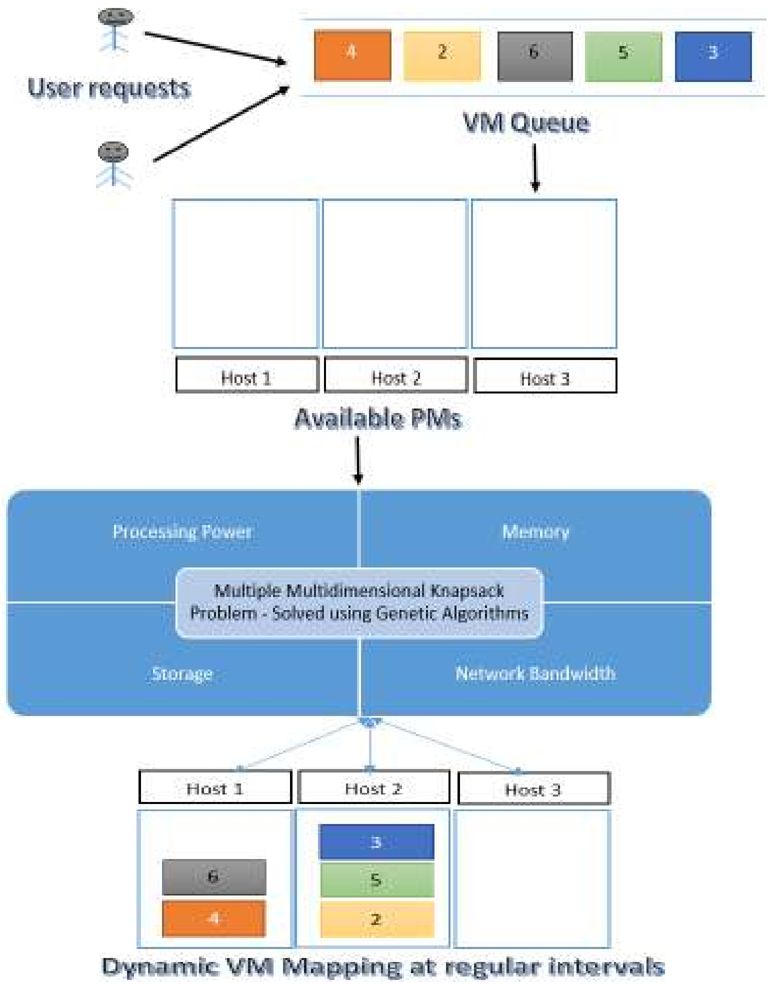


Fig. 6: Mapping VM to PM by implementing multiple multidimensional knapsack algorithm

- Once when we get the optimal output, or the maximum number of times is reached, we iterate this algorithm, to get the best solution.
- This algorithm may take a considerable time in doing these calculations, but provides the optimum placement which minimizes the chances of migration.
- According to the result suggested by this algorithm, the VM image distribution will take place.
- This process does not end up here. This algorithm along with the scheduler keeps a constant study of the status of the PMs and VMs.
- Thus the current requirements of the cloud network will be read clearly, every now and then, so that resources can be dynamically granted, wasting nothing.

By this, there will be continuous updates about the virtual machine requirements and the existing available resources pool. These resources, since it takes more than one dimensions, the placement will be optimal.

Genetic algorithm: Genetic algorithms are efficient methods deployed, where a searching or an optimization problem exists. This algorithm will work fine even if there a huge data as inputs and to be handled fast, as said (Khan, 2013). This algorithm comes to aid when there has no solution for a given problem, or the existing solution must be enhanced for optimal result. In our case, Genetic algorithms help to optimize the benefit of a group of objects in a knapsack, considering its capacity constraint, as discussed by Koza (1992). The working of this algorithm is explained as follows, see Fig. 7.

Initially the inputs to the problem are considered as the primitive population. This primitive population is initialized by either of the two ways, random initialization or induced initialization. In random initialization, the populations are created by mining at random, without any rules. Induced initialization, on the other hand gets constant information relating to the given values and background information, to construct the primitive population, as explained by Hinterding (1994).

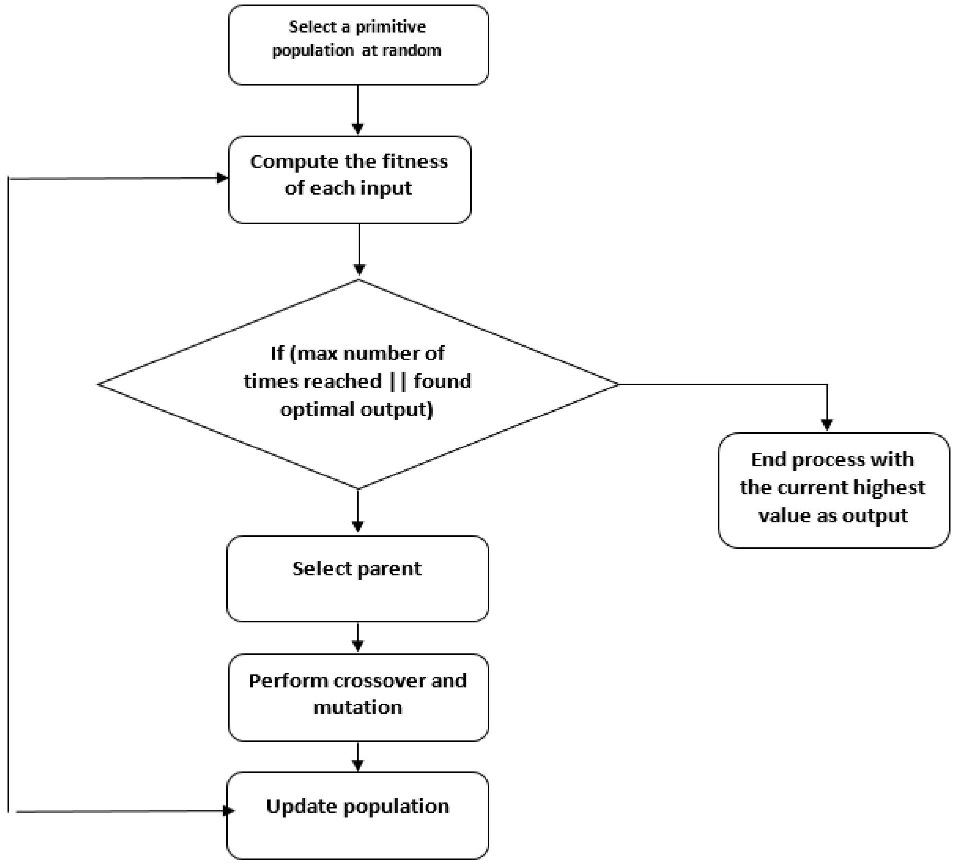


Fig. 7: Working of genetic algorithm

After this, with the help of the individuals in the existing population, this algorithm gives birth to the next generation of population to create new sequence of new populations. Creation of new generations of offspring continues until the algorithm meets with the stopping condition. The creation of new generations is done as follows, as discussed by Khuri *et al*. (1994).

Each element is scored by a fitness value, which helps to find the optimal solution. According to the nearness of the fitness value, the elements are classified into groups of different ranges. If the fitness value of an element is reasonably good, then it will be qualified to be a parent. The elements that have best fitness will be automatically moved to next generation.

Then parents give birth to their children in either of the ways.

Mutation: In this way of giving birth to a child, only a single parent is involved. By changing the vector entries of a single parent, randomly, gives birth to a child.

Cross over: In this method, to create a child, two parents are involved. Combining the vector entries of two different elements to form a child.

Once when a new generation is formed, it replaces the existing generation. If the new generation contains an optimal solution, the problem is solved, else this algorithm continues by creating a new generation from the existing generation, as discussed by Sun and Wang (1994).

Genetic algorithm to solve a knapsack problem: To solve the knapsack problem, the fitness function can be defined by calculating the sum of benefits of an item that can be included in the knapsack, ensuring that the maximum capacity of the knapsack is not exceeded. Thus with the first generation population, the fitness of each element is found. It is then categorized into groups of different ranges after sorting the fitness value ascending. Since we follow random selection of parents, from each group an element is chosen at random. The more fitness an element had, the more likely is the chances for it to qualify as a parent. Then parent(s) give birth to new generation either by mutation or cross-over. There are cases where in the first generation itself there is an element found with a high fitness value. These elements are called as elite and they are passed on to the next generation without any changes, with reference to the discussion by Mahajan and Chopra (2012).

The algorithm:

1. Select first generation from $\square = \{s_1, s_2, s_3 \ldots s_n\}$
2. Calculate the fitness value $\forall \square$
3. Sort these elements in the ascending order of their fitness value.
4. Set Count to 0.
5. No_change_count set to 0.
6. Do;
7. Hi_fit = most fit element in the current generation
8. Find the top two fittest elements and consider them elite; pass them to next generation without any change.
9. Do;
10. Set position to 0.
11. Random_val = Random (0, 1)
12. Select parentA from the groups in the current generation using Random_val.
13. Random_val = Random (0, 1)
14. Select parentB from the groups in the current generation using Random_val.
15. Perform cross-over between ParentA and ParentB
16. Position = position + 2
17. Pass the created children to the next generation
18. Until position = (size/2) -2
19. Count = Count + 1
20. Replace current generation with the newly created generation.
21. Sort the current generation and perform mutation.
22. New_fit = most fittest element in the current generation.
23. If New-fit < = Hi_fit then
24. No_change_count = No_change_count + 1
25. Else
26. No_change_count = 0;
27. End if
28. If (No_change_count>max_number_of_times) then
29. Break
30. Until Count = max_number_of_generations

Evaluation method: Virtual machine placement demands for some constraints to be satisfied. It must ensure that the resources are properly matched, so that the above said goal is met, as said by Volgenant and Zoon (1990). The virtual machine placement problem can be represented in the mathematical formulation as follows:

Let,

$P = \{p_1, ..., p_u\}$: a set of u physical machines
$V = \{v_1, ..., v_y\}$: a set of y virtual machines
$C = \{c_1, ..., c_z\}$: a set of z constrains

- T_r: The arrival time of $v_r \in V$ such that $\forall$ v_q, v_r: $q<r \Leftrightarrow T_q<T_r$
- $D_{r,s}$: The demand of $v_r \in V$ in constraints S
- $M_{q,s}(T_r)$: The free capacity of constrains S in $P_q \in P$ at T_r
- $N_{q,s}$: The total capacity of $p_q \in P$ in constrains S
- $x_{q,r} \in \{0, 1\}$, $1 \le q \le u$, $1 \le r \le y$
- $x_{q,r} = 1 \Leftrightarrow v_r \in V$ is placed in $P_r \in P$
- $x_{q,r} = 1 \Leftrightarrow X_s$, $r = 0$, $\forall$ $S \neq q$

Maximize:

$$\sum_{q=1}^{u} \sum_{r=1}^{y} x_{q,r}$$

Subject to:

- $\sum_{q=1}^{u} \sum_{r=1}^{y} \sum_{s=1}^{z} x_{q,r} . D_{r,s} \le n_{q,r}$
- $\forall v_r \in V$, $\exists A \subseteq P$, $A \neq \emptyset$ | $\forall p_q \in A$, $\forall c_s \in C$, $M_{q,s}(T_r) \geq d_{r,s} \Rightarrow x_{t,r} = 1$, $p_t \in A$

The evaluation method is used to know how the virtual machines get fitted into the physical machine and check if they are placed to the optima, by analyzing the behavior of various algorithms. Each request has a unique dimension, so do the physical machines have. These dimensions vary from resource to resource.

The perfect scenario is a scenario in which, all available physical machines, perfectly accommodate the incoming requests. That is, each virtual machine is flawlessly mapped to the host nodes, without any compromise in the resource requirement. For example, consider a host which can accommodate ten resources. Now a VM request comes and it is directly mapped to the prior said host; instead of splitting it into many chunks and redirecting it to many hosts.

Here the number of virtual machine requests and nodes (hosts) are assumed as homogeneous which means the number of machines and hosts are identified randomly and set into their limits. After the perfect scenario the user can introducing some variants in their dimensions of both machines and hosts. For the heterogeneity scenario, the dimension of a set of machines may vary within the range. The range has maximum, minimum and median values. The proportional difference of the dimensional with respect to median is called as dimensional amplitude with respect to the machines.

The dimensional amplitude for the dimensional D for a set of H hosts, the dimensional amplitude D with respect to H and it is denoted in (1):

$$DA_{H,D} = 1000 \text{ (max - median) /median} \quad (1)$$

where,
Max = Maximum dimension range
Median = Median of the dimension

In a set of H of hosts where each host machine has m dimensions that are labeled from 1 to m, then the dimensional amplitudes denoted for all the dimension is referred as (2):

$$DA^* H, D = \sum_1^m DAH, i \quad (2)$$

Consider VM is set of virtual machines needs to be placed and PM is set of machines placed after applying the algorithm, h is the number of machines in VM and pm is the number of machines in PM. The equivalent VM placement ratio (α) is defined as (3):

$$\alpha = pm/h \quad (3)$$

If the hosts and machines are homogeneous, then the value of α becomes 1. On the other hand the ratio may vary which means either no resource left in host or data center.

RESULTS AND DISCUSSION

Experimental results: All the algorithms, which are taken under consideration for this study, were experimented to know the comparative results to find which algorithm suits the best to do the optimized placement of the virtual machines on to the physical machines. This is done with the help of the evaluation method that is described above. This experiment was done by simulating in CloudSim and the results were obvious that the existing methods showed poor performance while the algorithm which we wrote showed comparatively better results.

Figure 8, the scenario is that, keeping the host amplitude fixed, we studied the impact on virtual machines amplitude variation. Thus by increasing virtual machine amplitude evenly we studied that our

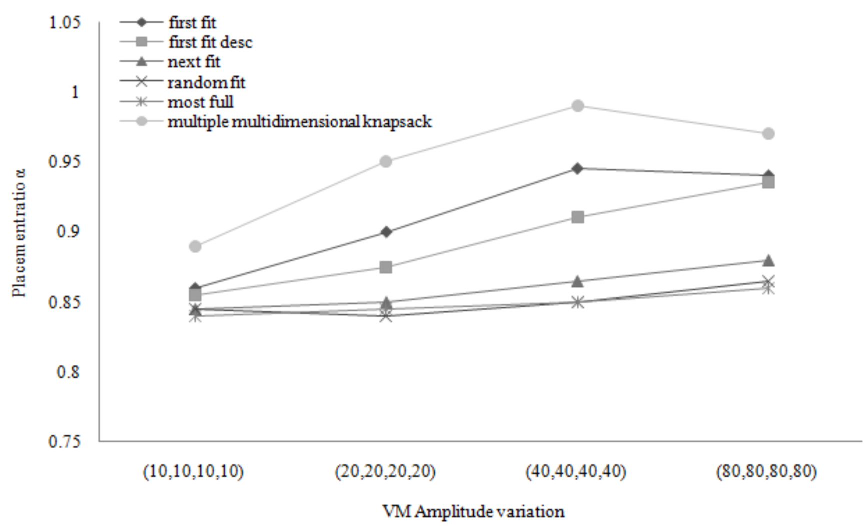


Fig. 8: Increasing VM amplitude variation evenly

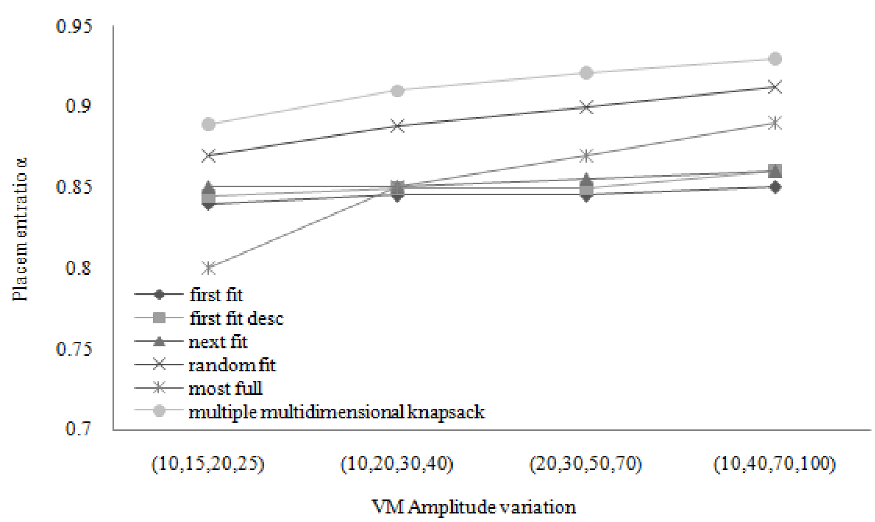


Fig. 9: Increasing VM amplitude variation unevenly

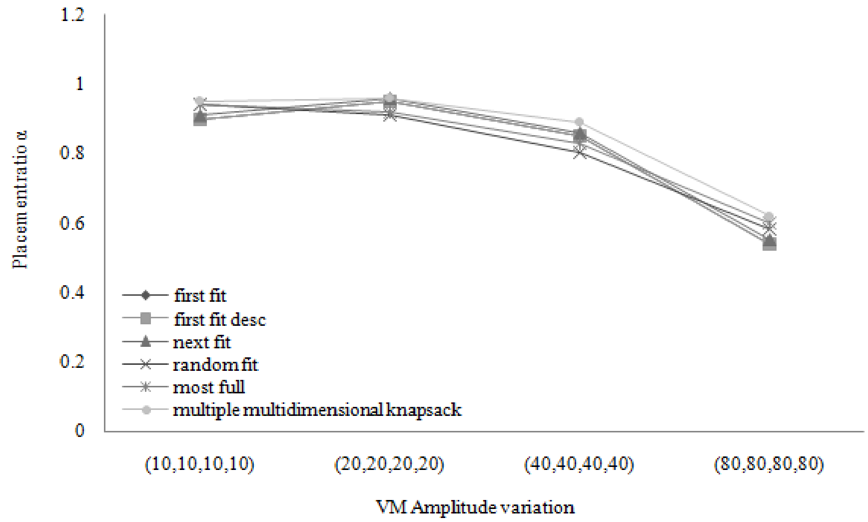


Fig. 10: Increasing host amplitude variation uniformly

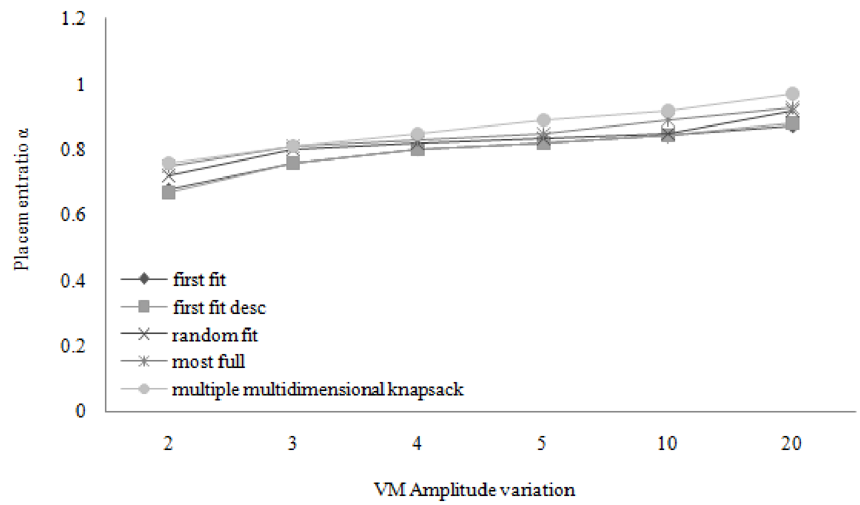


Fig. 11: Increasing VM density in a small range

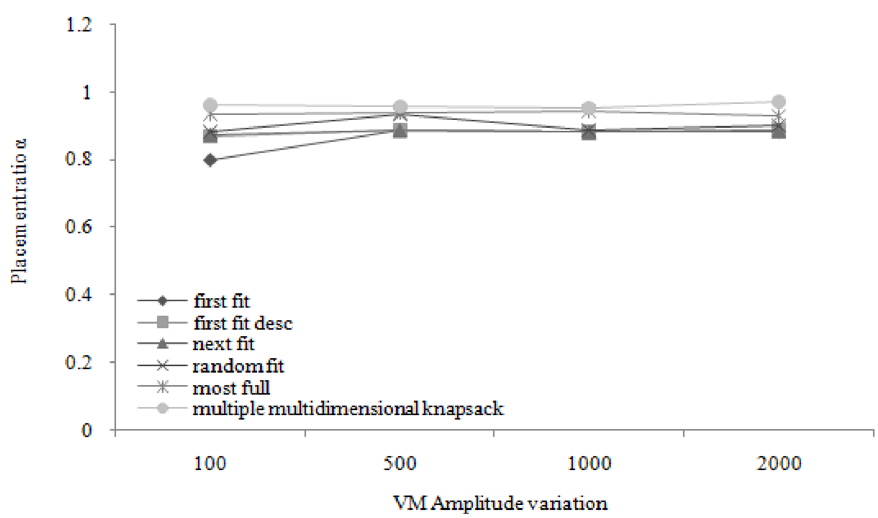


Fig. 12: Increasing VM density in a small range

algorithm achieved the best placement ratio which is 2 to 10% better than the other algorithms.

In Figure 9, we keep the host amplitude fixed and increase virtual machine amplitude unevenly we see that our algorithm achieved the best placement ratio which is 20 to 10 % better than the other algorithms.

In Figure 10, we increase the host amplitude variation to see the impact on placement ratio. Even in this scenario our algorithm shows better performance from 6 to 19% in spite of increasing the host amplitude uniformly or non-uniformly.

In Figure 11, we traced the impact of VM density by fixing the physical machine and virtual machine amplitude. Once again here too, our algorithm was the best performer showing 3-12% higher placement ratio compared to the other algorithms.

In Figure 12, here too we fix the physical and virtual machine amplitudes to see its impact on VM density. And for the results, our algorithm shows the better performance from 6 to 9%, which is more efficient than first fit and other algorithms.

As we see, comparing all the methods, our algorithm which solves the multiple multidimensional knapsack problems using genetic algorithm gives the optimal result in placing the virtual machines. Other algorithms do not contribute so much, in efficiently placing the virtual machines. With our algorithm, the chances of migrating a virtual machine from one host to another will be very minimum, but when using other algorithms it is not the case. It is obvious that they show poor placement.

CONCLUSION

Cloud computing which created such hype in the technology, deals with major problems like, security issues, power consumption and resource utilization. Of which, the resource utilization is taken under major consideration in this study. To optimize the resource allocation or technically to say, Distribute VMs to physical hosts, the already existing placement processes are learnt, analyzed and a new implementation idea is given. Though knapsack is not a new idea that is invented, implementing multiple multidimensional knapsack algorithm and solving it using genetic algorithms to map virtual machines to physical machines, serves the purpose. Also we are trying to extend our current work to be implemented in a real time cloud environment in near future.

ACKNOWLEDGMENT

We are so thankful to the Computer Science Department of Amrita School Of Engineering, Amrita Vishwa Vidyapeetham for providing us with immense support, motivation and suggestions, for making us work on this title and successfully complete this study. We also take immense pleasure in thanking our guide, Mr. Prakash, who was the backbone of this study. Without his guidelines and support, this study would have not been in shape. We also thank our friends, who encouraged us to go ahead with this idea. Without them, this process would have been very monotonous.

REFERENCES

Creasy, R.J., 1981. The origin of VM/370 time-sharing system. IBM J. Res. Dev., 25(5): 483-490.

Fidanova, S., 2005. Heuristics for multiple knapsacks problem. Proceeding of the IADIS International Conference on Applied Computing. Algarve, pp: 225-260.

Fréville, A. and S. Hanafi, 2005. The multidimensional 0-1 knapsack problem bounds and computational aspects. Ann. Oper. Res., 139: 195-227.

Gottlieb, J., 2000. On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In: Fonlupt, C. *et al*. (Eds.), AE'99. LNCS 1829, Springer-Verlag, Berlin, Heidelberg, pp: 23-37.

Hinterding, R., 1994. Mapping, order-independent genes and the knapsack problem. Proceeding of the 1st IEEE International Conference on Evolutionary Computation. Orlando, pp: 13-17.

Khan, M.H.A., 2013. An evolutionary algorithm with masked mutation for 0/1 knapsack problem. Proceeding of International Conference on Informatics, Electronics and Vision (ICIEV, 2013), pp: 1-6.

Khuri, S., T. Bäck and J. Heitkötter, 1994. The zero/one multiple knapsack problem and genetic algorithms. Proceeding of the 1994 ACM Symposium on Applied Computing, pp: 188-193.

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, MA, USA, Dec., ISBN: 0-262-11170-5.

Mahajan, R. and S. Chopra, 2012. Analysis of 0/1 knapsack problem using deterministic and probabilistic techniques. Proceeding of 2nd International Conference on Advanced Computing and Communication Technologies (ACCT), pp: 150-155.

Mell, P. and T. Grance, 2011. The NIST Definition of Cloud Computing. Technical Report, National Institute of Technology Special Publication 800-145, Gaithersburg.

Mohamadi, E., M. Karimi and S.R. Heikalabad, 2011. A novel virtual placement in virtual computing. Australian J. Basic Appl. Sci. Aust., 5(10): 1149-1555.

Pisinger, D., 1995. Algorithms for knapsack problems. Ph.D. Thesis, Department of Computer Science of University of Copenhagen, Copenhagen.

Shiva Jegan, R.D., S.K. Vasudevan, K. Abarna, P. Prakash, S. Srivathsan and V. Gangothri, 2014. Cloud computing: A technical gawk. Int. J. Appl. Eng. Res., 14: 2539-2554.

Singh, A., M. Korupolu and D. Mohapata, 2008. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. Proceeding of International Conference for High performance Computing, Network, Storage and Analysis, pp: 1-12.

Song, Y., C. Zhang and Y. Fang, 2008. Multiple multidimensional knapsack problem and it's applications in cognitive radio networks. Proceeding of Military Communications Conference. San Diego, pp: 1-7.

Sun, Y. and Z. Wang, 1994. The genetic algorithm for 0-1 programming with linear constraints. Proceedings of the 1st IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence (ICEC'94), Orlando, pp: 559-564.

Vasquez, M. and J.K. Hao, 2001. A hybrid approach for the 0-1 multidimensional knapsack problem. Proceedings of the International Joint Conference on Artificial Intelligence, pp: 328-333.

Vasquez, M. and Y. Vimont, 2005. Improved results on the 0-1 multidimensional knapsack problem. Eur. J. Oper. Res., 165: 70-81.

Volgenant, A. and J.A. Zoon, 1990. An improved heuristic for multidimensional 0-1 knapsack problems. J. Oper. Res. Soc., 41: 963-970.