CHAPTER 3

QOS AWARE DISTRIBUTED SYSTEM

Quality of Service (QoS) requirement in distributed system is one of the major concerns as the applications of distributed systems are increasing. The satisfaction of QoS needs has become the primary need. It will not be acceptable if the QoS is satisfied only in one end. The complete process starting from the source, the intermediate nodes and the destination also must promise the QoS. For acquiring the expected QoS the disk scheduling and thread scheduling must be done carefully to make the process efficient. The QoS condition must be user-oriented and not system oriented. QoS requirements has to be specified by the customer first itself i.e. it must be declarative. In message passing, remote invocation and system services the QoS requirements are crucial.

Load balancing is a method of computer networking for distributed workloads across multiple computers or a computer cluster, network links, CPUs, disk drives and so on. This method is used for maximizing throughput and minimizing the run time. This also helps in avoiding the overload and optimizing the resource use. Commonly used application of load balancing is to provide a single internet service from multiple servers. Another method of load balancing is called as round robin Domain Name System (DNS). It does not require specific software. Here many IP addresses are associated with a single domain name. The clients are requested to choose the required server to connect with.
A variety of scheduling algorithms are used by load balancers to find which backed server to send a request to. These simple algorithms include random choke or round robin. Hardware and software load balancers have a variety of special features such as asymmetric load and priority activation. Load balancing is also used to implement failover. It is the continuation of a service after the failure of one or more of its components.

Leader election is a method of assigning a single process as the organizer of some task distributed among several computers (nodes). In order to decide which of them will get into the “leader” state, the network nodes must communicate among themselves. They require an order to break the symmetry among them. Leader election algorithms are generally designed to be economical in terms of total numbers of bytes transmitted and time taken by the same.

SLA is basically a contract between a person and the costumer. For example, a DBA’s costumer is typically the corresponding company the DBA works for. During a natural calamity or a disaster or corruption, the maximum amount of data loss in the application transaction faces is considered to be SLA. The SLAs are generally a combination of many such things. SLA is like a contract that the DBA has obligated to meet. The DBA with zero downtime and zero data loss SLAs should make sure that these are positively met during disasters. If the DBA initially agrees to the SLAs and later fails to meet them when disaster occurs due to the peer ability of the system, there is a possibility of losing job. The knowledge about SLA is quite important before the DBA makes sure that the system is able to fulfill them. This helps to correctly architect the system to meet the SLAs. If a disaster occurs and if the system could not meet the SLAs obviously DBA will be held responsible.
3.1 INTRODUCTION

Nowadays the demands for network services are growing and there is a necessity to satisfy the customer requirements on time. The existing traditional techniques have ability to satisfy the customer requirements as an immediate source for a quick solution. Even though the traditional model promises scalability and reliability, in today’s dynamic and fast moving environment, the traditional technique is not enough and satisfying. It works by sharing its resources to multiple users and is linked with multiple servers, but at a time, only one particular client makes use of the resources from the server.

Incorporating QoS in distributed system is becoming very popular nowadays. Various QoS parameters in Cardellini et al (2002) are given to maintain QoS conditions in the end systems and the network-QoS specification includes the requirements for performance, policy, level of service, cost of service, reliability, security and QoS management. QoS mapping is used to represent QoS at different system levels. QoS parameters are used to derive resource requirements such as computation times, communication and memory. A mapping of QoS parameters to the system layers is done to give quantitative parameters to achieve performance, transfer rate, cost, response time and synchronization. In Ossama & Douglas (2001) and Kaiqi & Harry (2008) the authors have defined a set of QoS metrics in SLA, which includes percentile response time, cluster utilization, packet loss rate, cluster availability. The percentile response time is the time to execute a service request, cluster utilization is the percentage of the time that a computational cluster gets used, packet loss rate is the probability of a packet lost during transmission due to filling of buffer, finally cluster availability defines the number of clusters that the service providers afford and the sum of failures that is tolerated by the system. There are several concepts given in
Ossama & Douglas (2001) and Cardellini et al (2002) on QoWS-Service and Service Level Agreement (SLA), Service defines set of characters significant for web content provision. The two classes of users: high and low, and two main classes for requests: static and dynamic are used to support multiple service classes. The best way to guarantee all the QoS principles for QoWS in a web cluster is an approach called “dynamic server-partition algorithm” by Cook et al (2002) for performance isolation and high utilization mechanisms. There are set of qualitative measures of how appealing a web service is for applications that may potentially consume it. These metrics include availability, accessibility, performance compliance, security, energy efficiency and reliability.

The architecture offered is authenticated by a setup composing a red hat linux server and a Pseudo Domain Name System (PDNS) server and a middleware JBoss server. Transaction, communication, persistence and security services are provided by JBoss server which is managed by Java management extension microkernel. The IP address is converted to a URL and vice versa by the help of PDNS server. Replication is used for increasing the availability of the data in all clusters.

The Load Balancing Factor (LBF) is inversely proportional to the number of requests. When the value of LBF is less, many requests are forwarded to server. The load balancing metrics are observed by SLA and clustering updates. Whenever a variation is perceived, the concerned application server takes care by taking necessary steps.

There are many application servers in the system. The middleware plays the role of an application server. The election process is used for the coordinator selection from among the application servers. The election process is carried out for coordinator election whenever a cluster group is started. The dynamic configuration, load balancing and run time monitoring
of QoS-aware cluster driven by SLAs are accounted only by the proposed middleware model. Whenever a client issues a request it is appended in the request queue which is then allotted to server by its coordinator.

3.2 DEVELOPMENT MODEL

The system is made up of many application servers, where the middleware is an application server. The application server is chosen as the coordinator by election process. When a request is submitted by a client, it gets added in the request queue of coordinator. It is then allocated to a server, by the coordinator. In this process the election is based on least IP address of the cluster members PDNS which is made up of two software components: a PDNS server and a PDNS client. Network resources over the internet are identified by numeric IP addresses, but these are difficult for users to remember. The PDNS database contains records that map user-friendly alphanumeric names for network resources to the IP address used by those resources for communication. The major advantages given by Cricket & Paul (1998) for using PDNS are that: it eliminates management of host tables, it is consistent on all hosts, PDNS is used on the Internet and the Internet would not exist without it. The entire setup in supported by the PDNS server. The PDNS is configured to resolve the type of URL client into a coordinator IP address. This plays a critical role as the PDNS ensures that a client always goes through the coordinator node which balances the load.

3.2.1 Architecture Model

The architecture of a QoS Aware Shared Application Cluster (QASAC) is shown in Figure 3.1. Resource manager of this architecture cooperates with all the member servers to ensure QoS services. Figure 3.1 shows the basic architectural model of proposed QoS aware shared...
application cluster (QASAC) where the Application Server (AS) indicates the physical node in the cluster.

![QoS Aware Shared Application Cluster (QASAC)](image)

**Figure 3.1 QoS Aware Shared Application Cluster (QASAC)**

The coordinator node is one which receives the request from the client and the rest of the nodes are acting as members of the cluster. When a request comes to the leader node, it checks with the dynamic load balance algorithm where to forward the request and it redirects the client to the selected node. The load balance algorithm is acting on the information collected from all the nodes by the coordinator at a monitoring interval. The architecture is augmented with the PDNS update to mask node failures from the client. PDNS updates are sent when a coordinator fails and a new coordinator gets elected, so that the client will not be routed to failed coordinated node. Application server communicates with the server using HTTP protocol to access the application hosted. The server uses RMI to communicate between servers to share information. Server in the end, communicates with PDNS with the help of SSH protocol to send the PDNS update.
3.2.2 Election Model

The architecture requires one server to act as a coordinator to carry out dynamic load balancing. Coordinator is selected based on election algorithm which specifies the minimum IP address to be elected. The entire client requests are initially responded by the coordinator to route it to a particular application server based on balancing algorithm as shown in Figure 3.2. Every server in this cluster is aware of all other members who are currently operating.

![Figure 3.2 Process three holds an election as coordinator](image)

Whenever a node fails in the cluster, the election algorithm is performed and a coordinator is elected as shown in Figure 3.3.

![Figure 3.3 Process two holds as election as coordinator](image)

When a coordinator fails, all other node will perform election algorithm as shown below. As the coordinator gets elected, it notifies the PDNS to update its records to point the newly elected. When a client request
comes to PDNS for resolution, it gets the latest information, thus the architecture masks node failure from client.

```
membershipChanged(deadMembers, newMembers, allMembers)
{
    setAllMembers(allMembers);
    // coordinator election
    resourceManager = min(allMembers);
    if(ipAddResMgr.equals(myLocalHostAddress))
    {
        updatePDNS(resourceManagerIp)
    }
}
```

The dead members, new members and all members are given as the input to the function membership. The minimum of all the members is chosen as the resource manages. PDNS of the resource manager IP is updated if the IP address of the resource manager is equal to the load host address.

### 3.2.3 Monitoring Information

At uniform interval the resource manager collects the monitoring information from all the nodes. This will be used for load balancing and application hosting. Table 3.1 represents the different fields used by the resource manager and its description. The ResourceManagerMonitor() function is used to collect information about each node.
Table 3.1 Monitoring data structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description of Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>Internet Protocol (address of the member)</td>
</tr>
<tr>
<td>FreeMemInfo</td>
<td>Free Memory Information</td>
</tr>
<tr>
<td>Number of Request</td>
<td>No of Request served by the server</td>
</tr>
<tr>
<td>Response Time</td>
<td>Average response time in a monitoring interval</td>
</tr>
<tr>
<td>Number of Thread</td>
<td>Number of threads created by the server</td>
</tr>
<tr>
<td>Queue Length</td>
<td>Number of request pending</td>
</tr>
<tr>
<td>AppList</td>
<td>List of application run by member</td>
</tr>
</tbody>
</table>

The IP field name contains the member’s IP address. The FreeMemInfo holds information about the free memory space available. The information about the number of requests served by the server is contained in the Number of Request field. Response Time is the field that contains the average response time in a monitoring interval. The number of threads created by the server is accounted in the Number of Thread field. The Queue field contains information about the number of pending requests. AppList field contains the list of application run by the member.

```java
resourceManagerMonitor() {
    HashMap nodesInfo = new HashMap();
    sleep(monitoringInterval);
    // To collect load information from All the members of the cluster
    for(i=0;i<allMembers.size;i++) {
        HashMap nodeInfo = getNodeInfo(allMembers[i]);
    }
```
nodesInfo.put(allMembers[i],nodeInfo);
}
evaluateSLA();
}

getNodeInfo(member){

    HashMap nodeInfo = new HashMap();
    String numberOfReqt = getNumberOfRequest();
    int queuedReqst = getQueue();
    nodeInfo.put("AppList", getAppList());
    String freeMem = Long.toString(getFreeMem());
    nodeInfo.put("FreeMem", freeMem);
    nodeInfo.put("NumberOfRequest", numberOfReqt);
    String respTimeForInterval = Long.toString(getResponseTime());
    nodeInfo.put("ResponseTime", respTimeForInterval);
    String numberOfThread = Long.toString(getNumberOfThread());
    nodeInfo.put("NumberOfThreads", numberOfThread);
    String queuedReq = Integer.toString(queuedReqst);
    nodeInfo.put("Queue", queuedReq);
    return nodeInfo;
}
The information about the nodes is stored in the hash table. The data of the members of the cluster are collected and updated after the monitoring interval. The node information is communicated with all the members. The node information composes of the details about the number of request received; number of request queued up, the amount of free memory in each node and the response time for the request. The Service Level Agreement (SLA) is evaluated finally.

### 3.2.4 Load Calculation

Load balancer unit of the system balances the request load. This is done based on the network traffic and resource utilization. The load balancing unit will ensure that the response for the request doesn’t extend beyond the Service Level Agreement (SLA).

Therefore, the load balancing factor of a node is computed by taking into account the five principal factors i.e., Free Memory \( Mem_i \), Number of Request \( Nor_i \), Number of Threads \( Thr_i \), Response Time \( Res_i \), and Queue Length \( Que_i \). The calculations of these factors are as follows Giorgia et al (2007):

\[
MemFactor_i = \left( \frac{Mem_i}{\text{TotalMem}} \right) \times 100 \quad \text{Where} \quad \text{TotalMem} = \sum_{i=1}^{n} Mem_i \tag{3.1}
\]

The \( MemFactor_i \) given by Equation (3.1) represents the free memory in java virtual memory. The \( ThdFactor_i \) is calculated using Equation (3.2).

\[
ThdFactor_i = \left( \frac{100}{Thr_i} \right) / \left( \text{TotalThr} \right) \quad \text{Where} \quad \text{TotalThr} = \sum_{i=1}^{n} Thr_i \tag{3.2}
\]
The \( \text{ThdFactor}_i \) represents the number of threads created by the server process.

\[
\text{NorFactor}_i = \frac{100/(\text{Nor}_i)}{(\text{TotalNor})} \quad \text{Where} \quad \text{TotalNor} = \sum_{i=1}^{n} \text{Nor}_i \tag{3.3}
\]

The \( \text{NorFactor}_i \) given in Equation (3.3) represents the number of requests issued by the server process.

\[
\text{QueFactor}_i = \frac{100/(\text{Que}_i)}{(\text{TotalQue})} \quad \text{Where} \quad \text{TotalQue} = \sum_{i=1}^{n} \text{Que}_i \tag{3.4}
\]

The Equation (3.4) is used to calculate the \( \text{QueFactor}_i \) which represents the queue length i.e., the number of processes in the queue.

\[
\text{ResFactor}_i = \frac{100/(\text{Res}_i)}{(\text{TotalRes})} \quad \text{Where} \quad \text{TotalRes} = \sum_{i=0}^{n} \text{Res}_i \tag{3.5}
\]

The \( \text{ResFactor}_i \) represents the response time of elapse between the delivery of a client request to an application server and the transmission of the reply from that server is given by Equation (3.5).

\[
\text{LBF} = \frac{\left(\text{MemFactor}_i + \text{ThdFactor}_i + \text{NorFactor}_i + \text{QueFactor}_i + \text{ResFactor}_i\right)}{5} \tag{3.6}
\]

The Load Balancing Factor (LBF) is calculated using Equation (3.6) by taking the median of five units above, assuming that each parameter has the same weight in the construction of this unit.
3.2.5 QoS Metrics in QASAC

A set of metrics is defined for the architecture, which govern the operations of QASAC. The QoS metrics are defined as:

(a) **Response Time** \((Res_i)\) – The system provides better response time by balancing the load between the applications based on the LBF. The LBF is computed based on the response time \((Res_i)\) factor of individual application. So, the response time of the system is reduced to a large extent hence increasing the server utilization. The response time \((Res_i)\) has to be less for an efficient system.

(b) **Service availability** – The calculations of LBF implies that, the service availability is greatly increased, thus making it reliable. Percentage time of the computational services is offered to the client / customer. The service must be available at all the time for a reliable system.

(c) **Server utilization** – The middleware supports the creation and removal of users, provides the ability to verify an identity, manages the addition and removal of services associated with user IDs. The PDNS allocate all available disk space. For creating a new partition from an unallocated space, Disk Management or DiskPart.exe is used.

(d) **Reliability** – Reliability is the server’s overall ability to maintain its quality that is usually measured by its failure per unit time. It should be high in order to increase its availability.
(a) QoS metric shows that, the response time is reduced to a large extent owing to less waiting of processes in the system’s queue. (b) Shows that, as the response time is reduced, the system availability gets increased and more requests are served. (c) QoS shows that, the server utilization is increased and hence, the overall performance of the system is increased and streamlined. (d) Finally it shows that, the dependability or the reliability of the system is increased due to its availability and increased utilization rate.

3.3 SLA INFORMATION

Figure 3.4 shows the setup of a QASAC driven by SLA model, where a set of application servers are present which are each associated with their own applications which run in the server on request by the server i.e. the application App-1 hosted on two application servers and App-1 is hosted on one. App-2 is configured in two nodes. Hence, the requests made by the clients for App-2 will be served by any of the two servers. Among the set of ‘n’ application servers in the system, one server is selected as the leader. Once, a client request is received by the leader of the system, it identifies the server with least workload in the system. The client request is then redirected to the application server. On receiving the request, the server resolves the URL to the IP address by using the PDNS. On completion of the processing of the requested page, the server returns the requested page to the client after resolving the IP address to the URL again by using the PDNS.
3.3.1 Configuration

Sample SLA information is provided below. Application name and number of nodes in the application cluster and load threshold are to be configured by the user. If the calculated load value exceeds the value configured in the SLA and the NumberOfNodes parameter would be greater than one, then the application will be hosted in more than one server.

<SLA>

<ApplicationInfo>

<ApplicationName>App1</ApplicationName>

<NumberOfNodes>2</NumberOfNodes>

</ApplicationInfo>

</SLA>
SLA information is stored by each application server and the time stamp of the file is used for comparing with the latest information. The leader node intimates its members with the information about the updating on the SLA parameters by its user.

3.3.2 Load Balance and Resource Monitor

As shown in Figure 3.5 the MacroResourceManagerMBean sets the monitoring interval and starts the monitoring thread. This also performs the select host function which selects the least loaded node for the load balancer. MacroResourceMonitoring monitors the membership changed event and elects the leader. The leader node sends the PDNS update and collects the membership information from other nodes in an interval of one second. LoadBalancer receives the request and checks if the node is the leader. If it is the leader, it performs the request routing, else respond to the request. MonitoringMBean monitors the JVM memory and returns the detail when requested.
The load balancing filter receives one request from the queue and decides where to forward the request. The load balancer filter class is used to get the IP of the host and the reference of mbeam for the IP and checks the target and prepares the response and forwards the response to the leader node or the target node.

3.3.3 Pseudo Code - Macro Resource Monitoring

In any peer-to-peer network with many heterogeneous nodes, it is necessary to take care of resources. When a request for a resource is received, the allocation of resource must be monitored and optimized. It is an important task to make sure that the resource reaches the destination in right time in a proper format. The following is a pseudo code for Macro Resource Monitoring:

```java
public class MacroResourceMonitoringImpl {
    public void startMonitoring() {
        //Start the thread in order to check the cluster membership
        monitor.start();
    }

    public void setMonitoringInterval(int interval) {
        this.monitoringInterval = interval;
    }
```

Figure 3.5 Resource monitoring and load balance filter
public void run() {
    boolean running = true;
    while (running) {
        // Sleep for a certain time
        Thread.sleep(mrm.getMacroMonitoringInterval());
        // If this is the leader collect membership memory information
        if (isLeader)
            { collectMemberInfo(); }
    }
}

public void membershipChanged(Vector deadMembers, Vector newMembers, Vector allMembers) {
    // Get membership changes
    this.deadMembers = deadMembers;
    this.newMembers = newMembers;
    this.setAllMembers(allMembers);
    // Leader election
    leader = Collections.min((Collection)allMembers).toString();
    // extract the Leader IP address
String ipAdd = leader.substring(0, leader.lastIndexOf(':'));

// Get local IP address
myLocalHostAddress = InetAddress.getLocalHost().getHostAddress();

if (ipAdd.equals(myLocalHostAddress))
{
    //Send the update to DNS server
    //Collect statistics from other member as well
    collectMemberInfo();
}

The process is initialized first, then the interval time for monitoring the resource is set. Whenever, the membership of a node changes, the details of the existing members are collected and the least IP address is calculated. If the calculated IP address equals the current host IP address, the leader is also set. After, the leader is set, it sends the update about the IP address to the PDNS server. For an interval ‘T’, the JVM memory of all members are collected by the leader using RMI and then it is stored.

3.3.4 Pseudo Code - Macro Resource Manager

The resource manager plays an important role in data communication. A leader is selected among the nodes and it takes care of the resource management. The task of the resource manager is to allocate the resource that is in the nearby location and checks for the correctness of data and transfer of data. The code for resource manager is explained below:
public class MacroResourceManager {

    protected void startService() throws Exception {
        //Instantiate the macro resource monitoring

        //Start the MBean service
    }

    public String selectHost() {
        // Read the Monitoring information

        // Choose the lease loaded host and return this to Load balancer
    }

    public String getLeaderIp() {
        // Read the Monitoring information and return the leader
    }
}

The MBean service and the monitoring thread are initialized. Then, the monitoring interval is also fixed. The leader of macro resource monitoring is identified and that information is passed. As a next step, the member information is read from the monitoring MBean. The initial IP is set to a maximum value. This initial IP is checked with all members to find out the maximum free space available with other members. The IP address of the member which is having the maximum free memory available is informed.
3.3.5 **Pseudo Code - Load Balancing Filter**

The task of the load balancing filter is to filter up the unwanted occupied space of each node and clean it thereby making space for any upcoming process. The role of a load balancing filter is crucial in any network for eliminating the unnecessary information from the system. The following piece of code implements the load balancing filter in a peer-to-peer network:

```java
public class LoadBalancerFilter implements Filter {
    public void init(FilterConfig fConfig) throws ServletException {
        print("Filter initialized");
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        // Get the local host IP
        myLocalHostAddress = InetAddress.getLocalHost().getHostAddress();

        // Get the reference of mbean and get the leader IP
        // Check if this node is the leader
        if(myLocalHostAddress.equals(leader)) {
            // Get the target host from Macro Resource manager
            addrStrHost = selectHost();

            // Prepare the target response
```
// Check if the target is not mine
if(!myLocalHostAddress.equals(addrStrHost))
{
    resp.sendRedirect(url);
}
else {
    // Process the request
}

public void destroy () { }

The process is initiated only when receiving a request. The IP address of the system is received. The request and response of CastServletRequest and HttpServletRequest are categorized. The MBean reference and the IP address of the leader are obtained from the server. The system is checked whether it is the leader. If it is the leader, the reference on the MBean is acquired. The least loaded IP is got from the MBean and if it is the remote IP address, then the message is sent to the client along with the URL. The request is passed to the application and the unused resources are destroyed.

3.3.6 Testing Procedure

The test environment is set up first for the testing process. The MBean is deployed for resource monitoring. The sample web application
‘.war’ has to be deployed in to the deploy directory. The PDNS service is started. The number of application servers are identified and based on the requirements, it is started. The testing procedure is done by verifying the web application access in the browser. The results mentioned in the test cases below are verified. Following section describes the test cases for the System testing performed.

### 3.3.6.1 Test case – system testing

The test cases performed under the given scenario and the execution of clustering of nodes, PDNS update and load balancing are given in Table 3.2.

<table>
<thead>
<tr>
<th></th>
<th>Test Case</th>
<th>Excepted Results</th>
<th>Actual Results in Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Clustering</td>
<td>Scenario: System when gets started need to form cluster. Execution: Start the application server in a node by typing following command: <code>$ sh run.sh –c test –b 192.168.42.131</code></td>
<td>Message in the application server console, stating the system has joined the cluster with node information.</td>
<td>Refer: Figure 3.6</td>
</tr>
<tr>
<td>1.2</td>
<td>Scenario: When another system comes up in to network and has the same cluster name need to join. Execution: Start the application server in another node by typing following command: <code>$ sh run.sh –c test –b 192.168.42.128</code></td>
<td>Message in the application server console (for both) displaying the all the member IP addresses.</td>
<td>Refer: Figure 3.7</td>
</tr>
</tbody>
</table>
Table 3.2 (Continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Test Case</th>
<th>Excepted Results</th>
<th>Actual Results in Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td><strong>High Availability – Leader Failure – PDNS update</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Scenario: System as a whole able to perform the leader selection and send update to PDNS the address of the leader. Execution: Start the application server in a node by typing following command: <code>$ sh run.sh –c test –b 192.168.42.131</code></td>
<td>Console message and from client when queried the PDNS server displays the leader IP.</td>
<td>Refer: Figure 3.8</td>
</tr>
<tr>
<td>2.2</td>
<td>Scenario: When the leader goes down out of network due to some reason the system able to select the new leader as designed and update the PDNS accordingly. Execution: <strong>Stage 1:</strong> Start the application server in a node by typing following command: <code>$ sh run.sh –c test –b 192.168.42.131</code> <strong>Stage 2:</strong> Start the application server in another node by typing following command: <code>$ sh run.sh –c test –b 192.168.42.128</code> <strong>Stage 3:</strong> Pull out the node 192.168.42.128 out of network.</td>
<td>Console message and from client when queried the PDNS server displays the leader IP.</td>
<td>Refer: Figure 3.8, 3.9, 3.10, 3.11 and 3.12</td>
</tr>
</tbody>
</table>
### Table 3.2 (Continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Test Case</th>
<th>Expected Results</th>
<th>Actual Results in Figure</th>
</tr>
</thead>
</table>

#### 3. Load Balancing

**3.1 Load Balancing**

**Scenario:** The leader able to collect the performance factor from the member system.

**Execution:** Start the application server in another node by typing following command:

```bash
$ sh run.sh -c test -b 192.168.42.128
```

Console message displaying the members and the parameter.

Refer: Figure 3.13

**3.2 Load Balancer able to select the least loaded node and forward the request to the node.**

Console message and client URL.

Refer: Figure 3.14 and 3.15

---

**Figure 3.6 Screenshot of test case 1.1 in Table 3.2**
When a node comes up in the network, it forms the cluster (QASAC Partition) and detects that it is the only node available. As in Figure 3.6, the marked position shows that all members contains an IP address 192.168.42.131 and is belonging to partition (cluster) QASAC Partition.

Figure 3.7 Screenshot of test case 1.2 in Table 3.2

When another node comes into the cluster with the same partition name it joins the cluster and in console it displays all members. Marked position in the above Figure 3.7 shows that all members contains two IP
addresses 192.168.42.128 & 192.168.42.131 and is belonging to partition (cluster) QASAC Partition.

At initial stage, only single node formed the cluster and sends the update to PDNS with its IP. As time progresses, another node joins the cluster and system performs the selection logic. According to the selection process, only a single node sends the update to the PDNS server. Figure 3.8 shows the screenshot of test case 2.1(stage 1) in Table 3.2.

**Figure 3.8 Screenshot of test case 2.1(stage 1) in Table 3.2**

The marked position in Figure 3.8 shows that all members contain the IP address 192.168.42.131 and belongs to partition (cluster) RMQAASpartition. As this is the only member, it sends the update to PDNS server which is in the last octet of its IP address.
In Figure 3.9, the marked position shows that all members contain the IP addresses 192.168.42.128 & 192.168.42.131 and belong to partition (cluster) RMQASPartition. Since, there are two members, leader election is performed and leader (192.168.42.128) sends the update to PDNS server in the last octet of its IP address.
Figure 3.10 Screenshot of test case 2.1(stage 3) in Table 3.2

Figure 3.10 shows the command line result from the client machine request to the IP of the clustered URL.

Figure 3.11 Screenshot of test case 2.2(stage 1) in Table 3.2
The node 192.168.42.128 leaves out of network and the other node 192.168.42.131 is selected as the leader. As this node is the only member and as well as the leader, it sends the PDNS update.

Figure 3.12 Screenshot of test case 2.2(stage 2) in Table 3.2

Figure 3.12 gives the command line result from the client machine request to the IP for the Clustered URL after the failure of one node.
The console message showing the memory information collected across all the members of cluster is shown in Figure 3.13. The marked position shows that, there are two members in the cluster (192.168.42.131 & 192.168.42.128) and each having free memory of 89 & 75 MB respectively.

Figure 3.13 Screenshot of test case 3.1 in Table 3.2
Figure 3.14 Screenshot of test case 3.2 in Table 3.2

Figure 3.14 illustrates the console message showing the memory information collected across all the members of the cluster and selection of a particular host to route the request. Marked position shows that, there are two members in the cluster (192.168.42.131 & 192.168.42.128) and each having the free memory of 87 & 82 MB respectively. Also, it shows that the node 131 is selected to route the request, as this is having more free memory.
As the node with IP address 192.168.42.131 is having more free memory and is selected as leader, it takes care of the process of responding to the request. In this way the load balancing is performed. Figure 3.15 shows the page of a leader responding to a client.

**3.4 RESULTS AND DISCUSSION**

In order to analyze the performance of the system, a load of 400 login requests have been applied to the system to study the load balancing pattern on servers. Results show that, as the load gets increased, the requests are distributed to the servers according to their load information. Also, it is observed that when different weight is assigned to load parameter it impacts the load balancing decision. Table 3.3 specifies the weight applied on each of
the parameters in the system. The different parameters under consideration are free memory, number of request, number of threads, queue, and average response time used to compute the system load. The result of the weight applied is given in Table 3.3. It consists of a set of time intervals ‘T’ \{T_1,T_2, \ldots, T_7\} each of 10 seconds duration and two servers A, B and the set of parameters specified in Table 3.1 which finally are used to calculate the load factor for the given system, and it depends on the time intervals and the member server.

**Table 3.3 Weights on Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Free Memory</th>
<th>Number of Threads</th>
<th>Number of Request</th>
<th>Queue</th>
<th>Average response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.4, gives the load analysis details of the two servers A and B considering the following parameters: free memory, number of threads, number of request, queue size and average response time. The LBF is calculated by the Equation (3.6) and the analysis details are plotted in Figure 3.16 to Figure 3.21. Figure 3.16 represents the relationship graph between free memory of the servers (A and B), and the time intervals submitted for processing. The drop in the graph of server B represents that, the server is being utilized for processing the requests. Thus, there is a reduction in the free memory allocated to the server. The peak in the server A represents that it has a lot of free memory for processing the time intervals and hence extra jobs are assigned to the server for handling. The free memory at the starting and ending of the execution of the process must be same, but it is not found to be the same at the end of execution because of the absence of garbage collection.
It is observed that at T1 server B has 130 threads. At T2 server B has 128 threads. However, at T3 the threads increase to 215 in server B. In comparison, server A has less increase in number of threads. Hence in Figure 3.16 shows crossover at free memory A and free memory B for the time interval T2.

Figure 3.17 represents the number of threads created in each server for processing the interval. The Server B has more threads showing that, it is being utilized largely for processing and Server A is utilized less due to the
less number of threads created. There is a step by step increase in the load allotted to the Server B showing that it is being utilized consistently, but Server A is utilized almost the same way throughout the time intervals, showing that it is being utilized less.

![Graph showing time intervals vs number of requests for servers A and B](image)

**Figure 3.18 Time Intervals vs Number of Request (Server) A&B**

Figure 3.18 represents the processes number of requests and the interval submitted for processing to the server. At time interval T1 and T2, both servers are being utilized same due to number of request (If number of request should be one that indicates to initialize the server). At time interval T3, T4 and T5 Server B has more number of request than Server A. At time interval T6 Server A has more number of requests than Server B. The graph shows that, Server A is being utilized for processing than Server B.

Figure 3.19 represents the processes waiting in the queue for processing and the interval submitted for processing to the server. Every node in the network is associated with a queue that is used to insert input process into memory and release the process from memory; the queue size is calculated by the previous internal queue size and subtracting the number of outgoing request, which is then used to calculate the current number of incoming request.
Figure 3.19 Time Intervals vs Queue (Server) A&B

Figure 3.20 shows that, Server A has less average response time than Server B, showing that the servers B provide a faster response to the incoming process than server A. The response time depends upon the number of requests and the queue size. At time interval T3, for Server A, the number of requests is 56 and the queue size is 68, so the response time is 8399, whereas in Server B, the number of requests is 67 and the queue size is 84 and the response time is observed as 13679. The same applies to all consistent time intervals.

Figure 3.20 Time Intervals vs Average response Time (Server) A & B
Figure 3.21 Load Balancing Pattern – Server A and Server B

Figure 3.21 shows the load information computed (using LBF) on each server member against the time interval. From the Figure 3.21, it is observed that, at a time T1, the load at Server B is more than Server A. At this point load balancer makes the decision to forward the requests to server A till the next interval. The same is repeated for each uniform time interval to distribute the load across multiple servers. At time interval T4, the load balancing factor in dependant on the number of requests, as in Server A, the number of request is less, so the load balancing factor is more where in Server B as the number of request is more, so the load balancing factor is less.

From the Table 3.4 Load Analysis, the average response time of the proposed system of QASAC architecture shows 4% improvement when compared to existing system. By the test performed on QaAS architecture, the response time ranges from 2% to 8%. In our QASAC architecture the response time has increased to 12%.
This architecture uses Pseudo Domain Name System for keeping track of any failure and elects the alternate resource manager. The server takes 0.071ms to provide service to the client, thereby reducing the average waiting time of the client.
Table 3.4 Load Analyses

<table>
<thead>
<tr>
<th>Interval</th>
<th>Server</th>
<th>Free Memory</th>
<th>Number of Threads</th>
<th>Number of Request</th>
<th>Queue</th>
<th>Average response time</th>
<th>Load balance factor(lbf)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Server A</td>
<td>85531776</td>
<td>150</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>39.81611809</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>88744704</td>
<td>130</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40.18490755</td>
</tr>
<tr>
<td>T2</td>
<td>Server A</td>
<td>82514680</td>
<td>142</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>30.22525564</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>83269600</td>
<td>128</td>
<td>1</td>
<td>73</td>
<td>1</td>
<td>30.04981732</td>
</tr>
<tr>
<td>T3</td>
<td>Server A</td>
<td>78982328</td>
<td>177</td>
<td>8</td>
<td>47</td>
<td>5176</td>
<td>10.82091976</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>67839336</td>
<td>215</td>
<td>34</td>
<td>150</td>
<td>10892</td>
<td>9.255972441</td>
</tr>
<tr>
<td>T4</td>
<td>Server A</td>
<td>71428776</td>
<td>201</td>
<td>56</td>
<td>68</td>
<td>8399</td>
<td>10.07917878</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>70377360</td>
<td>308</td>
<td>67</td>
<td>84</td>
<td>13679</td>
<td>9.929976341</td>
</tr>
<tr>
<td>T5</td>
<td>Server A</td>
<td>73820408</td>
<td>218</td>
<td>72</td>
<td>82</td>
<td>10664</td>
<td>11.05458062</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>59795000</td>
<td>302</td>
<td>84</td>
<td>1</td>
<td>23650</td>
<td>9.192932559</td>
</tr>
<tr>
<td>T6</td>
<td>Server A</td>
<td>72263992</td>
<td>226</td>
<td>86</td>
<td>1</td>
<td>11834</td>
<td>20.09752333</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>70908400</td>
<td>302</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20.13701789</td>
</tr>
<tr>
<td>T7</td>
<td>Server A</td>
<td>70648744</td>
<td>229</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40.04284513</td>
</tr>
<tr>
<td></td>
<td>Server B</td>
<td>70048240</td>
<td>302</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>39.95744406</td>
</tr>
</tbody>
</table>
3.5 SUMMARY

Advanced techniques of resource management have become essential. An infrastructural solution to the needs of modern organizations is provided by on-demand computing. Resources are dynamically discovered by grid computing and utility computing and these trends maps to application in order to satisfy the QoS guarantees.

The implementation of a QoS-aware middleware that provides distributed applications with one such a resource management technique in the context of application server technologies has been discussed in this chapter. As a future augmentation, work can be carried out by evaluating and addressing the issues with more specific load balancing parameters and deploying multiple applications with QoS requirement.