Eurasian Journal of Analytical Chemistry
ISSN: 1306-3057 OPEN ACCESS 2018 13 (SP): 431-440

# Cloud Scheduling of Virtual Machine Using Machine Learning and Decision Tree

Sourav Modak, S. Nagadevi

Received 11 November 2018 ▪ Revised: 30 November 2018 ▪ Accepted: 06 December 2018

***Abstract:*** This paper tries to provide a possible solution to a very common issue in Virtualization technology, in cloud computing, i.e., the placement of virtual machines (vm's) in the physical machine. The physical machine here is considered as the server, where each of the vm's are placed. The challenge is to reduce the time and may be the space complexity of the scheduling. This paper here talks about probably a new kind of approach, as a solution to this problem. The algorithm here used takes the inspiration in the way the human brain behave while making decisions, where it first rely upon the past experience and then either modifies the experience in accordance to the situations, or else go for new techniques, if inferred. The algorithm here however is not capable of inferring, or deducing new techniques, however it is capable enough to store the past experience or the pattern in which the vm's were allocated and if that repeats, it instead of again calculating or in other words, traversing through the whole algorithm, it directly allocate the vm's in the server.

***Keywords:*** VM's, PM's, Virtualization.

## INTRODUCTION

The paper here brings a new type of algorithm or a new kind of improvisation that can be applied to an already established algorithm to make it more optimized and make it faster. As discussed earlier, the algorithm here however is not capable of inferring, or deducing new techniques (as it would be a science fiction equivalent), however it is capable enough to store the past experience or the pattern in which the vm's were allocated and if that repeats, it instead of again calculating or in other words, traversing through the whole algorithm, it directly allocate the vm's in the server.

This gives huge advantage to the algorithm, since it is not hitting the "logical part" or specifically speaking the "deep critical logical part" of the algorithm, it gives a huge speed advantage. This will not only reduce the time but also may be the space used, depending upon the algorithm if it uses the memory variables. The algorithm here is the outer housing for other algorithms. The inside main critical log may be any algorithm (may be first fit or best fit) but this algorithm will border or cover it externally and provide an interface which will automatically help the scheduler to decide whether to rely upon the past experience or go and hit the logic part of the algorithm. Here to store the info about the experience, the pattern that is discussed earlier in this paper, is done by Relational Database Management Systems, where the test of this algorithm is done in MySQL. The code used the tables of the database and at last only stores the decodable info about the pattern in which a vm is allocated if it is a new one. The rest depends upon the type of algorithm used to process the scheduling.

### Abbreviations and Acronyms

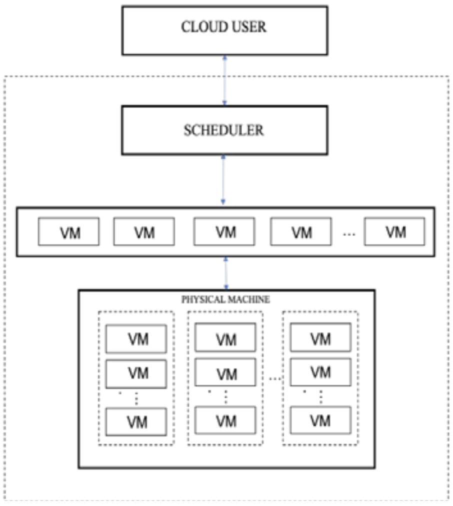
- **ID-**Identification number as stored in the database as one of the attribute of server/VM.
- **VM-** Virtual Machine.
- **PM-** Physical Machine.
- **HP-** High Priority.
- **LP-** Low Priority.
- **HC-** High Consumption.
- **LC-** Low Consumption.

Sourav Modak, Department of Computer Science and Engineering, SRM University, Chennai, India. E-mail: official.srv.modak@gmail.com

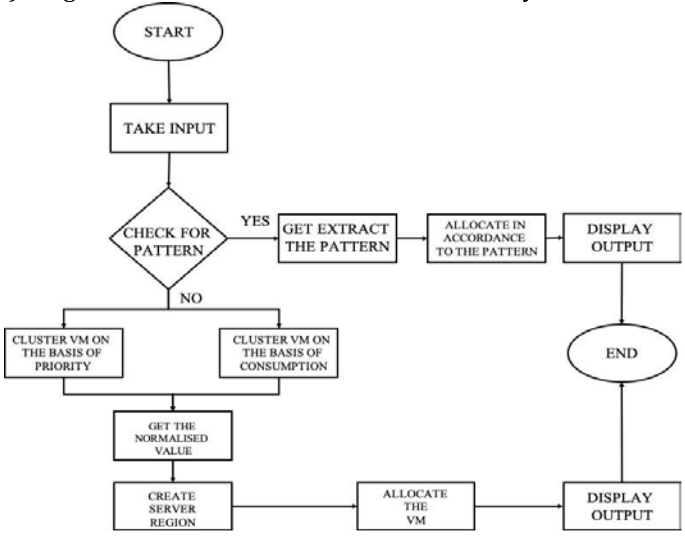
S. Nagadevi, Assistant Professor, Department of Computer Science and Engineering, SRM University, Chennai, India. E-mail: nagadevi.s@ktr.srmuniv.ac.in

- **n-** nth number in the loop.
- ❖ Sometimes Physical Machine or PM are referred in this paper as servers and should be hence considered as same.

**Figure**



(a) Diagram of Virtual Machine Allocation in Physical Machine



(b) Algorithm Flow Diagram

**Algorithm**

**a.** Start.
**b.** Take input.
**c.** Go to database.
**d.** Decide the algorithm to be used, whether first fit or best fit.
**e.** If best fit, go to the next step, else go to **step k**.
**f.** Search for pattern in the database of the local scheduler.
**g.** If present go to **step n**, if not present go to next step.
**h.** Cluster the VM's by making two groups in order of high priority and low priority.
**i.** Cluster the VM's by making another two groups in order of high consumptions and low consumptions.
**j.** Get the normalised value;
   I. To get the normalised value, go to next sub step.
   II. Create a queue and name it as High VM's.
   III. First add the high priority, low consumption values to the High VM queue.
   IV. Then add high priority, high consumption values to the High VM queue.
   V. Create another queue, and name it as Low VM's.
   VI. Then add low priority, low consumption values to the Low VM queue.
   VII. Then add low priority, high consumption values to the Low VM queue.

VIII. Export both the queues as the normalised value.

**k.** Create the server region, as high resource region, and low resource region, on the basis of resource it possess.
   I. If best fit, go to next step. Else go to next sub step.
   II. Allocate the VM's to the server on the basis of FCFS.
   III. Go to **step m**.

**l.** Allocate the server region
- ➢ First allocate the high resource region and then low resource region.
- ➢ While allocating the VM to the server region,
    a. First allocate the High VM queue.
    b. Then the Low VM queue.

**m.** Now for each server, see the pattern and write it in the database.
- ● *The pattern should look be of this form as discussed in the pattern section.*

**n.** Infer the pattern and allocate the VM's to the server.

**o.** Display the allocation as the output.

**p.** Stop.

**Description of Algorithm**

- ❖ Before we start with this algorithm, we need to understand that this algorithm act as gate for other efficient. This algorithm is just like a toggle that switches or enable the learning and inferring capabilities of the existing algorithms so that we can use the old experiences of that algorithm results instead of again tirelessly calculating the scheduling pattern. So we store this pattern in such a way that when any similar kind of it is encountered we directly do as we have done previous time, and save the cost and time of repeated calculating and scheduling. Below is the best fit algorithm and the first fit is discussed in the later sections.

## BEST FIT ALGORITHM AS CORE ALGORITHM

- ❖ Get the **physical machine (PM)** or server input and **Virtual machine (VM)** input.
  - ● The input here is trivial, so in algorithm we only concentrate upon the input of the values but not the type of input given. As far as implementation of this algorithm goes, there's much of possibility that this will have different modes of inputs, depending upon on the scenario it has been used in. It can be a manual input or dataset input based on data from different internet giants like google, amazon etc. This algorithm implementation is done and tested with random inputs from different datasets as we have gathered from different firms as mentioned.
- ❖ Cluster the Virtual Machine in the order of the priority.
  - ● The clustering of Virtual machine can be done in any manner. The easiest one may be a K-Means clustering, where we cluster value based on the basis of the average value that it taken as input dynamically. We can use other clustering method to cluster them in two different groups. The only thing to be kept in mind is it should be optimised properly so that it don't compromise on the performance of the algorithm. However the algorithm may not hit the clustering part after it is trained properly, but till then it may work inefficiently if the above thing is not kept in mind.
  - ● For the implementation part we have used the K-Means clustering algorithm to cluster the VM and proceed with the algorithm.
- ❖ Cluster the Virtual Machine then in accordance to the consumption.
  - ● The cluster on the basis of the consumption is a tricky one. The PM's and VM's will have more than one offerings as resource, so this increases the dimensions for comparison. This may increase the complexity of the algorithm so, we need to deal with this thing carefully, as referred as **Curse of Dimensionality** [12]**.** As the curse of dimension states that the complexity increases as we increase the dimensions, therefore resulting in decrease in performance. Hence it is important that the comparison is made efficiently else it will result in decrease in performance of the algorithm since the number of VM's in the real-time scenario will be crazily high.
  - ● For this, we may consider only the dimensions which is important and may have the major impact in deciding if the unit (server or VM) possess high resource or the unit possess low resource when compared to others. After then, as we figure out the way in which we compare the VM's on the basis of resource, we may start to cluster it into two groups, i.e., high resource consumption VM's and low resource consumption VM's.
  - ● The clustering in accordance to the resource consumption can also be done in many ways. In the implementation check we take the sum total all the resource values such as the ram, the diskspace and the processor cost.

- We may also individually compare each of the configuration and then decide the higher and lower consumption with all the things that is discussed earlier should be kept in the mind.

❖ The server are then grouped into high resource servers and low resource servers.
- This is to create the server regions. Each region may have more than one server which is grouped together and titled either as high resource or low resource. This is done so that we have all the high resources to be filled first and then the lower resource one. This will give a performance advantage to the algorithm execution and also we can allocate more and more VM's to the server with lowest resource wastage of server or physical machine if we follow this pattern.

❖ Then calculate the normalised value, out of the two parameters i.e., priority and resource consumption for the Virtual Machines to cluster them into Important Virtual machines, and Next Important Virtual Machines. **Do note that this step depends also on the type of algorithm used for the allocation. If it is First Fit [6], then this step have to be modified for FCFS mode of process. Else we can carry forward with the above step, if it is Best Fit algorithm [7].**
- This is most complicated step in this algorithm. As we have compared broadly two dimensions, i.e., the priority and resource consumption of the server. So as discussed earlier the **Curse of Dimensionality[12]** increases the complexity of the algorithm and hence we need to find something in between or the normalised value, so that we may proceed with the algorithm while keeping the complexity to minimum and also increase the performance.
- This is achieved by applying the common sense. As we are discussing the best fit algorithm as the core algorithm for the VM placement in the PM, we proceed with the idea that we first need to allocate the high priority VM's first and then the low priority one. Also in each high or low priority, we need to see if the VM's in each groups (high priority or low priority) is high resource consumption VM or low resource consumption VM.
- So we might follow this flow:- high priority/low consumption>high priority/high consumption>low priority/low consumption>low priority/high consumption. This is just one of the suggested flow and might not be the best possible flow, since the scenario is situation dependent and hence we don't label or claim any flow to be the best, but rather give the closest possible solution to the problem we are discussing.

❖ Hit the RDBMS [8] for extraction of the pattern, to see if there is any past identical pattern as discussed above is stored which can be used.
- As we need to store the pattern after the program session ends, and hence we use the relational database to store the pattern since the relational database can provide the highest performance when it is needed to extract anything that is stored in the secondary memory. Also since the non-volatility as its nature provides us the service to store it permanently.
- We use a table named in the format - <Server-(ID)> and the table should contain the pattern in which the each VM's are stored as rows. The pattern would be in the format as discussed in the pattern section.

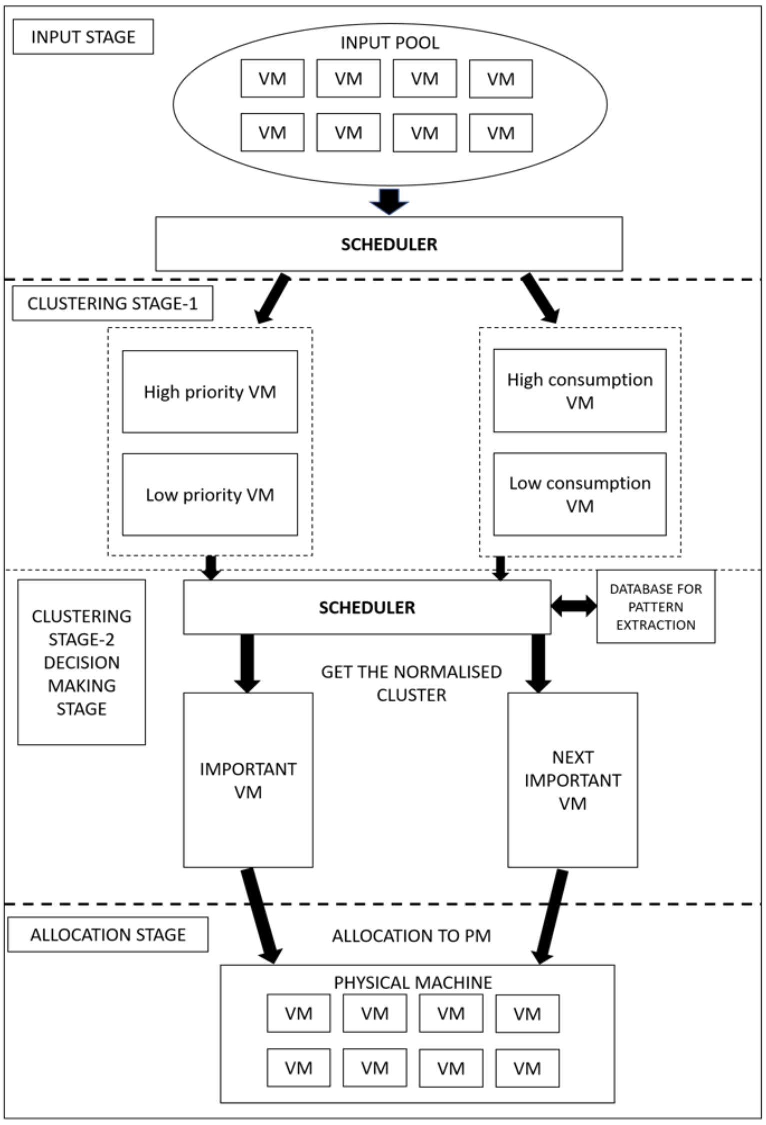
❖ The high resource region is then allocated to the Important VM group, followed by the Next Important VM group, if the high resource region has enough room.
- Here as discussed in the earlier steps, we find a normalised queue of VM's so that we can be sure of the fact that we have considered both the priority and consumption capability of the VM's before allocating the VM's.
- This normalised queue is further divided into two groups, where the one is called Important VM group and Next-Important VM group and as the name suggest, the former should be used to first allocate it in the server region and then we use the latter queue's VMs.
- This normalisation in this case is only done considering two dimension, but as the scenario is situation dependent, the programmer may use more than one dimension to apply the filter and then obtain a normalised value. This is a flexibility provided by the algorithm so that the logic can be extended.

❖ Same way fill the low resource region group with the Important VM, and then Next Important VM.
- Do the same for the low resource consumption as we have done with the high resource consumption VMs.

❖ Record the pattern, i.e., the way in which the VM's are allocated to server with all necessary details such as number of servers and number of VMs, and map it for extraction/search[9], during pattern check.
- The pattern is recorded in the RDBMS in the way as discussed earlier.

❖ Display Allocation.

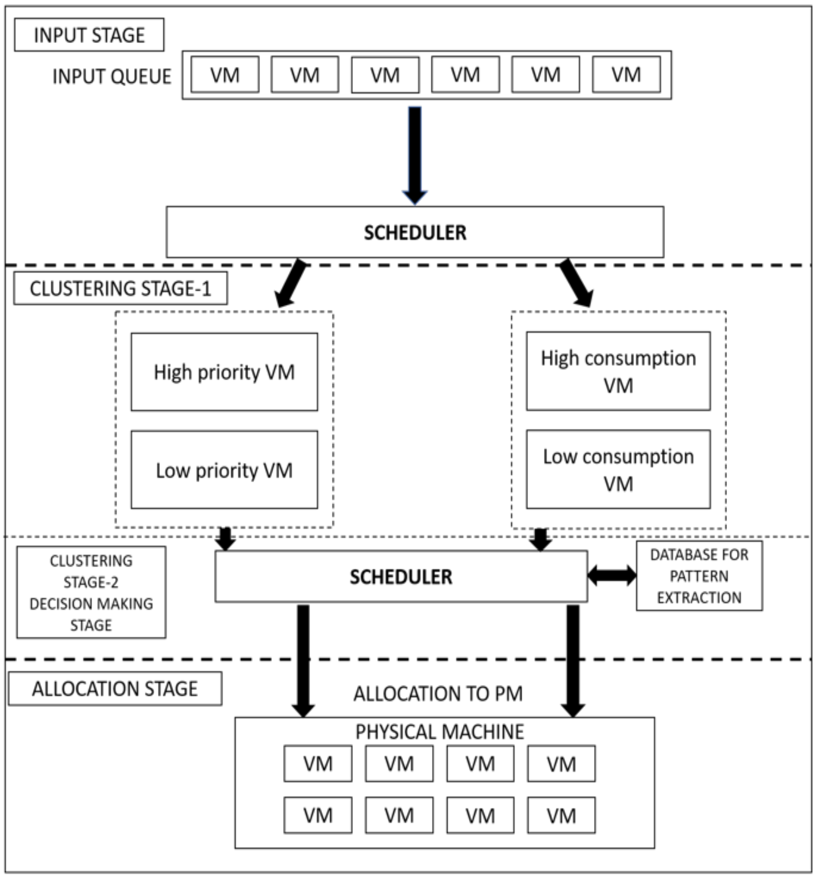
❖ Hit the RDBMS [8] to store the record.



Flow Diagram for best fit algorithm

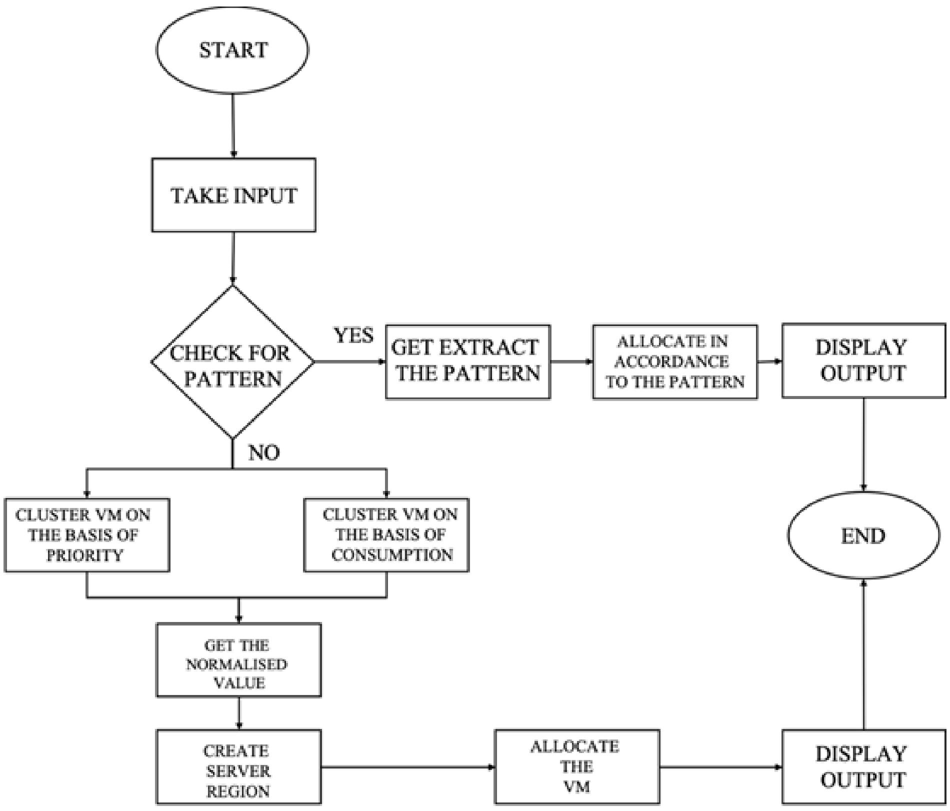
## FIRST FIT ALGORITHM AS CORE ALGORITHM

- ❖ Get the **physical machine (PM)** or server input and **Virtual machine (VM)** input.
  - This thing remain more or less the same since it's just the input aspect of the code and need not be different.
- ❖ Create the server regions.
  - The server regions contain the servers and is allocated and with VMs in the order of FCFS. The server regions are same as we have seen in the Best Fit algorithm section.
- ❖ Cluster the VM with respect to the priority and also consumption of VM.
  - This is a step which we need to follow in this type of algorithm too since our pattern depends on type of VM that is if it is a high priority VM or low priority or if it is high consumption or low consumption.
- ❖ Check for the pattern in the database and see if we have any similar pattern.
  - This is also same as that of the best fit algorithm case and we directly extract the pattern from the data from the database if it is found.
  - If the pattern is no found then we proceed with the algorithm into the main logic part where we then allocate the VM into the PM and then store the pattern.
- ❖ Go to the allocation of VM's to the PM in the order of FCFS sequence and then record it into the database to use it in the future.
- ❖ Display the output.

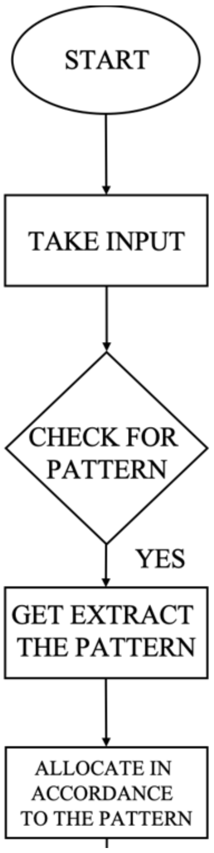


Flow Diagram for first fit algorithm

**THE FLOWCHART/DIAGRAMMATIC EXPLANATION**



## ABOVE DIAGRAM IS FOR BEFORE TRAINING SCENARIO



## ABOVE DIAGRAM IS AFTER TRAINING SCENARIO

### The Logical Explanation of the Efficiency of the Algorithm

Let's try to perceive the algorithm mathematically.
First let's break down the algorithm into logical pieces.
For **Before Training Pathway,**
It has,

1. Input Module.
2. Search module for pattern.
3. Clustering module.
4. Decision making module.
5. Output module.

As shown in the F-Section, the algorithm has two possible pathways of execution and hence the above points are grouped under **Before Training.**

For **After Training Pathway,**
It has,

1. Input Module.
2. Search module for patter search.
3. Clustering module.
4. Output Module.

So after seeing each of the pathway we may now write the best possible complexity for each of the module.

So,

- **For Input** – we have complexity of **O(1).**
- **For Search** – we have best complexity of **O(n log n)** (for quick sort[11]).
- **For Clustering** – as far as the implementation of this paper, we have complexity of **O(n).**
- **For Decision making module** – as far as the implementation of this paper, we have complexity if **O($n^2$).**
- **Output Module** – we have **O(n).**

Seeing the above we can say that we have maximum complexity of **O(n$^2$)** (till we further optimize the decision making module) for the **Before Training Pathway.** And **After training** we have maximum complexity of O(n log n) (if we use the quick sort for search).This will result in a significant increase in the speed and performance of the algorithm since the complexity of time decreases from **O(n$^2$)** to **O(n log n)**. And hence the mathematical explanation for its efficiency.

## SOME POINTS TO BE NOTED

- This is a gateway algorithm and only serves the purpose of toggling the learning capability of the already existing algorithms.
- This algorithm may be slow at the initial stages, but will eventually mature into faster algorithm as it gains the training by the possible real-time scenarios.
- The search algorithm used here is the one which will be a deciding factor for this algorithm. Since the pattern extraction is one of the most crucial aspects, so we need the best search algorithm to be used for the pattern extraction. The performance will depend on the looping structure and the memory taken by the variables in the search algorithm. So this particular thing of using a better search algorithm needed to be used should be kept in the mind.
- The algorithm will work better if used with best fit algorithm. Since the design of this algorithm is done with keeping in mind the best fit algorithm, it is suggested to be used with the best fit algorithm.
- The algorithm need to have certain initial training in that particular scenario before it can be deployed for usage in that scenario so that it gets a head start and have some data with it to make it faster or improve the performance.

### The Pattern

The pattern which is discussed throughout the paper is of this form.

The abbreviation used is given in the abbreviations section, but for the sake of convenience is also defined below.

- **ID-**Identification number as stored in the database as one of the attribute of server/VM.
- **VM-** Virtual Machine.
- **PM-** Physical Machine.
- **HP-** High Priority.
- **LP-** Low Priority.
- **HC-** High Consumption.
- **LC-** Low Consumption.
- **n-** nth number in the loop.

➢ The Pattern: -

```
 Server-(1):-        {[[VM-(ID (1))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
(LP/LC)]],
                      [[VM-(ID (2))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]],
                      [[VM-(ID (3))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]],
                      .
                      .
                      .
                      .,
                      [[VM-(ID (n))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]]},
Server-(2):- {[[VM-(ID (1))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],
                      [[VM-(ID (2))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]],
                      [[VM-(ID (3))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]],
                      .
                      .
                      .
                      .,
                      [[VM-(ID (n))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or
                      (LP/LC)]]},
```

Server- (3):- {[[VM-(ID (1))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],

[[VM-(ID (2))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],
[[VM-(ID (3))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],
.
.
.
.,
[[VM-(ID (n))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]]},
.
.
.
.,

Server- (n):- {[[VM-(ID (1))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],

[[VM-(ID (2))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],
[[VM-(ID (3))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]],
.
.
.
.,
[[VM-(ID (n))]-[From which Cluster (HP/LC) or (HP/HC) or (LP/LC) or (LP/LC)]]}.

## FUTURE IMPROVISATIONS

- The decision algorithm can be made more efficient.
- The overall algorithm needs optimizations.
- We can fabricate a specialised algorithm for searching the pattern, which may further reduce the complexity.
- We may use the datasets to pre train the algorithm instead of training it in the scenario which will save some time.
- Indexed search may be for pattern extraction.

## CONCLUSIONS

After testing the algorithm by an implementation, through java code, in unix[10] environment, it was found that the algorithm works perfectly as expected and also over extended period, and with more number of dataset input, as it learns, it becomes faster and faster. Also since the algorithm is just an external interface which decides either to go to the calculation part or use the past experiences. Due to this like human brain as the experience increases the time complexity is expected to decrease, and eventually lead to minimum time complexity achieved till date, since after the exhaustive learning, and with huge experience, it will be just take input, read database and output. Which can further more be optimized by efficient search algorithms.

## REFERENCES

[1] Power and resource-aware virtual machine placement for IaaS cloud by Madnesh K. Gupta, Ankit Jain, TarachandAmgoth.

[2] https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/.

[3] https://www.geeksforgeeks.org/program-best-fit-algorithm-memory-management.

[4] https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm

[5] Best Searching Techniques https://www.geeksforgeeks.org/searching-algorithms/

[6] https://ccrma.stanford.edu/guides/planetccrma/Unix.html

[7] https://medium.com/human-in-a-machine-world/quicksort-the-best-sorting-algorithm-6ab461b5a9d0

[8] https://en.wikipedia.org/wiki/Curse_of_dimensionality

## AUTHOR'S BIOGRAPHY

Sourav Modak

He is currently a final year B.Tech student at SRM Institute of Science and Technology Department of Computer Science and Engineering, Chennai, pursuing guided research work under S. Nagadevi.

S. Nagadevi

She is currently an Assistant professor in Department of Computer Science and Engineering in SRM institute of science and technology, Chennai and pursuing research work in cloud computing.