6. RESOURCE SCALABILITY AT CAC

6.1 INTRODUCTION

Research communities working on the issues related to cloud scalability challenges have not investigated much towards the possible impact of mobile users accessing the cloud resources. Almost every smartphone users have begun to access the vast resources of cloud data centers, and this has become essential to develop new architecture and protocols for handling mobile consumers. The architecture so developed should focus fully on a different set of data arising out of mobile in the form of redundant and unstructured data in a dynamic and changing network. Hence it is a diverse platform where the workload needs to adjust based on the communication instead of focusing only on computation. Having said in a broader scope for mobile cloud resource-scalability problem, it is essential to address the need for scalability at a micro level, like the mobile client interacting with the CAC for offloading the tasks. Though the CAC is a resource-rich server, it is only in comparison with a resource-less mobile device, and hence it can handle only a limited number of offloading process. Thus for greater scalability, it is essential to migrate to cloud resources. In this chapter, an effective resource scalability algorithm is proposed and implemented into CAC that is built to function as a micro private cloud.

6.2 CAC - MICRO PRIVATE CLOUD

Surrogate system working in between the cloud and mobile must have certain features like resource scalability through auto-scaling functionality. Developing such a system without cloud like functionalities will certainly be a disadvantage when more number of mobile users connect for offloading their data for computation. Hence multiple virtual machines are necessary to address them, and these virtual machines must be instantiated when load arises and disengaged when load reduces. Cloud middleware is essential to do this process of handling the mobile users for offloading purpose. A suitable middleware that has the future scope to handle mobile users that is currently is the OpenStack.
The CAC is based on OpenStack implementation, which is currently the No.1, Infrastructure as a Service (IaaS) orchestrator used to setup an On-premises private cloud environment [87]. The implementation stack of the CAC is presented in Figure 6.1. The experimental setup is done on a dual-core Intel i3 processor, with 8GB memory. The host operating system is Ubuntu 14.04, on top of which is the OpenStack cloud middleware. Virtual machines instances that are created are mainly used for application services, where multiple applications are loaded as and when the recommendation engine identifies an application. This research uses mainly two different applications for case study namely, video converter and Book Optical Character Recognition (OCR). One VM is exclusively allotted for running the various CAC components namely the CAE and the CAAS. Other VMs are used for the computation of the incoming tasks sent from mobile devices.

<table>
<thead>
<tr>
<th>Video Converter/ Book OCR Application</th>
<th>CAE and CAAS</th>
<th>Computation VM1</th>
<th>Computation VMn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest VM 1 Ubuntu</td>
<td>Guest VM 2</td>
<td>Guest VM 3</td>
<td>Guest VMn</td>
</tr>
</tbody>
</table>

![Figure 6.1: Implementation stack of CAC](image)

6.2.1 OpenStack Component

OpenStack provides a rich set of features to implement cloud computing platforms for public, private and hybrid clouds. With various sets of interrelated services, it provides Infrastructure as a Service (IaaS), upon where the software platforms make the applications to run seamlessly using
virtual machines. OpenStack is divided into different components integrated through Application Programming Interface (APIs) as presented in Figure 6.2.

OpenStack cloud operating system supports various shared services like, compute, storage and networking services managed through a dashboard. Some of the major components are the “Horizon” dashboard, which provides administration interface, while “Nova” compute, automates computing resources using virtualization technologies. “Neutron” networking service manages the virtual connection between VMs and takes care of IP addresses. “Cinder” and “Swift” are storage services, where the former provides persistent block level storage, while the later is a scalable, redundant storage system. “Glance” Image provides a repository of server images, and “Keystone” provides authentication and authorization for all services and catalog of user’s credentials.

![Figure 6.2: Basic OpenStack components architecture](Source: www.openstack.org)

These services of OpenStack enable the applications to run seamlessly with massively scalable architecture using virtual machines deployed on top of the hypervisors namely Xen, KVM, LXC, and VMware. For the CAC server, OpenStack is built as a single node cluster with all the
components running inside a single machine. Hence it can be seen that the resources are extremely minimal when compared to a data center cloud infrastructure. However, the difference is only in terms of additional resource, while the control of the key implementation algorithms remains to be the same where a single node cluster is sufficient.

6.3 CLIENT AWARE RESOURCE SCALABILITY

Client aware cloudlet functioning as the Resource-rich Middleware (RM) server is a micro-private cloud implemented on commodity hardware with very limited resources. As discussed, most of the resources i.e., atleast 2 VMs are needed for running the client requesting applications and CAC components. Hence the need to start new VMs if available within or from its cluster is essential when there is an unexpected number of mobile clients request for offloading their task. The launching of new VMs when there is a load imbalance at the CAC is governed by Client Aware Resource Scalability (CARS) protocol. Moreover, CAC being an exchange point at the edge of the internet, the private cloud need to coordinate with the public cloud, making this into a hybrid cloud environment which is visualized in Figure 6.3. Though hybrid VM scalability is considered as a future scope of the research, it is one of the potential areas to explore. This is because; the idea of cloudlet is fast becoming an exchange platform to the cloud that is currently evolving as an upcoming computing paradigm called edge computing.

![Diagram showing resource scaling at CAC using CARS protocol](image)

**Figure 6.3: Resource scaling at CAC using CARS protocol**
6.3.1 CARS protocol

The Client Aware Resource Scalability protocol given in Algorithm 4 identifies the history of past interactions of the mobile clients and its corresponding load as the important client aware features. For example in the video conversion process, the load for converting a video file depends on the size of the file. The history of that load is dependent on the number of access made by those mobile users. Hence if the same user is arriving at time (t+1), then it can be assumed that a similar load can be expected given the past value of the same user. Therefore, the load which is nothing but the CPU utilization of the running virtual machine is to be calculated.

Algorithm 4: CA Resource Scalability Protocol

```plaintext
while application streaming at CAC
{
    get ( No. of mobile Users, No. of Application Instance running,
           No. of Application Service in Queue)
    for VM(p)
        Check CPU utilization in %;
        Define Upper Threshold = 80 %;
        Lower Threshold = 25%;
        {
            if CPU > 80% for more than 3 min
                then upscale and Initiate VM(k)
            else if CPU <25% for > than 10min
                then descale VM(p)
        }
}
```

Thus based on the evidence and likelihood a look-ahead approach for predicting the load is carried out using the auto scaling functionalities inbuilt in the OpenStack’s Heat component. This functionality is fine-tuned to predict the expected workload and initiate the VMs when the load exceeds the threshold. The upper and lower thresholds for virtual machine scaling are estimated based on multiple trails for the different input condition.
The CPU utilization of the CAC when multiple mobile client’s access are tested and the values are mapped for a time period of 3000 sec as depicted in the graph of Figure 6.4.

![CPU Utilization Graph](image)

**Figure 6.4: Initiation of VMs of CAC for CPU load conditions**

It can be seen from the graph that at time 0 the CPU utilization of VM1 is also 0 and as the 1\textsuperscript{st} client uses the server for computation, it uses around 30 seconds for the first file of 10MB size. As the 2\textsuperscript{nd} client with file size15 MB and 3\textsuperscript{rd} with 21MB uses VM\textsubscript{1} concurrently it reaches beyond 80%. The upper threshold for CPU utilization is fixed at around 80% for more than 3 minutes. During this time few more files namely 4\textsuperscript{th}, 5\textsuperscript{th} and 6\textsuperscript{th} files are converted thereby reaching the peak of 100%. Hence after 3 minutes, the 2\textsuperscript{nd} VM gets initialized for resource scalability, which starts at around 480 seconds. Similarly, the VM\textsubscript{3} starts at 780 seconds. The lower bound of 25% for 10 minutes shuts down VM\textsubscript{3} at 1500\textsuperscript{th} second.

### 6.3.2 Observation of CARS

Tests are carried out for multiple mobile clients offloading video files ranging between 10 to 420MB, which was used in Table 4.10. Based on the CA resource scalability protocol the CAC efficiently provided the necessary
service without any interruption. The following are the estimated CPU load conditions when offloading begins.

- Mobile Device 1 - CPU utilization around 30 to 40%
- Mobile Device 2 - CPU utilization around 75 – 90%
- Mobile Device 3 - CPU utilization > 90%

Hence for heavy duty applications like video conversion, only 2 to 3 consecutive conversion operations could be performed in a single VM, hence when the request is greater than 3 then additional VMs are instantiated based on the threshold conditions of CARS. Thus VM scalability is an essential process while designing the framework for CAC.

### 6.3.3 Cloud migration service

The concept of CAC is to utilize the available resources within the micro-private cloud environment for minimizing the cost and response time. This could well be achieved if the number of requests is minimal at any given time and if the requested service within the scope of availability. However, with the resource of a dual-core processor with 8GB memory, the CAC server will be of no use when there is a surge in request for processing High Definition (HD) videos or performing 3D rendering. Hence there must be always a cloud supported resources for execution that can able to support resource migration to external virtual machines.

A Client Aware Migration Service (CAMS) is deployed within the OpenStack middleware, for resource migration to the remote cloud datacenter. The CAMS already acts as an exchange service for helping the recommendation engine for downloading and deploying recommended applications into the application server. However, with respect to resource scalability, it performs migration of tasks based on the information provided by CARM via the internet to the cloud and completes the task, and stores it at the CAC. If the client’s context changes then based on CAAS the output can be further sent to the cloud storage. Thus CAMS is an essential service as long as the CAC is self-sufficient in terms of applications and resources.
6.4 BOOK OCR APPLICATION

Another application utilizing the CAC framework is an OCR app, targeting students who wish to review a book, presented in Figure 6.5.

![Diagram of the client aware cloudlet architecture for book OCR App](image)

**Figure 6.5: Client aware cloudlet architecture for book OCR App**

The main objective of the application is to advocate the users with different necessary features of the book such as ratings, reviews, and opinions along with the associated details of relevant authors and information. Initially, a mobile client, who is interested to know about a book of his choice, captures the image of the cover page of the book. The image is then sent to the CAC for extraction of the text information inside the book. This is done using Optical Character Recognition (OCR) software called Tesseract. The extracted text is collected and stored in the CAAS module. Further, it is sent to the cloud for getting the reviews, rating and other information available about the book from available web services. Then that information is sent back to the user for making a decision. The sequence of steps involved in the application is given below.
Step 1 – Client Side

Initially, at the client side, the user uploads the image from the mobile gallery to the server. The image is the front page of the book captured instantly and uploaded as given in the screenshot of Figure 6.6.

![Figure 6.6: File upload from mobile to server](image)

Step 2 – Server Side

At the server side, the received image is given to the Tesseract OCR engine for converting the printed text given into editable one. A test case image of the printed text is given in Figure 6.7 and its corresponding editable text is given through the mobile app screen shot in Figure 6.8.

![Figure 6.7: Tesseract OCR engine – captured image (printed text)](image)

![Figure 6.8: Tesseract OCR engine – converted image (editable text)](image)
Step 3 – Client Side

After conversion, the book detection system at CAC collects the information related to the book namely name of the book, its author, rating, reviews and further suggestion for reading. This information is sent to the users mobile which is depicted in Figure 6.9.

![Figure 6.9: Book detection with rating, review, suggestions](image)

The application at the mobile end is implemented using Android and the Tessaract engine is used on an Ubuntu 14.04 VM. The application is tested for around 25 books and the information retrieved from the cloud depends on the availability. However, the conversion of image to text by the CAC server has been satisfactory.

Thus the functionality of the proposed CAC has been tested with two compute-intensive applications for mobile and proved that with the help of CAC framework the mobile has saved considerable power and time. Moreover, the client need not worry about the resultant file if moved away from the CAC because it gets stored in the user’s personal cloud storage.
6.5 COMPARATIVE RESULT ANALYSIS

Results evaluated using the CAC offloading decision algorithm and CA security certification algorithm can effectively identify the performance of overhead and undecidability factor for offloading decision. Therefore samples of 15 different access frequencies are considered for comparative result analysis.

6.5.1 Offloading decision time (overhead)

The proposed security algorithm for accessing the CAC may inherit certain level of overhead. It can be seen that after a few frequent accesses, the time is drastically reduced as shown in Table 6.1.

Table 6.1: Comparison of overhead time for offloading decision

<table>
<thead>
<tr>
<th>Frequency of Access</th>
<th>CASC (sec)</th>
<th>COSMOS (sec)</th>
<th>MobiCloud (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5</td>
<td>1.5</td>
<td>4.5</td>
</tr>
<tr>
<td>2</td>
<td>2.25</td>
<td>1.25</td>
<td>4.25</td>
</tr>
<tr>
<td>3</td>
<td>2.25</td>
<td>1.25</td>
<td>4.1</td>
</tr>
<tr>
<td>4</td>
<td>2.1</td>
<td>1.5</td>
<td>4.25</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1.5</td>
<td>4.25</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1.25</td>
<td>4.1</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>1.25</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>0.5</td>
<td>1.5</td>
<td>4.25</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>1.5</td>
<td>4.1</td>
</tr>
<tr>
<td>10</td>
<td>0.6</td>
<td>1.25</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>0.5</td>
<td>1.25</td>
<td>4.5</td>
</tr>
<tr>
<td>12</td>
<td>0.65</td>
<td>1.25</td>
<td>4.25</td>
</tr>
<tr>
<td>13</td>
<td>0.5</td>
<td>1.25</td>
<td>4.1</td>
</tr>
<tr>
<td>14</td>
<td>0.5</td>
<td>1.3</td>
<td>4.25</td>
</tr>
<tr>
<td>15</td>
<td>0.55</td>
<td>1.2</td>
<td>4</td>
</tr>
</tbody>
</table>

The work is checked with the other offloading techniques for mobile cloud computing namely COSMOS [3] and MobiCloud [58].
Figure 6.10: Comparison of service overhead for offloading

Comparative result analysis in the graph of Figure 6.10 shows that the overhead for the CASC is initially high and then it reduces after a few accesses. The reason is that initially it needs to go through a security certification process for getting trusted credentials. After few accesses, the behaviour of the user is verified instantly and the checking process for offloading gets minimized, hence after 5 or 6 access (depends on RI) the time to take decision reduces. As the frequency keeps increasing, the time taken stabilized with a least possible service overhead. On the other hand, MobiCloud and COSMOS take the decision only on the basis of energy and power and does not concentrate on the behavioural pattern with regard to security. Hence the frequency does not impact the overhead time for offloading decision process.

6.5.2 Comparison of undecidability

The percentage of undecidability is a factor assessing the offloading decision algorithm for its proper judgement. There are situations where the algorithm fails to decide correctly due to test cases out of bound or null value. The CAC offloading decision correctly judges with the highest percentage of decidability factor at 95% which is depicted through graph in Figure 6.11. The COSMOS and MobiCloud have their algorithm with lesser decidability factor because of its dependence on the distant cloud servers.
6.5.3 Comparison of server setup cost

The cost in terms of expenses incurred for the setup is calculated based on the additional resources used. In the case of CloneCloud [88], the total cost is around 23.4$ because of its reliance towards Amazon cloud infrastructure. While the COSMOS spend more time in decision making at the local server, and hence to compute the offloaded task, it takes many number of VMs. Moreover, these VMs does not have a complete auto-scaling feature to dynamically start and stop the virtual machines based on the request. However, to execute all the processes of the CAC, the operational cost involved is very minimal when compared to other offloading servers, which is shown in the graph of Figure 6.12. This is because the work is deployed in an free and open source platform of OpenStack cloud infrastructure and the CARS protocol for resource scalability.

![Figure 6.11: Percentage of undecidability](image1)

![Figure 6.12: Expenses for offloading server setup](image2)
6.5.4 Availability of service

Availability is the degree to which a system or component is operational and accessible when required for use. In the proposed framework, consider the mobile clients accessing the server with a test case of 100 evaluations. Then the availability of the service can be given as the ratio of the service delivery to its service failure. Thus the proposed framework has got better availability than the other models, which can be visualized through the graph in Figure 6.13. The reason is attributed to the instantiation of additional VMs in case more resources are required.

![Figure 6.13: Availability of offloading and computation service](image)

6.5.5 Qualitative comparison

The comparative results shown in Table 6.2 substantiate that the proposed framework is the only one with security certification and recommendation as an additional feature for an intermediate surrogate server. Moreover, the CAC decision criteria takes into account of multiple client attributes, while the other models considers only battery and bandwidth latency. Very few models have the support for resource scalability and cloud migration, while the proposed has implemented it, and also can handle heavy network load conditions. Hence in comparison, the framework proved better over the existing works in the domain of MCC, oriented towards the features of offloading decision process.
Table 6.2: Comparative analysis of the existing offloading schemes and proposed CAC framework

<table>
<thead>
<tr>
<th>Offloading Decision Schemes</th>
<th>Deployment Environment</th>
<th>Application Type</th>
<th>Static/Dynamic Adaptation</th>
<th>N/W Load</th>
<th>Decision Attributes</th>
<th>Security Certification</th>
<th>Recommendation</th>
<th>Cloud Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSMOS [3]</td>
<td>Android</td>
<td>Face Recognition</td>
<td>Static</td>
<td>Low</td>
<td>Bandwidth, Energy</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hyrax [8]</td>
<td>Hadoop</td>
<td>Multimedia Searching</td>
<td>Static</td>
<td>Very High</td>
<td>Network Limitations</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VM based Cloudlet [10]</td>
<td>Virtual Machine</td>
<td>KPresenter, GIMP</td>
<td>Static</td>
<td>Low</td>
<td>Bandwidth</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Energy Aware [14]</td>
<td>Java, RMI</td>
<td>Image Processing</td>
<td>Static</td>
<td>Very Low</td>
<td>Energy</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Weblets [20]</td>
<td>C#</td>
<td>Augmented Reality</td>
<td>Static &amp; Dynamic</td>
<td>High</td>
<td>Latency</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MAUI [26]</td>
<td>.NET</td>
<td>Face Recognition</td>
<td>Dynamic</td>
<td>Very Low</td>
<td>RTT, Energy</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Secure MCC [58]</td>
<td>Android</td>
<td>Focus Drive</td>
<td>Dynamic</td>
<td>Low</td>
<td>Location</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Proposed CAC Server</td>
<td>Android</td>
<td>Video Conversion/OCR</td>
<td>Dynamic &amp; Static</td>
<td>Very High</td>
<td>Client and Context</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6.6 SUMMARY

- In this chapter, the need for micro-private cloud setup for implementing the CAC framework is discussed with industry standard cloud middleware components.

- The importance of resource scalability for the surrogate system when there is an unexpected demand from mobile clients is presented.

- Execution of Book OCR application proves that the CAC can handle image processing and as well any other compute-intensive services.

- The proposed framework comprises of various processes and hence a competent algorithm for comparison was not favorable, therefore, a part of the system is compared for its undecidability and availability. The outcome of the comparison shows that the proposed CAC is better under most of the conditions.

- Since the proposed work is a new approach that integrates recommendation service for enhancing the overall quality of the service, the benchmark data for evaluating the performance is yet to be standardized. However, most of the attribute selection and threshold conditions are as per NIST mobile security framework.

- Qualitative comparison points out that the proposed CAC framework is a unique method for offloading decision with security and quality enhancement service.