Chapter 5

Design and Implementation of Proposed Algorithms

The previous chapters have presented and discussed in detail, energy efficient algorithms for cloud computing. To demonstrate the working and usefulness of key algorithms, a cloud framework, named ACA-Cloud, has been designed and implemented. This chapter presents the architecture and implementation of the framework that includes the algorithms proposed in previous chapters.

This chapter starts with the discussion of the system architecture and the working model of each component. To design the structural components and their behaviour, various Unified Modeling Language (UML) diagrams have been prepared and discussed. It helps to analyse the requirements and visualize the design, and further validate the architectural design of the framework. Finally, the chapter outlines the details of implementation and the used technologies.
5.1 System Architecture

There are four high-level components in this hierarchical system. Figure 5.1 shows the different modules of the proposed system and their interactions. The next section explains the functionality of each component.

5.1.1 Interface Services

The Interface Service facilitates the users to request, reserve, negotiate, and monitor the resources. A user can access the portal as a cloud user or a cloud admin. As a cloud user, the user can register, request, provision and negotiate the resources. A cloud admin can authenticate new users, manage user accounts, monitor and get access to cloud resources. There is a Green SLA negotiation with the resource requesting user. After negotiation, a request can be denied or resources can be reserved within a time slot. Green SLA and EBRPP have already been discussed in chapter 4 and further, a negotiation process among the different entities is presented in sequence diagrams in Figure 5.10 and Figure 5.11.

5.1.2 Account Manager

The Account Manager provides authentication modules to handle different account types and authentication. Modules in this package handle the account validating credentials and store the account information in the user database. It allows to performing a number of user operations like creating, activating, deleting, and updating user accounts.

5.1.3 Cloud Cluster Controller

The Cloud Cluster Controller (CCC) is responsible for monitoring the status of all the physical machines/hosts and making appropriate resource management decisions in response to the present workload and incoming users requests. The functionality of various system components are explained below.

(a) VM Handler

The VM Handler (VMH) is invoked when a request for a new VM is received. VMH finds the destination host w.r.t energy and performance efficient ranking of hosts, and the accommodating capacity of the new VM request.
Figure 5.1: Architecture of proposed ACA-Cloud framework

The resource utilization of a new VM in general is assumed to be equivalent to the average resource utilization for small and medium VMs. High utilization is assumed for large VMs as these are generally used by research scholars for their scientific experiments. VMH gets the load status of present hosts, finishing time of currently running VMs, and estimates the forthcoming VM requests from Host Supervisor Global. Then according to the VM allocation policy, a new VM request is placed to Host, based upon energy and performance of host ranking, its accommodating capability, and the finishing time of other running VMs on destination host. VMH sends information of a new VM and destination host to Global Executor.
(b) Host Supervisor Global

The Host Supervisor Global (HSG) is an overseer module which periodically collects data from various hosts and the Load predictor. HSG analyses the current state of the physical hosts, their power consumption, and number of running VMs on each host, and gets forthcoming resource demands from the Load predictor. Besides this, it retrieves and updates each host information into the cloud cluster database like CPU mode, number of CPU(s), CPU frequency, CPU socket(s), Core(s) per socket, Thread(s) per core, NUMA cell(s), Memory size, available/allocated CPUs, IP, MAC address, number of running VMs etc. It further communicates information to the Load balancer and VM Handler.

(c) Load Balancer

The Load balancer (LB) is an intelligent module which assesses the present load plus a margin of the growth and takes decisions of load distribution so that there is optimal energy consumption. It estimates the number of resources needed to reserve and maintain a set of active/inactive hosts to cater to present requests and a margin of the growth, as discussed in the prediction model in chapter 3.

This module decides the source hosts and destination hosts for VM migration and performs VM migrations. It gathers various parameters from HSG, makes the decision and issues necessary commands to Global Executor to shift the load from source host to destination host and lower the power state of source machines. Class diagram and activity diagram of this module are explained in sections 5.2.2 and 5.2.3, respectively. Besides estimating the number resources for any time, there is a need to check whether a host is operating in the safe workload range. Every host has an upper and a lower threshold load that it can carry. If this limit is crossed then it affects VM’s performance and hampers the SLA. So each host adopts MM 40-80% policy [46] to strike balance between energy consumption and SLA violation. If the upper or lower threshold load limit is violated then LB makes the decision to shift and balance the load.

(d) Global Executor

The Global Executor (GE) is connected directly with all the hosts in the cloud. This module is responsible for performing all the actions communicated by the VM Handler and the Load Balancer. Figure 5.2 general protocol used by VMH and LB while communicates with GE.
Figure 5.2: Communication protocol VMH-GE and LB-GE

Flag: The flag indicates the successful/failure execution of a GE.

Command: The command indicates the action (start, suspend, resume, save, restore, shutdown, reboot, destroy, migration VMs or pm-suspend, pm-power off, pm-hibernate, Wake On LAN etc.) needed to be performed.

Data: Data means source host, destination host, VM id, virtual networks etc.

### 5.1.4 Host Controller

Host Controller (HC) provides actual resources to a user's VM and communicates with the cloud cluster controller. The main components of HC are servers, workstations, storage, networks, etc. The functionality of each component in a host controller is as follows.

(a) **Host Supervisor Local**

The Host Supervisor Local (HSL) resides on each host to gather parameters at host level. It collects utilization, power consumption, information of a host, number of running VMs and its other information like VM Names, operating system type, state, CPUs, CPU time, maximum memory, used memory, security model, security DOI, security label, number of cores occupied by each VM, VM state, priority, VM's execution time etc. HSL provides information periodically to HS Global. It also gives necessary instructions to Local Executor to manage VMs running on a host and performs actions on local host such as suspend, power-off etc.

(b) **Local Executor**

The Local Executor (LE) is responsible for carrying out actions as communicated by HSL such as connect Hypervisor, start VM, suspend VM, resume VM, save VM state, shutdown VM, reboot VM, destroy VM, migration VMs, get virtual network list, get dom id, host suspend, host power off etc.
(c) Utilization Monitor

The Utilization Monitor is a periodically invoked module by HSL. It provides information of the percentage of host utilization.

(d) Energy Monitor

The Energy Monitor provides the power consumption information of a host. The energy consumption of servers depends on its power consumption function over a period of time. Therefore, the total energy consumption $E$ of a data center is defined in equation 4, as discussed in chapter 3 as part of the energy consumption model.

5.2 Design Details of ACA-Cloud

The first step of designing is to assess the requirements, identify the boundaries and interactions among the system components and users. To design the structural components and their behaviour, Unified Modeling Language (UML) [130] has been used. UML is a general-purpose development modeling language that provides a standard way to develop an abstract model of a system. It helps to visualize and explore the system components from different prospective with varying degrees of abstraction. In this section, the various use case diagrams, class diagrams, activity diagrams, sequence diagrams and state diagram are shown to provide the finer details of the system.

5.2.1 Use Case Diagrams

Use Case diagrams help to analyse the functional requirements of the system. In Use Case diagrams, there are actors and ovals which represent entity and use-case, respectively. Figure 5.3 shows the Use Case diagram for user authentication and management. In the first use case, user's registration, login credentials and administrator management have been demonstrated. Once the user’s credentials are verified, the user can submit resource requests and monitor the status of provisioned and pending resources requests as shown in Figure 5.4. The administrator can monitor the status of the cloud cluster and forthcoming VM requests.
Figure 5.3: Use Case Diagram for user authentication and management

Figure 5.4: Use Case diagram for resource provision
5.2.2 Class Diagram

A Class diagram helps to analyse the structural requirements of the system. Figure 5.5 shows the structure of the system by various system classes and the relationship along with their cardinality. Figures 5.6 and 5.7 present two classes with their attributes and methods. The main set of classes details are covered in Appendix A.

Figure 5.5: Class Diagram for ACA-Cloud

```
GlobalExecutor

#cmdExecStatus: int
#cmd: String

+executeCmd(cmd: String): int
+vmRunningOnHost(hostname: String): int
+getFileContents(fileName: String): String
+startVMOncRemoteHost(cmd: String): boolean
+suspendVMOncRemoteHost(cmd: String): boolean
+resumeVMOncRemoteHost(cmd: String): boolean
+saveVMSateOnRemoteHost(cmd: String): boolean
+restartVMOncRemoteHost(cmd: String, restoreVMFileName: String): boolean
+shutdownVMOncRemoteHost(cmd: String): boolean
+restartVMOncRemoteHost(cmd: String): boolean
+destroyVMOncRemoteHost(cmd: String): boolean
+migrationVM(cmd: String): boolean
+getVMIDOnRemoteHost(cmd: String): int
+getVirtualNetworkList(cmd: String): int
+hostSuspend(cmd: String): boolean
+hostPoweroff(cmd: String): boolean
+hostHibernate(cmd: String): boolean
+hostWoL(cmd: String): boolean
```

Figure 5.6: GlobalExecutor class of ACA-Cloud
## Host Class of ACA-Cloud

A Host object contains the following attributes:

- `#hostId: int`
- `#name: String`
- `#ip: String = new String()`
- `#mac: String = new String()`
- `#List<VM> vms = new ArrayList()`
- `#util: double = 0.0`
- `#pw: double = 0.0`
- `#runningVMs: int = 0`
- `#CPU_Model: String`
- `#CPUs: int`
- `#CPU_Freq: String`
- `#CPU_Sockets: int`
- `#cores_per_socket: int`
- `#Threads_per_core: int`
- `#NUMA_cells: int`
- `#Memory: int`
- `#hostStatus: String`
- `#availableCPUs: int`

### Constructor

```java
<<create>>~Host()
```

### Methods

- `+getCPUs(): int`
- `+getHostId(): int`
- `+getName(): String`
- `+getIP(): String`
- `+getMAC(): String`
- `+getUtil(): double`
- `+getPw(): Double`
- `+getRunningVM(): int`
- `+getCPUModel(): String`
- `+getCPUFreq(): String`
- `+getCPUModel(): String`
- `+getCPUSocket(): int`
- `+getCoresPerSocket(): int`
- `+getMemory(): int`
- `+getHostStatus(): String`
- `+getAvailableCPUs(): int`
- `+setVM(vms: ArrayList<VM>)`
- `+setCPUs(CPUs: int)`
- `+setHostId(id: int)`
- `+setName(name: String)`
- `+setIP(ip: String)`
- `+setMAC(mac: String)`
- `+setUtil(u: double)`
- `+setPw(pw: double)`
- `+setRunningVM(rvm: int)`
- `+setCPUModel(cpumodel: String)`
- `+setCPUFreq(CPUFreq: String)`
- `+setCPUModel(cpumodel: String)`
- `+setCPUSocket(cpus: int)`
- `+setCoresPerSocket(cps: int)`
- `+setMemory(m: int)`
- `+setHostStatus(hs: String)`
- `+setAvailableCPUs(acpu: int)`
- `+toString(): String`
- `+compareTo(compareHost: Host): int`

### Static Classes

- `static class SortAscendingHost implements Comparator<Host>()`
- `static class SortAscendingHostByVMs implements Comparator<Host>()`

---

**Figure 5.7: Host Class of ACA-Cloud**
5.2.3 Activity Diagram

The Activity diagram is one of the useful graphical representations in UML to explain the dynamic nature of the system. It shows the sequence of activity along with conditions of flow, iteration, and concurrency.

5.2.3.1 ECRS Algorithm’s VMHandler for Initial VM Placement

Figure 5.8 shows the Activity diagram of VM Handler. It shows the steps to be followed to map a new VM request to a host.

5.2.3.2 ECRS Algorithm’s Load Balancer for Optimizing Current Workload

Figure 5.9 shows the Activity diagram of the Load Balancer. The LoadBalancer assesses the status of a cloud cluster and migrates the load among the active hosts when a host is under loaded or overloaded. It also handles the overhead of VM migration.

Figure 5.8: Activity diagram of VMHandler module of ECRS Algorithm
5.2.4 Sequence Diagram

Sequence diagrams are interaction diagrams which depict the sequence of messages exchanged between a numbers of entities. It is one of the useful design tools as it provides the dynamic behaviour of the system.

5.2.4.1 GSLACRR Successful User Negotiation

Figure 5.10 shows the sequence of events performed during a successful negotiation process between an authenticated user and other involved entities. These events are performed when resources are available as per the user's requirements for the desired time slot.
5.2.4.2 GSLACRR Renegotiation for User’s Requirement

The sequence diagram (Figure 5.11) depicts the view of resources usage negotiations with cost benefits on non-availability of resources for the user's desired time slot, as explained in chapter 4.

5.2.5 State Diagram

State diagrams define the diverse states of an object during its lifetime. Figure 5.12 illustrates the different states that a user can experience during its lifetime. When a user submits a new VM request, it enters to the *Initiate* state. The request enters to the *Accepted* state on successful negotiation of usage resource and its execution time. If the resources are not available immediately, then the request enters the *Reservation* state. When the resource needs to initiate for the VM request or resources are available for the immediate VM request, the request enters to the *Provisioning* state. From the *Reservation* and the *Provisioning* states, it enters to the *Running* state when the user starts to use the VM services. The request enters the *Rejected* state when the user's negotiation is unsuccessful. Finished state is reached on three different occasions: (i) completion of VMs finishing time (ii) rejected state (iii) system failure.

![Sequence diagram of successful negotiation process of GSLACRR](image)

Figure 5.10: Sequence diagram of successful negotiation process of GSLACRR

71
Figure 5.11: Sequence diagram of GSLACRR renegotiation for user’s requirement
5.3 Implementation Details

The ACA-Cloud has been designed using open source technologies namely Ubuntu OS [131], Kernel-based Virtual Machine (KVM) [132], Java [133], PHP [134] and MySQL [135]. To provision IaaS, "ACA-Cloud" has used Operating System-based Virtualization. Although there is a concern about the overhead of host operating system, there are a number of issues related to hardware-based virtualization. One of the prominent issues is the compatibility of the hardware device drivers with hardware-based virtualization which is readily available in the host operating system. Hardware device drivers must be supported by the hypervisor as the virtualization layer directly communicates with the host hardware in hardware-based virtualization. Secondly, a wide range of host management and administration based advanced features are not available in hardware-based virtualization, though these are commonly available in the host OS in Operating System-based Virtualization. Cloud setup uses KVM as a Virtual Machine Manager and an open source Libvirt API [136]. Libvert has a set of APIs to enumerate, monitor and manage virtual machines, virtual networks and storage. Livert has been chosen as it is compatible with various hypervisors such as KVM [137], Xen [108], and VMWare [138]. Thus, with minimum efforts the underlying hypervisor can be changed [139].
The "ACA-Cloud" is implemented in Java. Java is the chosen programming language as it is an incredibly entrenched technology. It has a number of advantages, such as; it is platform independent, fast, robust, secure and reliable. Moreover, powerful development tools such as Eclipse [140], Netbeans [141] for Java programming are freely available. Netbeans [141] has been used as a Software Development Platform. A user can request and access the resources using web based interface developed using PHP. The choice of PHP as a web based programming language has been made because it has excellent documentation and community support. Table 5.1 gives the details of implementation technologies.

Table 5.1: Implementation technology used by the ACA-Cloud

<table>
<thead>
<tr>
<th>Datacentre</th>
<th>Heterogeneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>KVM 0.14.1</td>
</tr>
<tr>
<td>Virtualisation API</td>
<td>Libvirt 0.10.2.1</td>
</tr>
<tr>
<td>Host Operating system</td>
<td>Ubuntu 12.04</td>
</tr>
<tr>
<td>Network File System (NFS) Server</td>
<td>NFSv4</td>
</tr>
<tr>
<td>Technology</td>
<td>Java 1.7, PHP 5.5.12</td>
</tr>
<tr>
<td>Software Development Platform:</td>
<td>Netbeans 7.1</td>
</tr>
<tr>
<td>Webserver:</td>
<td>Apache Tomcat 7.0.22.0</td>
</tr>
<tr>
<td>Database:</td>
<td>MySQL 5.6.8</td>
</tr>
</tbody>
</table>

5.3.1 Experimental Setup

For experiment, a heterogeneous virtual environment has been set up that consists of eight servers with data as given in Table 5.2. All servers are connected with 100 Mbps Ethernet and 8 TB Networked Attached Storage (NAS) storage. Hosts run VM and a common storage provides seamless storage to allow live migration [142] of VMs. NFS server and client [143] has been configured to create NAS. The Dell Core 2 Duo machine was chosen as the cloud cluster controller and the rest of machines were selected as compute hosts.
Table 5.2: Experimental setup [144-147]

<table>
<thead>
<tr>
<th>Server type</th>
<th>Dell power edge r710 server</th>
<th>Dell Power Edge 2900 server</th>
<th>Dell i5 2.5 GHZ E7500</th>
<th>Dell Core 2 Duo E7500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Model</td>
<td>Xeon E5520</td>
<td>Xeon®5400</td>
<td>i5 2.5 GHz</td>
<td>Core 2 Duo E7500</td>
</tr>
<tr>
<td>Cores</td>
<td>2*4</td>
<td>2*4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
<td>8 GB</td>
<td>4 GB</td>
<td>2</td>
</tr>
<tr>
<td>Disk</td>
<td>1350 GB</td>
<td>438GB</td>
<td>320GB</td>
<td>120GB</td>
</tr>
<tr>
<td>P_{idle} (W)</td>
<td>128.7</td>
<td>40</td>
<td>39.8</td>
<td>39.3</td>
</tr>
<tr>
<td>P_{max} (W)</td>
<td>247</td>
<td>80</td>
<td>106.8</td>
<td>78.9</td>
</tr>
</tbody>
</table>

Figure 5.13: Authenticated user resource request screen
5.3.2 User Interface

The user can login into the system and submit the VM request via a web based user interface. Figures 5.13, 5.14 and 5.15 show screen shots of an authenticated user's VM request, GSLA negotiations, and the admin panel respectively. The web interface allows the admin to see the machine status, configuration, perform actions like start, power-off, and watch the VM status on each machine etc. Screen shots of the development environment for the ACA-Cloud project have been provided in Figures 6.17 and 6.18.

Figure 5.14: GSLA negotiation screen
Figure 5.15: ACA-Cloud administration panel screen

Figure 5.16: Development environment and output generated by GlobalExecutor for ACA-Cloud
5.4 Summary

This chapter describes the design and implementation of the ACA-Cloud framework. The detailed requirements and design have been analyzed using UML. The implementation details of the experimental testbed that implements the proposed algorithms are provided. The main objective of the implementation of the ACA-Cloud framework is to demonstrate the working of the proposed algorithms. The next chapter discusses the experimental results and validation of the proposed algorithms.