Honey Bee Behavior Inspired Particle Swarm Optimization Technique for Adaptive Resource Allocation

B.PRASANALAKSHMI
Associate Professor, Department of CSE,
Professional Group of Institutions,
Palladam, India.
bplakshmi@ieee.org

A.KANNAMMAL
Professor,Deparmtent of Comuputer Application
Coimbatore Institute of Technology,
Coimbatore, India
kannaphd@yahoo.co.in

Abstract— Cloud computing is one of the rapidly improving technologies. It provides scalable resources needed for the applications hosted on it. As cloud-based services become more dynamic, resource provisioning becomes more challenging. The QoS constrained resource allocation problem is considered in this paper, in which customers are willing to host their applications on the provider's cloud with a given MA requirements for performance such as throughput and response time. Since, the data centers hosting the applications consume huge amounts of energy and cause huge operational costs, solutions that reduce energy consumption as well as operational costs are gaining importance. In this work, we propose an energy efficient mechanism that allocates the cloud resources to the applications using HBF-PSO framework.

Keywords- ***Resource Allocation, Ant colony framework,*** Cloud computing, Intelligent Agents component.

I. Introduction

Cloud computing provides much utility based on the pay –as-you go model. Cost and efficiency related issues were discussed by many researchers , which will no longer be a problem. This contribution also includes increase in efficiency, reduce energy consumption and operational cost. Load on servers in the data center changes dynamically. In order to tackle this dynamism with balanced load , the resources are allocated the services so that minimum number of servers will be used for hosting the services. Load balancing is a technique to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, and minimize response time. Load balancing can be useful when dealing with redundant communications links. For example, a company may have multiple Internet connections ensuring network access even if one of the connections should fail. A failover arrangement would mean that one link is designated for normal use, while the second link is used only if the first one fails. With load balancing, both links can be in use all the time. A device or program decides which of the available links to send packets along, being careful not to send packets along any link if it has failed. The ability to use multiple links simultaneously increases the available bandwidth. Major telecommunications companies have multiple routes through their networks or to external networks. They use more sophisticated load balancing to shift traffic from one path to another to avoid network congestion on any particular link, and sometimes to minimize the cost of transit across external networks or improve network reliability. The notion of complex collective behaviour emerging from the behaviour of many relatively simple units, and the interactions between them, is fundamental to the field of artificial life. The growing understanding of such systems offers the prospect of creating artificial systems which are controlled by such emergent collective behaviour; in particular, we believe that the exploitation of this concept might lead to completely new approaches for the management of distributed systems, such as load balancing in telecommunications networks.

In such networks, Calls between two points are typically routed through a number of intermediate switching stations, or nodes; in a large network, there are many possible routes for each such call. It is thus possible to relieve actual or potential local congestion by routing calls via parts of the network which have spare capacity. Load balancing is essentially the construction of call –routing schemes which successfully distribute the changing load over the system and

Paper Type: Article
Corresponding Author: Prasanalakshmi.B, email: bplakshmi@ieee.org

minimise lost calls. Of course it is possible to determine the shortest routes from every node to every other node of the network. In this way the average utilisation of nodes will be minimised, but this is not necessarily the ideal way to avoid node congestion, as this has to do with how the traffic on the network is distributed. Controlling distributed systems like these by means of a single central controller has several disadvantages. The controller usually needs current knowledge about the entire system, necessitating communication links from every part of the system to the controller. These central control mechanisms scale badly, due to the rapid increase of processing and communication overheads with system size. Failure of the controller will often lead to failure of the complete system. There is the additional practical commercial requirement that centrally controlled systems may need to be owned by one single authority.

II. PARTICLE SWARM OPTIMISATION AND BEE ALGORITHM-LOAD BALANCING

To bring about such a load balanced solution , optimization should also brought about with minimum cost and power to attain faster resource allocation. Swarm-based optimisation algorithms (SOAs) mimic nature's methods to drive a search towards the optimal solution. SOAs include the Bee Colony Optimisation algorithm , the Genetic Algorithm (GA) [3] and the Particle Swarm Optimisation (PSO) algorithm [4]. Mathur et al [6] describe a hybrid of the ACO algorithm and the GA for continuous function optimisation. Individual solutions in a population are viewed as "particles" that evolve or change their positions with time. Each particle modifies its position in search space according to its own experience and also that of a neighboring particle by remembering the best position visited by itself and its neighbors, thus combining local and global search methods [4]. There are other SOAs with names suggestive of possibly bee-inspired operations [7-10]. However, as far as the authors are aware, those algorithms do not closely follow the behaviour of bees. In particular, they do not seem to implement the techniques that bees employ when foraging for food.

A colony of honey bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources [7,8]. A colony prospers by deploying its foragers to good fields. In principle, flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more bees, whereas patches with less nectar or pollen should receive fewer bees [9,10]. The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees [8]. When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold (measured as a combination of some constituents, such as sugar content) deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance" [7]. This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) [7,10]. This information helps the colony to send its bees to flower patches precisely, without using guides or maps. Each individual's knowledge of the outside environment is gleaned solely from the waggle dance. This dance enables the colony to evaluate the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it [10]. After waggle dancing on the dance floor, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. More follower bees are sent to more promising patches. This allows the colony to gather food quickly and efficiently. While harvesting from a patch, the bees monitor its food level. This is necessary to decide upon the next waggle dance when they return to the hive [10]. If the patch is still good enough as a food source, then it will be advertised in the waggle dance and more bees will be recruited to that source.

In complex and large systems, there is a tremendous need for load balancing. For simplifying load balancing globally (e.g. in a cloud), one thing which can be done is, employing techniques would act at the components of the clouds in such a way that the load of the whole cloud is balanced. For this purpose, we are discussing honeybee foraging algorithm This algorithm is derived from the behavior of honey bees for finding and reaping food. There is a class of bees called the forager bees which forage for food sources, upon finding one, they come back to the beehive to advertise this using a dance called waggle dance. The display of this dance, gives the idea of the quality or quantity of food and also its distance from the beehive. Scout bees then follow the foragers to the location of food and then began to reap it. They then return to the beehive and do a waggle dance, which gives an idea of how much food is left and hence results in more exploitation or abandonment of the food source.

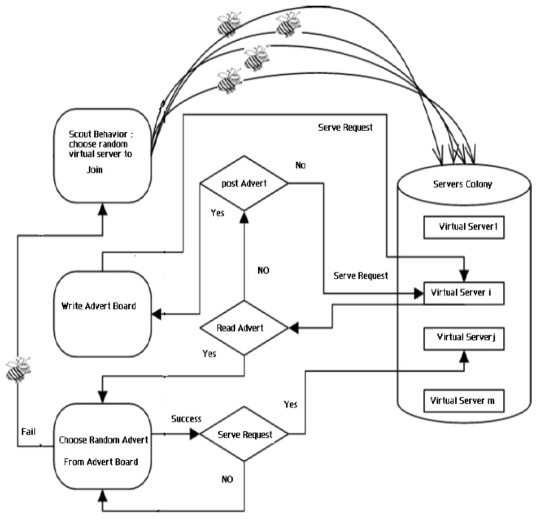

Fig :01 Server Allocations by Foraging in Honey bee technique (adopted from [5])

In case of load balancing, as the webservers demand increases or decreases, the services are assigned dynamically to regulate the changing demands of the user. The servers are grouped under virtual servers (VS), each VS having its own virtual service queues. Each server processing a request from its queue calculates a profit or reward, which is analogous to the quality that the bees show in their waggle dance. One measure of this reward can be the amount of time that the CPU spends on the processing of a request. The dance floor in case of honey bees is analogous to an advert board here. This board is also used to advertise the profit of the entire colony. Each of the servers takes the role of either a forager or a scout. The server after processing a request can post their profit on the advert boards with a probability of pr. A server can choose a queue of a VS by a probability of px showing forage/explore behavior, or it can check for advertisements (see dance) and serve it, thus showing scout behavior. A server serving a request, calculates its profit and compare it with the colony profit and then sets its px. If this profit was high, then the server stays at the current virtual server; posting an advertisement for it by probability pr. If it was low, then the server returns to the forage or scout behavior. It is diagrammatically given in fig. 01.

III. OUR SYSTEM ARCHITECTURE

The main aim of our resource allocation is to allocate the online service requests for applications which are CPU and memory intensive. The participants in the architecture include:

Users/Brokers: Users or brokers acting on their behalf submit service requests to the cloud via cloud controller for processing.

Cloud Controller: It acts as the interface between the cloud service provider and external users/brokers. It acts similar to the Queen in the bee colony.

Virtual Machines(VMs): This is where the applications of customers will be deployed. We can dynamically create, start, stop and migrate these VMs depending on our requirement, from one physical machine to another.

Physical Machines: These are the physical computing servers that will provide hardware infrastructure for creating virtual machines. The power consumption of each server in the data center along with the resource capabilities such as CPU processing power and primary memory before admitting them into cloud. We store this information consisting of Node Id, Processing Power, Memory and Power Consumption in a table.

We sort the nodes in the descending order of the following metrics.

$$PPW = \frac{processing\ power\ of\ the\ node(GHz)}{Power\ consumption\ of\ CPU(Watts)}$$

And

$$MPW = \frac{Memory\ capacity\ of\ node(GB)}{Power\ consumption\ of\ memory(Watts)}$$

Where, PPW = Processing Power per Watt, MPW = Memory Consumption per Watt

The power consumption is given as the power consumed by CPU or memory when their utilisation is 100% which is measured before admitting the node in the cloud. We assume that the power consumed by all the remaining components of a node are same for all the nodes. We consider that, all the nodes are having the same network connectivity and access to the shared persistent storage space that stores the VM disk images.
We store this information in a table called Available Resource Table with Available node's Id, current power consumption and remaining capacity updated by the gathered information from bee agents. This table is represented as an Array List with a pointer (Allocation ptr) being pointed to the node on which next service request is deployed. This pointer will be adjusted by different bee agents. The functionality of different bee agents are given below:

A. Cloud Controller & Queen Bee:

The requests from the customers consisting of the following, are given to the controller.

(i) Throughput THPUT) (In %)

(ii) Avg. Response Time(RTIME)

(iii) Application Code

(iv) Operating System

Cloud controller maintains a queue(Q) for storing the service requests for hosting the applications. It enqueues each of the service request received, in this queue. It generates the tester, scout, cleaner and worker bees periodically. The movement of these bee agents is modeled in the following way. Each bee except Queen & Worker maintains a Visited Node list which is initially empty. Each node in the cloud maintains a list of neighbouring node's information. Whenever an bee reaches a node, it updates the controller about the current utilisation and randomly chooses an unvisited neighbouring node. When all the nodes are covered, it makes the Visited Node list empty and continues again in the same way. We can change the number of bees that will be produced so that it will yield better results depending on our requirement. The next subsection describes the method used by worker bees for accepting or rejecting the service requests.

B. Worker Bee:

Whenever a service request received in the queue, one of the worker bees creates a VM with a specific CPU processing power and memory etc, if accepted. So, worker bees are always looking in the queue to check if there are some pending requests to be processed. If such a request is found, it dequeues the request. Since most of the CPUs are work conserving, we are creating a VM like Amazon Standard Instance [11] with specific CPU processing power and memory. Depending on the load, more intensive applications can use the resources of the other VMs having less load. The worker bee is only responsible for deploying the request on a VM. Load balancing decisions are taken by tester bee. After deploying, it creates a monitor agent that monitors the hosted application. The MA looks as in table 1.

Table 1: The schema for Monitor Agent

VM name	**Operating system**	**Space required**	**Type of application**

The monitor agent calculates the Avg. response time and throughput of the hosted application by continuously monitoring it. It passes this information to the hypervisor on that host in the form of a variable. When the tester bee queries the node for utilization information, hypervisor will send this value along with utilisation information .when the value is 0, there is no need for balancing, if the value is 1 , the tester bee will try to allocate this VM to a better node that have more available resources than current node among the currently running nodes and will not try to wake up a new node. If it's value is 2 then, it will wake up the next power efficient standby node if needed as it is required to handle the heavier loads. Depending on these values, the tester bee balances the load.

C. Tester Bee:

The main job of the tester bees is to get the utilization and power consumption information from each of the node and to update the available node's list as in table 3. It also takes the load balancing decisions. Load balancing on a node will be taken care by the hypervisor. We provide the algorithms for load balancing of various hosts in the cloud. We consider that the utilisation of 80% of CPU and 80% of memory of a node is considered to be desirable utilization and above 90% is considered to be peak. However, we can change them according to our requirement. We get the utilisation of CPU and memory information by probing the proc file system in Linux and resource utilisation in Windows. The tester bee will update this information in the Available nodes list along with the current power consumption shown by power top utility[12]. We have used the existing utilities for measuring power consumption because the existing power models, based on the utilisation, are not accurate at measuring it[13]. In order to improve the process of creating the VMs, we put three nodes below the current allocation node in standby mode and create the VMs with specific operating systems to be able to configure the application on them quickly. We prefer standby mode to hibernate mode because it requires less time to wake up a node from standby mode than from hibernation. We consider that the overhead of VM migration is negligible. The next subsection describes about the node discovery and registration which will be done by scout bees.

D. Scout Bee:

The aim of scout bee is to discover the newly added cloud nodes providing computing and memory services. When such a new node is found, it adds it to the available resource table(ART) as in table 4. It is done through node registration. Node Registration:

The node which wants to join the cloud must have to inform one of the nodes in the cloud by sending a request. Each request is being analysed by the scout bee and placed in the Data analysis table(DAT) as in table 2.When the scout bee visits a node and finds a request for joining, it registers the node with a unique id. It updates this information in the available node's list(ANL) with resource utilisation and power consumption and places this node in appropriate position in the list. Whenever a node is added by administrator in the event of resource scarcity, then the registration will be done by cloud controller node. We assume that the node that is completing the registration for newly joining node will not get failed during the registration process. If it fails before registration, new node will contact another node in cloud.

The various schema include:

Table 2: Schema for DAT:

Client_id	**Node_id**	**Space requirement**	**OS**	**Memory required**	**Power consumption estimate**

Table 3: Schema for ANL:

Node_id	**Utilisation**

Table 4: Schema for ART

Node_id	**Remaining power**	**Remaining space**

E. Cleaner Bee:

It maintains the available resource table by removing the unavailable resources from the list. When this agent reaches a node in cloud and if it didn't respond to that agent for a specific time duration, then it assumes that this node is failed and it takes necessary actions for recovery and will remove this node information from the available node's list. It removes the cloned VMs from the nodes if they are under utilised or if the performance is more than required. It also removes the VMs from the nodes whose service agreement get expired. It sends an alert to the customer before some days so that the service can be renewed if they needed. The algorithm to be proposed is given here:

IV FUTURE DIRECTIONS AND CONCLUSIONS

We have investigated several cloud computing test beds and found that cloudsim simulator[2] is suitable for testing the proposed mechanism. So, we are in the process of implementing the proposed mechanism on the cloudsim toolkit and the performance evaluation is in progress. We also plan to improve this by incorporating the load prediction and usage models so that this can be applied to real cloud environments.

We have proposed a power efficient, agent based solution for allocation of resources to cloud applications. We believe that this mechanism is very flexible and can be extended with improvements, as the solution modules are modelled as independent intelligent agents. We can incorporate additional functionalities in any of these HBF-PSO agents.

```
Begin
  Initialise parameters
    Analyse Available Node List(ANL)
        Number of servers($N_S$) – Resource
        For all resources($N_S$)
          Space and Memory utilisation of each servers
        Start_queue
          Receiving request from users ($N_R$)
            For all ($N_R$)
              Store data in Data Analysis Table (DAT)
             Start allocation of VM
               If VM available
                   Allocate VM
               Else
                   Check for standby / Power off VM
               Else
                   Create new VM
              Update VM allocation data to Monitor Agent(MA)
             Update ANL,ART
            Update Computing and Memory requirements serviced
        End $N_R$
        Check for node balance
        If balanced then
          Wait for another updated request queue
            If memory and space requirement could be satisfied
                      Allocate
            Else
                      Notify_admin(Resource scarcity)
            Else
               Balance the server load
                     Migrate VM / Clone VM
               If VM cloned
                  Check lease time > Criteria
                      Check utilization < satisfactory limit
                        Remove VM
                      Else
                       Criteria > Lease time > Grace period
                          Notify_User (end of service)
                      Else
                        If service time over then
                         Remove VM
                      End VMcloned
        End_balance
       End_queue
```

REFERENCES

[1] Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.

[2] Dorigo M and Stützle T. Bee Colony Optimization. MIT Press, Cambridge, 2004.

[3] Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison-Wesley Longman, 1989.

[4] Eberhart, R., Y. Shi, and J. Kennedy, Swarm Intelligence. Morgan Kaufmann, San Francisco, 2001.

[5] Martin Randles, Enas Odat, David Lamb, Osama Abu- Rahmeh and A. Taleb-Bendiab, ”A Comparative Experiment in Distributed Load Balancing”, 2009 Second International Conference on Developments in eSystems Engineering.

[6] Mathur M, Karale SB, Priye S, Jayaraman VK and Kulkarni BD. Bee Colony Approach to Continuous Function Optimization. Ind. Eng. Chem. Res. 39(10) (2000) 3814-3822.

[7] Von Frisch K. Bees: Their Vision, Chemical Senses and Language. (Revised edn) Cornell University Press, N.Y., Ithaca, 1976.

[8] Seeley TD. The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies. Massachusetts: Harvard University Press, Cambridge, 1996.

[9] Bonabeau E, Dorigo M, and Theraulaz G. Swarm Intelligence: from Natural to Artificial Systems.Oxford University Press, New York, 1999.

[10] Camazine S, Deneubourg J, Franks NR, Sneyd J, Theraula G and Bonabeau E. Self-Organization in Biological Systems. Princeton: Princeton University Press, 2003.

[11] “http://aws.amazon.com/ec2/,” accessed on Nov 23, 2010.

[12] “http://www.lesswatts.org/projects/powertop/,” accessed on Jan 25, 2011.

[13] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas, “Demystifying energy consumption in grids and clouds,” in Green Computing Conference, 2010.

[14] “http://www.cloudbus.org/cloudsim/,” accessed on Jan 13, 2011.