# Chapter 3

**System and Architecture**

*In this chapter we introduced our thesis Methodology, and Problem Statement. In section 3.2 we discussed about problem statement, and mathematical presentation of the problem; in section 3.3 we discussed about overview of our proposed work 3.4. we should give a short discussion of this chapter.*

### Introduction

Load Balancing is the process to divide the workload among different available resource in cloud and equalize with virtual machine to achieve the performance of system. Sometimes several resources are heavy loaded, lightly loaded or idle for execution of tasks. There for task are removed from heavily loaded resources and rescheduled them to lightly loaded resources. We have proposed a modified ant colony algorithm for minimizing the overall response time.

### Problem Statement

Until recently the main works on load balancing assumed solid nodes. Many instances of Cloud computing, as outlined herein, wherever dynamic and heterogeneous systems are necessary to produce on demand resources or services. The Amazon EC2, dynamic load balancing is handled by replicating instances of the precise middleware platform for internet services. This is often achieved through a traffic analyzer that tracks the time taken to method a shopper request. New instances of the platform are started once the load will increase on the far side predefined thresholds. Therefore, combos of rules impose the circumstances and answer for load balancing. Because the systems increase in size and quality, these rule sets become unwieldy and it should not be potential to take care of a viable observation and response cycle to manage the procedure work. In short, the dimensions of those systems could exceed the capabilities of connected meta systems to take care of a sufficiently agile and with efficiency organized load balancing (or general management) rule set. Once such a lot of management rules are outlined at intervals a system, there are probably to be conflicts amongst the rules; interactions and impact are normally terribly tough to research. A load balancing system is needed that self regulates the load at intervals the Cloud’s entities while not essentially having to possess full information of the system. Such self-organized regulation could also be delivered through distributed algorithms.

### Proposed Approach

In the natural world, ants of some species (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems.

The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to

many ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve

In our proposed work consider a minimum completion time of the task in each virtual machine. In existing ant colony algorithm only consider the virtual machine which has a minimum number of high priority task. In our modified ant colony algorithm, first we count the number of tasks and then sort them according their size in ascending order and then allocate them to vm and second is minimum completion time of task on each virtual machine.

#### Flow Diagram of Ant Behavior Approach

Following figure 3.3 shows the flow diagram of the proposed algorithm. In this diagram, shows where the proposed methodology is used in cloud computing.

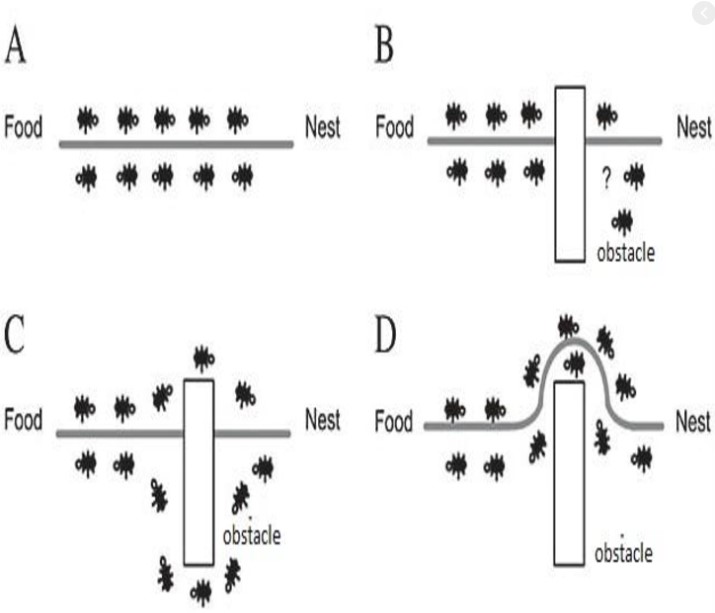


Fig: Behavior of ant colony

* + 1. **The algorithm of Ant Colony are shown as below:**

|  |
| --- |
| **Algorithm: Modified Ant colony behavior inspired load balancing algorithm** |
| Step 1: Initialize the pheromones of all VMs. |
| Step 2: Place all ants at the starting VMs randomly |
| Step 3: Every ant chooses the VM for the next task according to formula. |
| Step 4: When an ant completes its tour, update the pheromone according to formula |
| Step 5: If all the ants end their trip, continue to Step6; otherwise, repeat Step3 |
| Step 6: Nc = Nc + 1, calculate the makespan of each ant and reserve the current optimal solution |
| Step 7: Judge if it satisfies the iterative condition Nc > Ncmax, If it satisfies, end the iteration and output the best solution, else return to Step2 until satisfy the iterative condition |

Table 3.2. 4: The Ant colony behavior algorithm

#### Mathematical Representation

###### Initialize pheromone of VMj At the beginning, ants are distributed on VMs randomly, and then it will initialize the VMj pheromone value based on:

###### (0) = pe\_ × pe\_+ vm\_ ………………..(1)

###### Where pe\_numj is the number of VMj processor, pe\_mipsj is the MIPS (Million Instructions Per Second) of each processor of VMj and the parameter VM\_bwj that is related to the communication bandwidth ability of the VMj.

###### B. The rule of choosing VM for next task The k-ant chooses VMj for next task with a probability that is defined as:

###### 

##### Where

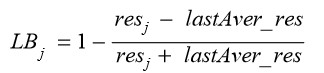
##### • τj(t) is the VMj pheromone value at time t.

##### • EV j is the computing capacity of VMj, it is defined as follows:

##### EVj = pe\_numj × pe\_mipsj + vm\_bwj (2)

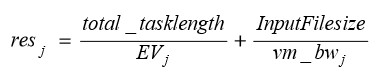
Where pe\_numj is the number of VMj processor, pe\_mipsj is the MIPS of each processor of VMj and the parameter VM\_bwj that is related to the communication bandwidth ability of the VMj.

• LBj is the load balancing factor of VMj, to minimize the degree of imbalance, which is defined as follows:



Where lastAver\_res is the average execution time of the virtual machines in the last iteration of the optimal path, and resj is the expected execution time of the task in the VMj, which is defined as follows:

Processing time of VM:



Where total\_tasklength is the total length of the tasks that have been submitted to VMj, and InputFilesize is the length of the task before execution. • α, β and γ are three parameters that control the relative weight of the pheromone trail, the computing capacity of VMs and the load balancing factor of VMs.

Once some VMs are loading heavy, it becomes a bottleneck in the cloud and it influences the makespan of a given tasks set. Therefore we define the load balancing factor LBj in the ant algorithm to improve the load balancing capability, and the bigger LBj of VMj should be chosen with high probability, that means the comprehensive ability of VMj is power now, and then it is high desirable

C Phenomenon Updating

Let τj(t) be the intensity of VMj pheromone at time t. The pheromone update is given by (7) 

Where ρ ∈(0, 1] is the pheromone trail decay coefficient. The greater the value of ρ is, the less the impact of past solution is. The value of ∆τj is defined as follows: When an ant completes its tour, the local pheromone updating is applied on the visited VMs, and the value of ∆τj is given by



Where Tik is the shortest path length that searched by ik ant at i-th iteration. When an ant completes its tour, if it finds the current optimal solution, it can lay a larger intensity of the pheromone on its tour and the global pheromone updating is applied on the visited VMs, and the value of ∆τj is given by



Where Top is the current optimal solution, and D is the encouragement coefficient.

#### 3.3.2 The pseudo code of our proposed algorithm are as below:

|  |
| --- |
| **begin**  **Initialize the number of ants, datacenter, server,**  **while (has\_task) do**  **Time= estimate\_execution\_time(task)**  **Distance\_matrix = calculate\_distance(task)**  **cost\_matrix = generate\_cost\_matrix(Time,Distance\_matrix,cost\_function)**  **Server=find\_best\_server\_using\_ant\_colony(Time,Distance\_matrix,cost\_matrix)**  **Assign\_task(task,server) If(no\_more\_task) : Wait\_for\_task()**  **Endif**  **If(machine\_is\_idle ) :**  **Server\_is\_Auto\_off();**  **Endif**  **If(task\_completed) : relize\_the\_server**  **Endif**  **end while**  **end** |

Fig: pseudo code of load balancing based on ant colony

#### 

#### Summary of my Proposed pseudocode

#### In Ant colony we see that they search food from source to destination by applying updating their pheromone. They use the shortest path to find this food from source to destination like as travelling salesman problem. In my proposed algorithms I used this ants idea for calculating cost for minimum number of task, efficient execution time and auto scaling when task is idle.

#### i) At first initialize a number of tasks, server, ants and datasender.to create two types of datasets one for the server and another for the task and take input from this

#### ii) If have some tasks are free in our system then our system is working. Our system is working in multithreading.

#### iii) Calculating execution time and distance matrix ,cost matrix, depending on task. They are working two different function that is estimate\_execution\_time and calculate\_distance

#### iv) They are working until find\_best\_server\_using\_ant\_colony .it is depending on tasks, execution time, distance matrix and cost matrix

#### v) if no more tasks in this system, then wait for new task until our budget in on.

#### vi) if our budget is low or finished or machine is idle then works on auto scaling method. That’s why our machine is auto off

#### vii) Finally calculating the execution time, number of servers, number of tasks and number of virtual machines.

#### 

#### Expected Outcome from the Work

Better utilization of resources and task take minimum completion time so that the performance of system will be improved. Our algorithm finding optimal least underloaded virtual machine for increasing the system performance.

### Discussion

This chapter showed a clear description about the methodology, algorithm, we used for the various analysis of our thesis. Here also showed an overview of Analytical approach, and the Flow Diagram and pseudo code and details.