INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

IMPROVING PERFORMANCE IN HINT-BASED EXECUTION OF WORKLOADS WITH EFFECTIVE GATEWAY

K. Suganthi[1], K. Sathish Kumar[2]

[1]PG Scholar,PGP College of Engineering and Technology,Namakkal
ksugvani@gmail.com
[2]Assistant Professor,PGP College of Engineering and Technology,Namakkal
Address:No. 42 A4/50, PVR Street, Mohanur Road, Namakkal – 637001.,Tamilnadu, India
Mobile:97 89 15 74 38

ABSTRACT

Cloud computing allow the transparent access to diverse physical resources which are made available in the form of services (IAAS). Transparent access are achieved through the effective Nefeli gateway and user provided Hints. Nefeli is a virtual infrastructure gateway that is capable of effectively handling diverse workloads of jobs in cloud environments. Hints are passed through the Nefeli gateway. Nefeli helps to avoid bottlenecks and achieve better performance within the cloud environment. In this paper we propose two algorithms. These are Backfilling Algorithm and Genetic Algorithm. Backfilling is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order. Priority order is "highest priority first". Genetic algorithm is a random searching method. The advantages of genetic algorithm is a balanced scheduling strategy of VM resources in cloud computing environment by considering the current states and historical data, this method will compute in advance its influence over the entire system.

Keywords: IAAS cloud, Backfilling Algorithm, Genetic Algorithm, Virtual infrastructure, VM Migration.

1. INTRODUCTION

A cloud is a type of parallel and distributed system a collection of interconnected and virtualized computer that are dynamically provisioned and presented as one or more unified computing resources based on service level agreements established through negotiation between the service providers and consumers. Computing "clouds" allow for the transparent access to diverse physical resources which are made available in the form of services. In general, cloud services can be classified according to the level at which they function as **a)** Software as a Service (*SaaS*), **b)** Platform as a Service (*PaaS*) and **c)** Infrastructure as a Service (*IaaS*). In this paper, we focus on *IaaS*-clouds that exploit the use of virtual machines (VMs) to deploy computing systems on-demand. We examine the effective deployment of VMs so that multiple and diverse workloads can be efficiently handled by the physical infrastructure.
Genetic algorithm is a random searching method that has a better optimization ability and internal implicit parallelism. It can obtain, and instruct the optimized searching space and adjust

the searching direction automatically through the optimization method of probability. With the advantages of genetic algorithm, this project presents a balanced scheduling strategy of VM resources in cloud computing environment. By considering the current states and historical data, this method will compute in advance its influence over the entire system.

Backfill is a scheduling optimization which allows a scheduler to make better use of available resources by running jobs out of order. It prioritizes the jobs in the queue according to a number of factors and then orders the jobs into a 'highest priority first' sorted list. It starts the jobs one by one stepping through the priority list until it reaches a job which it cannot start. Backfill operates based on this 'earliest job start' information.

This Section presented a brief introduction to this paper. Section II covers the background of this paper. Section III presents the Proposed algorithm in detailed view. Section IV presents the Implementation details of this paper. Section V presents the Conclusion and Future Works..

2. BACKGROUND

The background of this paper describe the design, implementation, and evaluation of Nefeli, a cloud gateway, that seeks to overcome contention of VMs for workloads consisting of diverse tasks executing in a cloud. Nefeli accepts requests from users for execution of particular workloads in the cloud and deploys these workloads within the cloud. Nefeli performs intelligent placement of VMs onto physical nodes using user-provided hints *Nefeli* adds a layer between the user and the infrastructure providing IaaS-cloud services, shown in Figure 1. Nefeli must interface with the lower level cloud services that handle the VM lifecycle and perform fundamental administrative tasks. This interface, denoted as Cloud *API*, allows us to query for specific aspects of the hardware resources as well as manage the VM deployment and migration.

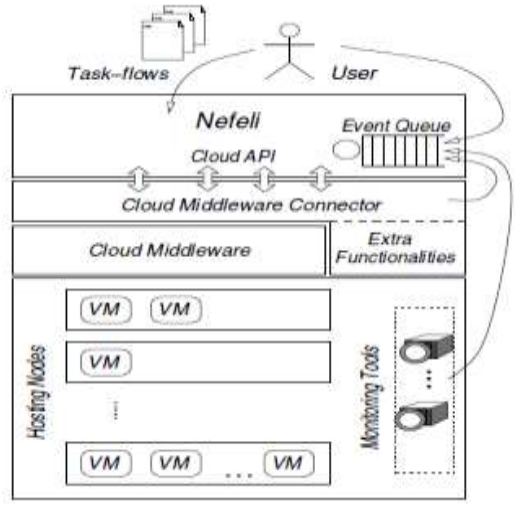

Fig.1: Nefeli Structured layout

During operation, Nefeli needs to obtain the following information:

• **Physical node properties***:*

These properties include free memory, total memory, CPU utilization, the name/ID of each hosting node and the amount of free disk space.

• **The current status of each VM:**

In our approach each VM may find itself in either *STAGING* or *RUNNING* state. A VM is considered to be *STAGING* when management operations such as disk image copying during a VM migration do not permit the VM to run.

• **VM properties:**

These are similar to the properties acquired for hosting nodes; VM properties include the memory usage and the disk space reserved in each virtual machine. Also the *IP*-address of each VM should be provided by the cloud *API* so as to be forwarded to the user as a VM access reference point. VM deployment operations are handled through the cloud *API* of Figure 1 and include:

• **Spawn a new VM.**
• **Shutdown a VM.**
• **Migrate a VM**.

For this operation, the names/IDs of the hosting nodes are needed.
While part of the interaction Nefeli has with the physical infrastructure can be provided by a cloud middleware there are cases where additional functionality is need For instance Open Nebula v.1.2.0 does not expose all host related information it gathers. In such cases, we have to realize any missing functionality and incorporate it in the "Cloud Middleware Connector" component (Fig1).
Nefeli has the role of an IaaS-cloud gateway. Users contacting Nefeli request virtual infrastructures created by instantiating sets *of* VMs.
In Figure 2, we present the environment in which the main components of Nefeli operate. The Deployer is the component that keeps track of all active task-flows. This component up- dates a list of all submitted task-flow descriptions upon arrival of external events. Event arrival also causes the interaction between Deployer and Planner. The Deployer provides a list of task-flows, constraints and the respective weights, and the Planner produces a single deployment profile as if it were a
single task-flow.
The profile produced is applied using calls to the cloud API. The event-based mechanism that triggers the Deployer's operation is not limited to events caused by new task-flow submission. This user generated event is not the only one required to handle task-flows, Nefeli must also be informed of a task-flow ending. Task-flow termination events come from either the user or some component of the task-flow itself.
The overall goal of Nefeli is to make choices regarding the deployment profile based on the user's needs and the system's performance. To this end, we have extended the capabilities of the notification mechanism so that Nefeli receives any kind of signal that will assist in achieving its goal.

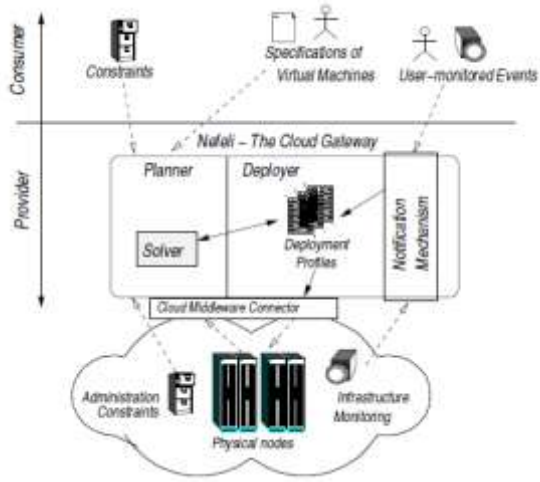

Fig.2 Nefeli Environment

NEFELI OPERATIONAL MODEL

Figure 3 shows the key steps followed starting from the user input until we reach a VM-to-host mapping, termed deployment profile. With V being all the VMs to be deployed and H the set of physical nodes, a profile M is a function from V to H (M : V 7→ H). Nefeli chooses, out of all possible profiles Mall, one that best suits the constraints expressed for the task flow at hand.
Profile production uses information gathered not only from the user hints but also from the cloud administrator and the physical infrastructure. Combining the user-provided constraints with the VM specifications, as described in the *XML*-document of the results in deployment patterns. These patterns are the outcome of examining user preferences alone and our approach uses them to match VM requirements to physical node resources.

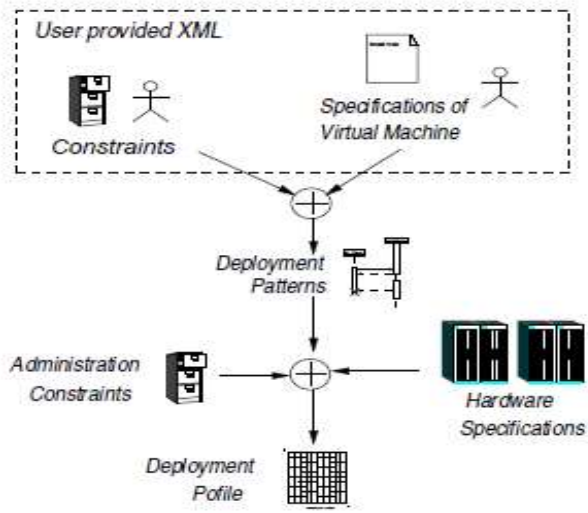

Fig.3: Nefeli Operational model

The Example of the utility function is presented by High Avail Utility Function and Simulated Annealing Algorithm. Simulated Annealing algorithm is used for profile production.
A hint-based VM scheduler that serves as a gateway to IaaS-clouds. Users are aware of the flow of tasks executed in their virtual infrastructures and the role each VM plays. This information is passed to the cloud provider, as hints, and helps drive the placement of VMs to hosts. Hints are also employed by the cloud administration to express its own deployment preferences.
Nefeli combines consumer and administrative hints to handle peak performance, address performance bottlenecks, and effectively implement high-level cloud policies such as load balancing and energy savings. An event-based mechanism allows Nefeli to reschedule VMs to adjust to changes in the workloads served. Our approach is aligned with the separation of concerns IaaS-clouds introduce as the users remain unaware of the physical cloud structure and the properties of the VM hosting nodes. Our evaluation, using simulated and real private Iaas-cloud environments, shows significant gains for Nefeli both in terms of performance and power consumption.

3. PROPOSED METHOD:

BACK FILLING ALGORITHM

The algorithm behind Moab backfill scheduling is straightforward, although there are a number of issues and parameters that should be highlighted. First of all, Moab makes two backfill scheduling passes. For each pass, Moab selects a list of jobs that are eligible for backfill. On the first pass, only those jobs that meet the constraints of the soft fairness throttling policies considered and scheduled. The second pass expands this list of jobs to include those that meet the hard (less constrained) fairness throttling policies. The second important concept regarding Moab backfill is the concept of backfill windows. The figure below shows a simple batch environment containing two running jobs and a reservation for a third job. The present time is represented by the leftmost end of the box with the future moving to the right. The light gray boxes represent currently idle nodes that are eligible for backfill. For this example, lets assume that the space represented covers 8 nodes and a 2 hour time frame.

To determine backfill windows, Moab analyzes the idle nodes essentially looking for largest node-time rectangles. It determines that there are two backfill windows. The first window, Window 1, consists of 4 nodes that are available for only one hour (because some of the nodes are blocked by the reservation for job C). The second window contains only one node but has no time limit because this node is not blocked by the reservation for job C. It is important to note that these backfill windows overlap. Once the backfill windows have been determined, Moab begins to traverse them. The current behavior is to traverse these windows widest

window first (most nodes to fewest nodes). As each backfill window is evaluated, Moab applies the backfill algorithm specified by the back filling policy parameter.

If the **FIRSTFIT** algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
2. The first job is started.
3. While backfill jobs and idle resources remain, repeat step 1.

If the **BESTFIT** algorithm is applied, the following steps are taken: The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.

1. The *degree of fit* of each job is determined based on the back filling metric parameter (processors, seconds, processor-seconds).
2. The job with the best fit starts.
3. While backfill jobs and idle resources remain, repeat step 1.

If the **GREEDY** algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. All possible combinations of jobs are evaluated, and the *degree of fit* of each combination is determined based on the parameter (processors, seconds, processor-seconds).
3. Each job in the combination with the best fit starts.
4. While backfill jobs and idle resources remain, repeat step 1.

If the pre-empt algorithm is applied, the following steps are taken:

1. The list of feasible backfill jobs is filtered, selecting only those that actually fit in the current backfill window.
2. Jobs are filtered according to the priority set by the Bfpriority policy parameter.
3. The highest priority backfill job is started as a preemptee.
4. While backfill jobs and idle resources remain, repeat step 1.

If **NONE** is set, the backfill policy is disabled.

GENETIC ALGORITHM

Genetic algorithm is a random searching method that has a better optimization ability and Internal implicit parallelism. It can obtain, and instruct the optimized searching space and adjust the searching direction automatically through the optimization method of probability. With the Advantages of genetic algorithm, this project presents a balanced scheduling strategy of VM Resources in cloud computing environment. By considering the current states and historical data, this method will compute in advance its influence over the entire system.
The genetic algorithm approach computes the impact in advance, that it will have on the system after the new VM resource is deployed in the system, by utilizing historical data and current state of the system. It then picks up the solution, which will have the least effect on the system. By doing this it ensures the better load balancing and reduces the number of dynamic VM migrations. The approach presented in this project solves the problem of load imbalance and high migration costs. Usually load imbalance and high number of VM migrations occur if the scheduling is performed using the traditional algorithms

STEPS IN THE GENETIC ALGORITHM

1. [**Start**] produce random population of n chromosomes (coding structure can be selected according to the problem domain) .
2. [**Fitness**] calculate the fitness value f(x) of every chromosome in the given population.
3. [**New population**] generate the new population by reiterating the following steps till the

Creation of new population is done.

3.1.[**Selection**] select two parent individuals from the population according to the fitness Value.
3.2.[**Crossover**] by using the crossover probability, generate the new offspring by reforming the parents.
3.3.[**Mutation**] with the probability of mutation, mutate the new child at some positions.
3.4.[**Accepting**] now the new offspring the part of next generation of population [19].

4. [**Replace**] use the new generation as the current generation.
5. [**Test**] if the stopping condition is satisfied then end the algorithm and return the individual with the highest fitness value.
5. [**Loop**] goto step 2.

GENETIC ALGORITHM ANALYSIS

GLOBAL SCHEDULING ALGORITHM

With the benefit of the genetic algorithm in the cloud computing environment for VM resource scheduling, this project presents an unbiased scheduling strategy for VM resources by using the genetic algorithm approach. Starting from the initialization phase in the private cloud environment, we took the optimal solution provided by the genetic algorithm for each VM scheduling request. At the start when there are no VM resources present in the current system, we pick the solution based on the algorithm which uses the computed probability. With the increase in the number of VM resources and system run time, the algorithm computes the influence it will have on the system with the help of current state and the historical data. The algorithm will arrange the VM which needs to be deployed with every physical machine and creates a solution set. After the solution set is ready the strategy chooses the optimal solution which has lower number of VM migrations. The Entire approach procedure is as follows:

Step 1:

In the initial phase there are no VM resources that are present in the system, so no historical information can be obtained. If there is one VM resource that needs to be scheduled, based on the computed probability P (ratio of single VM load to the sum of all VM loads), the algorithm chooses the most suitable, free, smallest loaded physical server and then begins scheduling.

Step 2:

As the number of VM resources in the system starts increasing with the running time, the Algorithm computes the load and variance of every physical node in every solution from the solution set S by using historical and current state of the system.

Step 3:

Then the algorithm uses the genetic algorithm approach to calculate the optimal mapping result for each solution in S. Whichever solution meets the predefined system variance constraints is referred as the best solution.

Step 4:

The algorithm also calculates correspondingly the migration costs divisors of each result in S to attain the best mapping solution;

Step 5:

By going through the cost divisor of each result, the algorithm chooses the one solution Which has the lowest cost as the final scheduling solution and completes the scheduling.

Step 6:

If there is new VM to Schedule then go back to step 2.

We have used the genetic algorithm in every solution and every scheduling to find the best mapping solution. To achieve the load balancing, we can take the best mapping solutions every time. Accumulation of the best solution can make it easier to find the best load balancing in quick time. One time scheduling cannot achieve the best load balancing because due to the load variation over the time. We are able to find the load balancing for scheduling by having a low migration cost.

EXPERIMENT AND ANALYSIS OF GENETIC ALGORITHM

To analyse the behaviour of the genetic algorithm with VM scheduling, the following experiment was carried out. We presume to have 5 physical machines with 15 VMs started on them. We can see the mapping relationship between VMs and physical machines in before section Figure 4. We used the predefined data for

last 15 minutes, which is shown in the table 1. For the whole system condition we assumed the following configuration values.

Population scale is 50 Probabilities, Replication *Pr* = 0.1, Hybridization = *Pc* = 0.9 and variation is self-adaptive probability Theoretically, Hybridization probability can be *Pc* = [0,1] and variation probability *Pm* = (0,1). System load variation constraint .A = 0.5, Stopping condition parameter.

By setting up the above parameters and running the algorithm, we got the following mapping relationship solution shown in Figure 5 after the section. As an Example the next data exemplifies the mapping.

Virtual Machine	CPU Utility	Virtual Machine	CPU Utility
V1	28.8	V9	18
V2	23.4	V10	9.2
V3	17.9	V11	8.8
V4	16.8	V12	7.3
V5	12.6	V13	8.1
V6	22.3	V14	28.8
V7	13.9	V15	24
V8	40.2	V16	26.9

Table 1 Data for mapping with VM load.

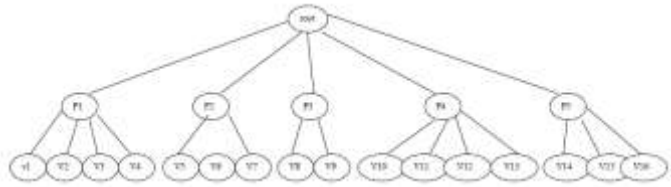

Fig. 4 Mapping relationship before using algorithm

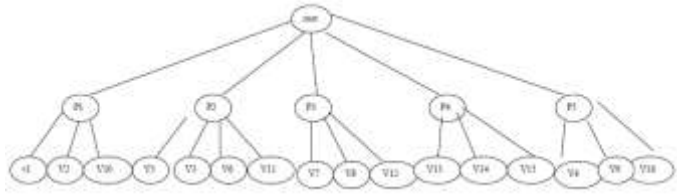

Fig 5 Mapping relationship after using algorithm

In the course of the experiment, we can see in Figure 4, the tree structure mapping relationships before using the algorithm. And in Figure 5, we can see the tree structure after the algorithm is applied on the Figure 4 tree structure. We can also see that VM loads are balanced on each physical machines i.e. every physical machine have approximately equal amount of load. The heat variation constraint, which is the stopping condition for the algorithm is also achieved i.e. overall system load variation, should be less or equal to 0.5. As a result, we can conclude that

the above explained algorithm has improved the load balancing. In the course of the experiment, we can see in Figure 4 and 5, the before and after effect on the tree structure mapping relationships using the algorithm. Figure 5, which shows the effect after using the algorithm, the VM loads are balanced on every physical node and also the system load variation parameter is also achieved. As a result, we can conclude that the above explained algorithm has quite a better globally astringency and it can come closer to be the best solution in very little time.

IV IMPLEMENTATION AND RESULTS

Two Algorithms are implemented in this paper. They are

1. Backfilling Algorithms.
2. Genetic Algorithms.

After performing the certification of the astringency behavior of the genetic algorithm, we carried out an experiment to test the performance of the overall strategy. We used most popular open source Virtual machine management infrastructure 'OpenNebula'. For the OpenNebula front end, we chose a physical machine with the configured with operating system UBUNTU 11.10, IntelR Core™ 2 Duo 2.4GHz CPU, and 4.0GB of RAM. On this machine, OpenNebula front-end is installed to schedule and manage virtual machines. Along with the host machine, we chose 6 physical machines as client nodes on which we installed OpenNebula client

platform with KVM VM. The client machines are configured with operating system Ubuntu 11.10, IntelR Core™ 2 Duo 2.4GHz CPU, 4.0GB of RAM, and 250GB disk capacity. The whole private cloud was created using the Local Area Network (LAN). The VM images were created using the Ubuntu –OpenNebula documentation. The tree structure is created such that, the host physical machine will work as root/scheduler, the other client machines will work as the second level of nodes and the VM deployed on the client machines will work as the leaf nodes.

OPENNEBULA

In private or hybrid cloud, the important infrastructure component is VI management (Virtual Infrastructure). It is the mechanism, which allocates virtual machines dynamically on the set of resource that meets the given requirement. Open Nebula manages the virtual machine infrastructure which is used for variety of responsibilities

1. Deployment of virtual machines, which can be deployed as a group or individual or on a public cloud or on a local resource.
2. It helps to automate the setup process of virtual machines (setup network, create disk images as per requirement, etc.) without touching the lower level of virtualization layer (external cloud like EC2 or internal private cloud like KVM, Xen, VMWare).

Nefeli combines consumer and administrative hints to handle peak performance, address performance bottlenecks, and effectively implement high-level cloud policies such as load balancing and energy savings. An event-based mechanism allows Nefeli to reschedule VMs to adjust to changes in the workloads served. Our approach is aligned with the separation of concerns IaaS-clouds introduce as the users remain unaware of the physical cloud structure and the properties of the VM hosting nodes. Our evaluation, using simulated and real private Iaas-cloud environments, shows significant gains for Nefeli both in terms of performance and power consumption.

V CONCLUSION AND FUTURE WORK

In this paper we present the Nefeli, to achieve the load balancing and power consumption. Our evaluation, using simulated and real private Iaas-cloud environments, shows significant gains for Nefeli both in terms of performance and power consumption. This paper presents Nefeli, a flexible gateway that allows for the effective use of virtual infrastructures. Nefeli accepts user "hints" regarding the nature of the workload under execution and exploits conditions that concern the physical infrastructure to better schedule and utilize instantiated VMs. These constraints essentially designate fundamental deployment patterns which should they be followed in the actual VM deployment, they could assist in avoiding potential bottlenecks. While experimenting with a prototype on both simulated and real private *Iaas*-cloud environments, we established significant gains for Nefeli both in terms of performance and power consumption. In offering a comprehensive VM management in *IaaS*-clouds, our approach displays a number of advantages: firstly, it considers the provision of virtual infrastructures as a whole and does not only deal with the handling of individual VMs. Secondly, users remain at all-time unaware of the inner structure and/or architecture of the physical nodes. Thus, Nefeli does offer a true separation of concerns in *IaaS* clouds. Lastly, our approach gracefully adapts to the changing workload needs as VM migration do occur in order to offset unfavorable deployments currently in place.

In the future, we plan to:

1) Investigate alternative constraint satisfaction approaches to address scalability issues present in large infrastructures;
2) Offer deployment hints that will effectively handle the deployment of virtual infrastructures in the context of real large cloud installations;

REFERENCES

[1]D. Nurmi, R. Wolski, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," Proc. IEEE/ACM Ninth Int'l Symp.Cluster Computing and the Grid (CCGRID), pp. 124-131, May 2009.

[2]Konstantinos Tsakalozos, Mema Roussopoulos, and Alex Delis, " Hint- Based Execution of Workloads in clouds with Nefeli" IEEE Transactions on Parallel and Distributed Systems, Vol.24, No.7, July 2013.

[3] http://www.opennebula.org, May 2011.

[4 http://www.openstack.org/, Feb. 2011.
[5] H.N. Van, F.D. Tran, and J.-M. Menaud, "Autonomic Virtual Resource Management for Service Hosting Platforms," Proc. ICSE Workshop Software Eng. Challenges of Cloud Computing, pp. 1-8,2009.
[6] P. deGrandis and G. Valetto, "Elicitation and Utilization of Application-level Utility Functions," Proc. Sixth Int'l Conf. Autonomic Computing, 2009.
[7] G. Tesauro and J.O. Kephart, "Utility Functions in Autonomic Systems," Proc. First Int'l Conf. Autonomic Computing, pp. 70-77, 2004.
[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of
Virtualization," Proc. 19th ACM Symp. Operating Systems Principles, Oct. 2003.

[9] R. Harper, L. Tomek, O. Biran, and E. Hadad, "A Virtual Resource Placement Service," Proc. First Int'l Workshop Dependability of Clouds, Data Centers and Virtual Computing Environments (DCDV),
June 2011.
[10] Amazon Web Services home page. http://aws.amazon.com/.
[11] Basic genetic algorithm http://www.obitko.com/tutorials/genetic-algorithms/ga-basic-description.php.
[12] OpenNebula Web site for java cloud API
http://opennebula.org/documentation:rel3.0:java
[13AS] Spanning tree
http://en.wikipedia.org/wiki/Spanning_tree

A Brief Author Biography

K. Suganthi – I obtained AMIE from Institute of Engineers(India), Kolkata. Now pursuing M.E (COMPUTER SCIENCE AND ENGINEERING) in PGP College of Engineering and Technology under Anna University, Chennai. My research interests include Cloud Networks.

K. Sathish Kumar – Completed M.E (COMPUTER SCIENCE AND ENGINEERING) from Anna University, Chennai. Working as Assistant Professor in PGP College of Engineering and Technology. My research interests include Networks.