CLOUD BASED VIDEO ON DEMAND MODEL WITH PERFORMANCE ENHANCEMENT

R. Lavanya, and V. Ramachandran
Anna University, Chennai, India,
lavanya@annauniv.edu / lavi_82@yahoo.com, rama@annauniv.edu

ABSTRACT

Today Internet has become an integral part of most of our life. The attempt to display media files through Internet was started from the mid-20th century. Several research works have been reported to provide Video on Demand model in distributed environments like RMI, Component based, SOA and Grid computing. Inspite of these developments, it is been understood that very little progress has been made for several decades, primarily due to the high cost and limited capabilities of computer hardware and standards. Thus it is necessary to develop a model that is reliable, scalable and cost-effective. The proposed research work aims to implement the Video on Demand Service in Cloud Environment in a more secure, scalable and cost effective manner. The performance measures like upload time and download time are analyzed with different video files of varying sizes that are divided into block blobs of 32KB, 64KB and 1MB and are reported.

Keywords: Cloud Computing, Security, Video on Demand, Rest API, Performance Analysis.

1. INTRODUCTION

Video distribution is pointed out as one of the most promising applications of the Internet. Nevertheless, this application is a costly service to provide because of its quality-of-service requirements and the number of potential users. In order to guarantee scalability, providers often serve low-quality videos with no continuous reception. Thus, most of the users are not satisfied with the quality of received video [1]. Seethalakshmi et al (2002) has developed an effective Remote Method Invocation (RMI) based distributed model where the clients can access the remote server at any time for the required media file and for each and every request from the client for a media clip, the server marshalls the data to the client. The media server creates a new thread of control for every request and hence a complete distributed environment has been explored [2]. As RMI based solution has few limitations, the SOAP interface was preferred as it provides XML abstraction that allows implementing the client in any language. SOAP protocol does not include the security of the web transactions it carries and so it is the responsibility of any application to take care of the safety of its critical data being transmitted. Komathy et al (2008) has proposed a comprehensive security model based on cryptographic techniques for the distributed web services connected through a SOAP-RPC based communication network to safeguard sensitive information over the wire [3].

The distributed architecture for VoD system requires handling high CPU and storage load. Grid based architecture for VoD has been developed to provide a highly scalable solution and ensuring high availability and reliability through redundancy mechanisms [4], but still there is a requisite for distributed model for cost effective video on demand service that provides, scalability and reliability. Cloud computing developed from technologies such as virtualization and grid computing and business approaches that emerged over a number of years. The major building blocks from Internet technology to cloud service providers are Arpanet, Internet, World Wide Web, Web Services, SOAs and Cloud Services with different computing paradigms such as Utility, Grid, Autonomic Computing, Open Source Software, Platform Virtualization, and so on [5].

Lau et al [6] propose a new strategy to integrate cloud solutions to curb the rising IT costs, the complexity of network management, and infrastructure inefficiencies for IPTV. Scalability is one of the major issues along with cost in the current VoD System [6]. Thus, the system has to be designed to handle the increasing number of clients and providers without any hurdle to exchange video data in heterogeneous environment. The conventional data center does not provide dynamic scalability.

This work attempts to define a Cloud based model for Video on Demand (VoD) as a Service which enhances the security and scalability of the VoD applications. This model simulates the real theatre environment wherein the full control including the rewinding, fast forward, play and stop is with the distributors. As the control is not given to the client, the user cannot pause, forward or rewind the video. All the clients who have requested for a video will see the same scene at a given time as in the case of multicasting.

2. IMPLEMENTATION OF THE PROPOSED CLOUD BASED VoD MODEL

In the proposed model, the video data are stored in distributed locations effectively through content delivery network. The server in the cloud environment along with the administrator (distributor) has the control and can manage the video stored in the "Storage as a Service" layer. It supports the streaming of all video file formats such as mp4, avi, wmv etc. The advantage of this cloud based model is, even if thousands of requests are placed on the server at a time, the server will be able to stream the video simultaneously to all the clients. The providers can upload the videos of any length and size in the form of block blobs to the cloud storage that can be accessed from anywhere at any time with provider's permission. The providers and the consumers details are stored in the structured cloud table.

The VoDaaS is deployed in the azure cloud environment and is provided on top of an effective "Infrastructure as a Service" layer that manages virtualization of resources and multi-tenancy. The single-tenant model has a separate, logical instance of the application for each customer, while in the multi-tenant model, a single logical instance of the application shared by many customers. The VoD clients can access the VoDaaS through the WebRole, which is driven by UI – the user interacts with a web page or service and this WebRole helps in processing the request and response. Thus, the WebRole is the single HTTP endpoint for the external clients. The VoD Service model is determined by the settings listed in the service definition file, which will be packaged with the WebRole binaries when the VoDaaS is deployed to the cloud. The Service Definition is named as "LiveCloudCinemasWebRole", which will access the storage service via Storage Library and Representational State Transfer (REST) APIs. The service definition of the VoDaaS shown below in Fig. 1, defines the service name, the communication endpoints of a service and the various configuration settings.

```
<ServiceDefinition name="livecloudcinemas" xmlns=
"http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
    <WebRole name="LiveCloudCinemasWebRole" vmsize="small">
        <InputEndpoints>
            <InputEndpoint name="HttpIn" protocol="http" port="80" />
        </InputEndpoints>
        <ConfigurationSettings>
            <Setting name="DiagnosticsConnectionString" />
            <Setting name="DataConnectionString" />
            <Setting name="AccountName" />
            <Setting name="AccountSharedKey" />
            <Setting name="BlobStorageEndpoint" />
        </ConfigurationSettings>
  </WebRole>
 </ServiceDefinition>
```

Fig. 1: Service Definition for the proposed VoDaaS Model

The size of the required Virtual Machine is defined as "small". The size of the compute nodes enables the scale up or scale down of the individual instances. The service configuration shown in Fig. 2 for the LiveCloudCinemasWebRole includes number of instances to be deployed for each webrole and the values for the storage resources like account name, account shared key and the URL of the BlobStorageEndpoint, as the VoDaaS uses both cloud blob and table storage.

```
<ServiceConfiguration serviceName="livecloudcinemas"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="LiveCloudCinemasWebRole">
    <Instances count="5" />
    <ConfigurationSettings>…… …. ….
      <Setting name="AccountName" value="cloudmovies" />
      <Setting name="AccountSharedKey"
          value="nh4csiEUJ8qgCJDXTUEMH+5S0JybmI4gG7tCJ
              6e8wAl716jtphtCOliUp3KafwiXFhls90BXH+J2r8F5sVQVqw==/>
      <Setting name="BlobStorageEndpoint" value="http://blob.core.windows.net" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

Fig. 2: Service Configuration for the proposed VoDaaS Model

The incoming traffic to the VoDaaS is forwarded to the WebRole through the load balancer that ensures functionalities like proper load distribution and fault tolerance. The WebRole instance is defined as five in the service configuration. The Azure load balancer therefore balances the requests across these five webrole instances. The values for the service definition are configured when the WebRole instance is running. Therefore, unlike the service definition, the values of the service configuration can be altered during runtime. For instance, the number of role instances can be increased or decreased, or the blob storage endpoint URL can be changed. The VoDaaS along with the Azure IaaS, provides multi-tenancy by allowing the video service to be available at a specific time period to all the registered viewers. The multi-tenant application is more vulnerable to instance failure than a single-tenant application. If a single-tenant instance fails, only the customer using that instance is affected, whereas if the multi-tenant instance fails, all customers are affected. To overcome this issue, multiple, identical copies of the VoDaaS has been deployed into multiple Windows Azure role instances (in this case, it is 5 instances), thereby achieving multi-tenant /multi-instance model and increasing the reliability.

The Service Definition and the Service Configuration helps in the setting of the necessary parameters for the successful deploying and running of VoDaaS. The below sessions discusses on the implementation of the Video on Demand model in a secure and scalable manner. The session (a) deliberates how the VoDaaS access of the cloud table storage to store the data and to fetch them. The session (b) exemplifies how the video file is divided into small blocks and is uploaded to the cloud blob storage using the Representational state Transfer API. The session (c) depicts how the video is secured, so as not to allow any anonymous clients accessing the video file. Finally in the Session III, the performance analysis of the deployed VoDaaS is measured using the parameters like upload and download time.

2.1 CLOUD TABLE STORAGE AS A SERVICE

As the video files will be delivered to the registered users; the details of the booking and the details of the VoD provider and consumer themselves are stored in the azure table storage. The following Fig. 3 shows the details of two tables stored in the “cloudmovies” storage.

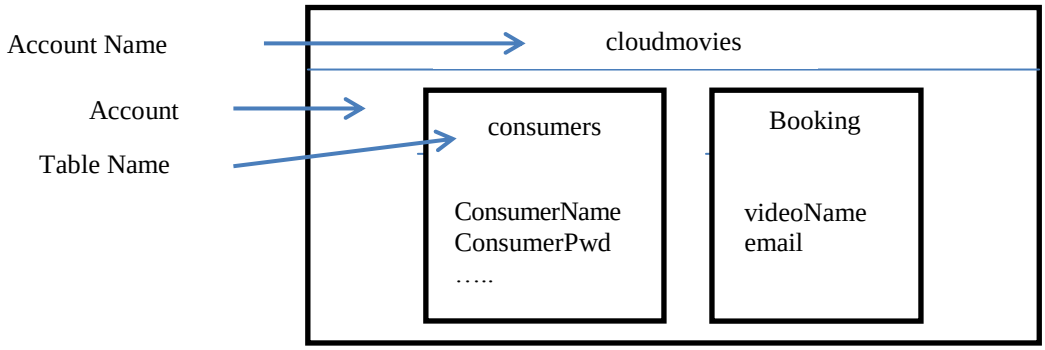

Fig. 3: Cloud Table Storage Model

To store the VoD provider and VoD consumer details in the azure table, the entity class named "clsconsumer" is inherited from 'TableServiceEntity' class and defines all the table entities (columns), such as ConsumerPwd and ConsumerName. Every row in the table needs to be defined with a partition key and a unique row key, which uniquely identifies an individual row. The email id of the client is stored in RowKey and depending on whether the type of client (VoD consumer or provider), is stored in PartitionKey. Thus, both the PartitionKey and RowKey together identifies a unique client. The definition of clsconsumer class is shown below in Fig. 4.

```
public class clsconsumer : TableServiceEntity    {
    private string _strConsumerPwd;
    private string _strConsumerName;
        …….. …. …..
    public string ConsumerPwd { get; set; }
    public string ConsumerName { get; set; }
        …….. …. …..
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
}
```

Fig. 4: Definition of the clsconsumer class

The class "clsconsumerDataContext", shown in Fig. 5 is used to insert the details into the cloud storage is shown below. The data context class will upload all the entities that are defined in the entity class "clsconsumer" to the cloud table storage. The constructor ensures the validations of the Table Endpoint represented as baseAddress and storage credentials which will be passed by the provider during insertion of a record. The 'Iqueryable' interface is used by the azure cloud service to create table named "Consumers" in azure table service, whose columns will be the entities defined in the entity class "clsconsumer". The 'AddConsumer' method is created that takes in the consumer entity object and call's the 'AddObject' method of the data context to insert the details of the consumer into Azure tables.

```
public class clsconsumerDataContext : TableServiceContext{
    public clsconsumerDataContext(String baseAddress, StorageCredentials Credentials) :
            base(baseAddress, Credentials) {
            }  public IQueryable<clsconsumer> Consumers {
                get{
                        return this.CreateQuery<clsconsumer>("Consumers");
                }}
                public void AddConsumer(clsconsumer objconsumer) {
                        this.AddObject("Consumers", objconsumer);
                        this.SaveChanges();
```

Fig. 5: Definition of the clsconsumerDataContext class

From the Frontend, when the user clicks on the submit button "Register User"; the following snippet of code is triggered. As a first step the authentication like account name and account key are fetched from the DataConnectionString that is specified in the ServiceConfiguration file and checks if there is permission to work with the Cloud Storage Account. The clsconsumerDataContext constructor is triggered by passing the Storage Credentials and TableEndpoint Address. Finally on checking if the client is a new user, the user details are added to the cloud table by calling the AddConsumer method shown in Fig. 6.

```
var Consumer = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
var ConsumerContext = new clsconsumerDataContext(Consumer.TableEndpoint.ToString(),
                          Consumer.Credentials);
clsconsumer objconsumer = new clsconsumer();
…… …….. ……
objconsumer.PartitionKey = radioButtonList1.Text;
objconsumer.RowKey = txtEmailId.Text;
objconsumer.ConsumerPwd = pwdpwd.Text;
…… …….. ……
ConsumerContext.AddConsumer(objconsumer);
```

Fig. 6: Code Snippet for the insertion of record to Cloud Storage

Similar way, when the user books a ticket for a particular video, all the details regarding the ticket booking are stored in the cloud table service and counterchecked when requesting for viewing a video.

2.2 API Using Representational State Transfer

In Representational State Transfer (REST), a client sends a document to a server, called as request, and the server replies with another document, called as response. Both the request and the response documents are "representations" of either the current or intended "state". The REST protocol is selected as it emphasize on simple point to point communication over HTTP using plain old XML and it is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al). The following code snippet, shown in Fig. 7 converts the file which the user uploaded (denoted as filename) into stream of bytes (denoted as value). As the video file is divided into small chunks of 1MB each, the number of blocks is identified and assigned with a unique Id.

```
……
 FileStream fs = new FileStream(filename, FileMode.Open);
byte[] value = new byte[fs.Length];
fs.Read(value, 0, (int)fs.Length);
Stream stream = new MemoryStream(value, false);
 ……
int numBlocks = (int)Math.Ceiling((double)length / blockSize);
string[] blockIds = new string[numBlocks];
string accountName = "cloudmovies";
string basekey = "nh4csiEUJ8qgCJDXTUEMH+5S0JybmI4gG7tCJ
                    6e8wAl716jtphtCOliUp3KafwiXFhls90BXH+J2r8F5sVQVqw==";
……
for (int i = 0; i < numBlocks; ++i)  {
   String block1 = "Lavi Cloudmovies"+i;
   byte[] toEncodeAsBytes = System.Text.ASCIIEncoding.ASCII.GetBytes(block1);
    string blockId =  System.Convert.ToBase64String(toEncodeAsBytes);
   blockIds[i] = blockId;
   if (stream.CanSeek)
            stream.Position = startPosition + i * blockSize;
   long blockLength = Math.Min(blockSize, length - stream.Position);
……
```

Fig. 7: Conversion of video file into Block Blobs

After dividing the video files into required number of blocks, each of these block blobs has to be uploaded to the cloud storage service, which is performed through the Http Request that is built using the REST API. The URI in the Fig 8 includes the URL of the blob along with the container name followed by the filename. The URI also contains two parameters namely comp and blockId. The parameter comp is assigned with the value “block” indicating the

cloud storage that a single blob block is to be uploaded and the Id is given as second parameter namely blockId. The following code shows the header and the contents of the HTTP PUT request that is built using REST API.

```
Uri uri = new Uri("http://cloudmovies.blob.core.windows.net/tamilmovies/" + onlyfilename +
"?comp=block&blockId=" + blockId);
HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
request.Method = "PUT";
request.Headers.Add("x-ms-date", DateTime.UtcNow.ToString("R", CultureInfo.InvariantCulture));
request.ContentLength = blockLength;
byte[] key = Convert.FromBase64String(basekey);
……..
Microsoft.Samples.ServiceHosting.StorageClient.ResourceUriComponents uriComponents = new
Microsoft.Samples.ServiceHosting.StorageClient.ResourceUriComponents(accountName,
"tamilmovies", null);
SharedKeyCredentials credentials = new SharedKeyCredentials(accountName, key);
credentials.SignRequest(request, uriComponents);
……..
```

Fig. 8: Creation of HTTP PUT Request using REST API

The raw byte array of the uploaded video is stored in the private container named “tamilmovies”, which can be accessed only using the account name and account shared key. The storage credentials are set and the actual stream of video data are added to the Http body of the Http Request and uploaded to the azure blob storage, as shown in the above code.

The stateless characteristics of REST increases the security further more by accepting all the information required to understand the request submitted by the client each and every time. By uploading files through REST API, the application guarantees the successful uploading of files from any platform. It also ensures quicker and faster action. Adding REST components indicates the possibility of scalability. The above created HTTP request is sent and the corresponding response is obtained. When the response is successful, the corresponding video block data is uploaded to the cloud blob storage. The uploaded block blob data will be committed only after sending the HTTP PUT blocklist request. The request contains all the blockIds in an XML format. The HTTP body of the HTTP PUT blocklist request is shown in the Fig. 9.

```
MemoryStream s = new MemoryStream();
XmlTextWriter xw = new XmlTextWriter(s, Encoding.UTF8);
xw.WriteStartDocument();
xw.WriteStartElement("BlockList");
for (int i = 0; i < blockIds.Length; i++)
    xw.WriteElementString("Block", blockIds[i]);
xw.WriteEndElement();
  …….
```

Fig. 9: Insertion of block ids to the HTTP body of the PUT Blocklist Request

Using XmlTextWriter, an XML document is attached to the HTTP request that contains all the blockIds to be committed.

2.3 Video Security

Usually, in the client side, when the video is cached, there is a very high possibility that the video will be pirated. When “Video on Demand as a Service” is offered to larger audience, the caching is a problem and it is not feasible to support real time encryption with unicast connections. Thus to increase the security, the private key encryption is adopted. Depending on the level of authorization a user is assigned, he or she is granted one or more permissions to perform specific operations or actions. These actions typically map directly to important business functions, or to the

management of the application itself. For example, a consumer can only read a video blob data to specified time duration that the provider permits. A provider can upload, read and delete a video data depending on his permission scope. Managing the access control is given based on the scope of the roles. Each scope inherits roles, permissions, and business rules from their parent. The following Fig. 10 illustrates a sample access control provided to the clients.

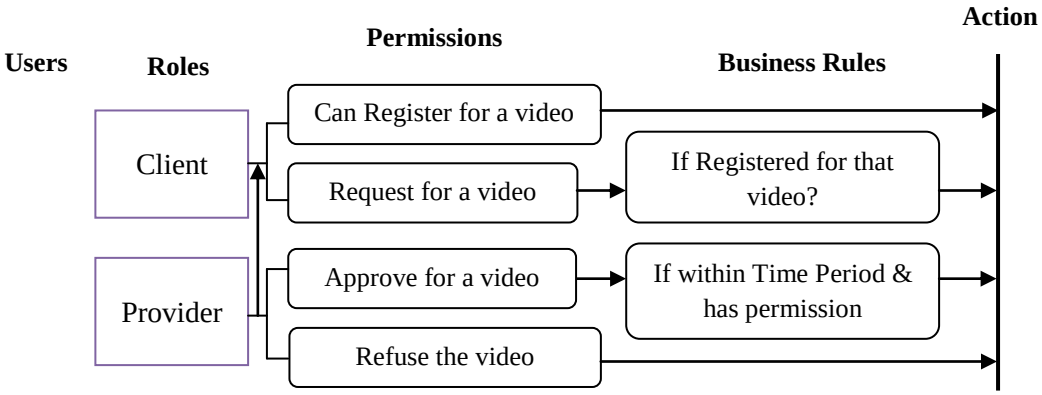

Fig. 10: Authorization – Access Control

The signed access signature is used to provide the read permission of the Video files to all registered users. The time period for which the video can be accessed is also specified. Only those who have this public key can watch the video. This public key is created by passing the private key of the distributor. In the following URL, it says that permission is given only to rithu.mp4 blob and only the read permission for the video is specified.

http://cloudmovies.blob.core.windows.net/tamilmovies/rithu.mp4?sr=b&si=readaccess&sig=f3M37MRO%2BuWs4d2%2FLaCDGLfPYp%2BAxjsfeUgSm2ihAmQ%3D

The string-to-sign is denoted as “sig” is a unique string constructed from the fields like movie starting and ending time, signed identifier along with the username and Private key that are verified in order to authenticate the request. The signature obtained is an HMAC computed over the string-to-sign and public key is obtained by using the SHA256 algorithm, and then encoded by using Base64 encoding. The following snippet of code in Fig. 11 shows partly a few attributes for setting the permissions for the video files.

```
……
var permissions = cont.GetPermissions();
if (!permissions.SharedAccessPolicies.ContainsKey("readaccess")) {
    permissions.SharedAccessPolicies.Add("readaccess", new
        SharedAccessPolicy()
            {
               Permissions = SharedAccessPermissions.Read,
               SharedAccessStartTime =  moviestarttime,
               SharedAccessExpiryTime = DateTime.UtcNow +
                     TimeSpan.FromHours(3)
            });   ……..
```

Fig. 11: Attributes for setting the permissions for the Video file.

3. Video Fetching

The Silverlight application, which was initially released as a video streaming plugins, now comes as a Web application framework that provides the functionality to integrate the video file into a single runtime environment. This player is used to play bytes of stream arriving at the clients end. The declarative programming language XAML is used to execute the streaming of video file. XAML is an XML dialect which provides a way to bind presentational

data and the declarative list of UI elements with some or all of the code used within them. This is a way to increase the processing power of the client as well as the server.

When the client fetches the video at the correct stipulated time, then the whole video will be streamed to the system, else if the client arrives earlier, then he has to wait till the stipulated time arrives. If the client arrives late, then he cannot view the whole video. If he comes half an hour late, then that half an hour video is not fetched for him. As the control is not given to the client, he cannot pause, forward or rewind the video. It simulates the real theatre environment wherein the full control including the rewinding, fast forward, plays and stop is with the distributors. Thus, all the clients requested for a video will see the same scene at a given time. This is executed with the following formula, where the Media Player Position or the video position is represented as MP_{pos}, video Start time and End time are represented as M_{ST}, M_{ET} respectively. The T_c is the time when the client logs in. The T_T represents the total length of the movie.

$$\mathbf{MP_{pos} = (M_{ET} - M_{ST}) * ((T_c - M_{ST}) (in Sec)) / (T_T (in Sec))}$$

The Progressive downloading approach is adopted to screen the video. Progressive download provides a quicker start for playback than a full "Download and play" method. Progressive download works better on networks where the average available speed equals or exceeds the speed of the stream. If the web server is slow or the end user is on a slow Internet connection, then it's possible for the end user to notice buffering. This problem is avoided to certain extent by using Content Delivery Network (CDN). The CDN is configured to distribute the video files to multiple locations and to assign a URI for each one of them. When a client keeps a request, the CDN evaluates the location of the request and routes it to the location that will provide the lowest latency and best connection speed for the requester. The file is then transferred directly from the location's server to the user who has requested it, thereby improving performance, scalability, and cost efficiency, to end users.

4. PERFORMANCE ANALYSIS

This section analyzes and measures the performance of the proposed Video on Demand as a Service (VoDaaS). The performance measure is based on the following criterions.

- The upload time and download time of video files of varying sizes that are divided into different block blobs.
- The average upload and download time for a chunk of different block blob size of different video files.

The different video files starting from 1MB to 75MB are uploaded to cloud storage by dividing it into block blobs of 32KB, 64KB and 1MB. The time taken to upload the video files of varying sizes from Windows Azure cloud storage is shown in Fig. 12.

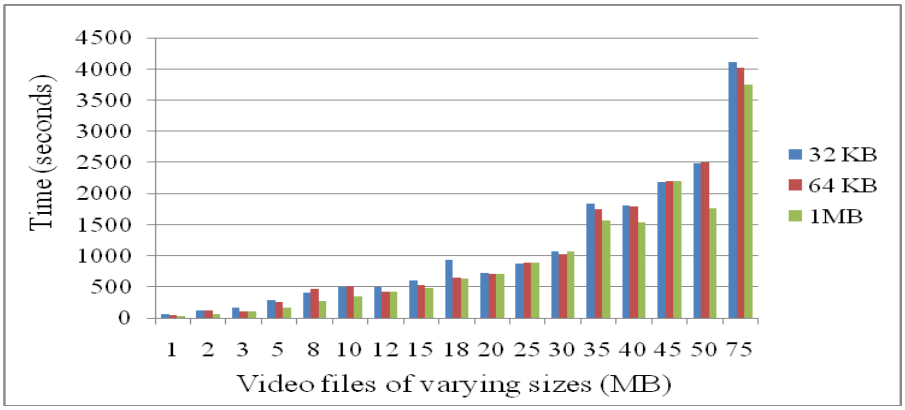

Fig. 12: Upload time for storing the blob data to cloud storage

The above result shows that the upload time of the blob data increases as the size of video file increases and in most of the cases, for a particular video file size, the upload time increases as the number of block blob increases. Therefore, the total time taken to upload 75MB file which is divided into 32 KB is higher when compared to other uploads in the given scenario. It is mandatory to divide the video files that are above 64KB into block blobs to upload it to cloud storage and the maximum size of a particular block blob is 4MB. While deciding to divide the blob into block blobs, to achieve faster and better upload time, reduce the number of block blobs or keep the block blob as higher as it can accommodate. Similarly, the time taken to download the video files of varying sizes from Windows Azure cloud storage is shown in Fig. 13.

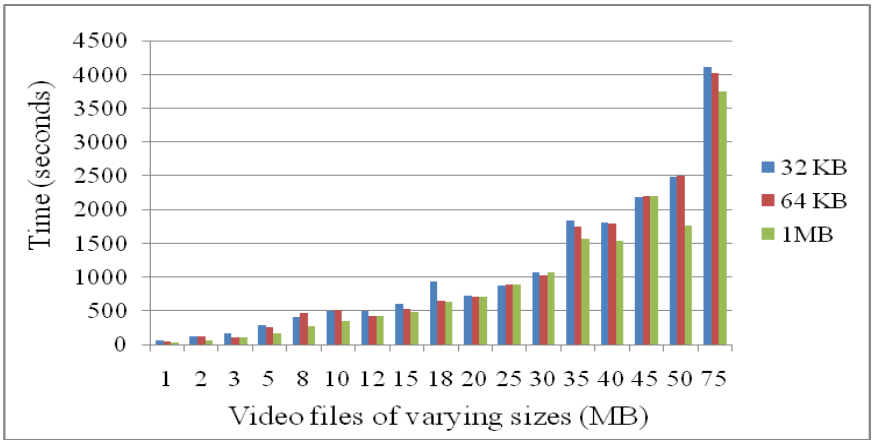

Fig. 13: Download time for fetching the blob data from cloud storage

Similar to the upload time, the download time of the blob data increases as the size of video file increases and in most of the cases, for a particular video file size, the download time increases as the number of block blob increases. Therefore, the total time taken to download 75MB file which is divided into 32 KB is higher when compared to other uploads in the given scenario. Downloading time is lesser when compared to the uploading time of the video file. The Factor Variation of upload and download time between 32KB and 1MB is calculated and is graphically represented in the Fig. 14.

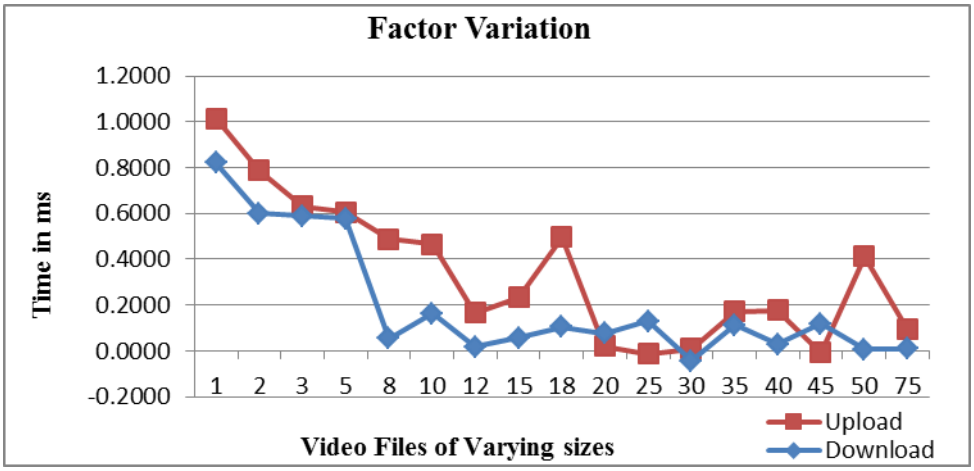

Fig. 14: Factor variations between 32KB and 1MB for both upload and download time

It is found that the download or the upload time for the video files that are divided into block blobs of 32 KB or 1MB has only slight variations and the variations also reduces as the video file size increases. Therefore, when the size of the video file is higher, the upload or the download time does not differ much based on the size of the block blobs. As the different video files starting from 1MB to 75MB that are uploaded to cloud storage by dividing it into block blobs of 32KB, 64KB and 1MB , the average time taken to upload a particular chunk of blob data to Windows Azure cloud storage is shown in Fig. 15.

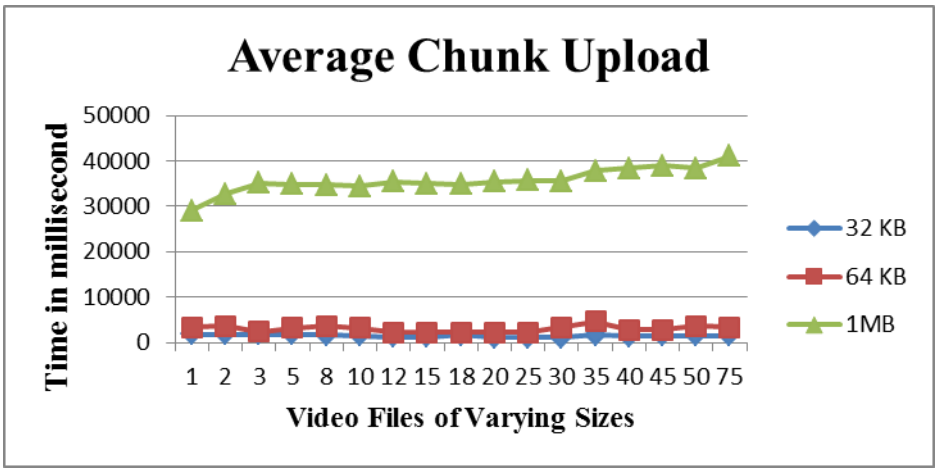

Fig. 15: Average upload time for a chunk

The average time taken for uploading a chunk of 32 KB and 64 KB data are below 5000 millisecond and to upload a chunk of 1 MB to cloud storage is seven times higher than uploading a 32 KB and 64 KB block blob. Even though, the upload time of video files that are divided into 1MB of block blob is less, the average time to upload a chunk of block blob is higher. Therefore, it is found that it is faster to upload a larger file than to upload multiple small files to the cloud storage. Similarly, the average download time for a particular chunk of block blob data is shown below Fig. 16:

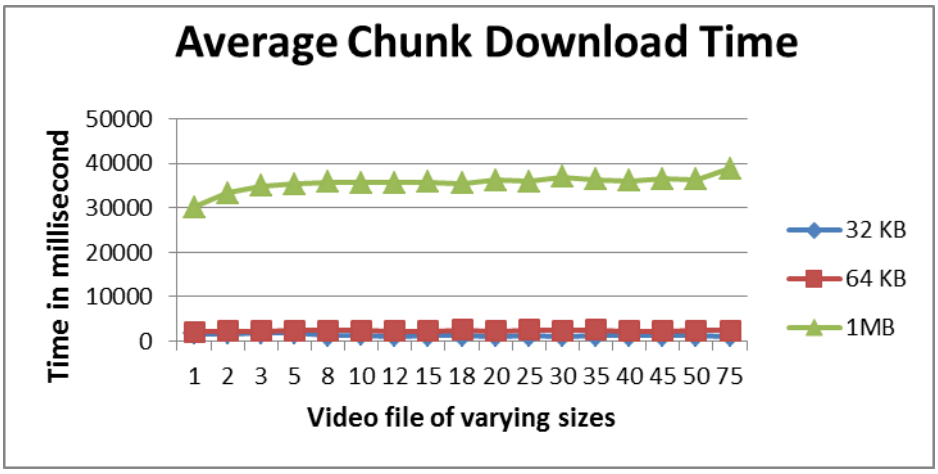

Fig. 16: Average download time for a chunk

The average time taken for downloading a chunk of 32 KB and 64 KB are below 5000 millisecond and to download a chunk of 1 MB to cloud storage is seven times higher than uploading a 32 KB and 64 KB block blob. Even though, the download time of video files that are divided into 1MB of block blob is less, the average time to upload a chunk of block blob is higher. Therefore, it is found that it is faster to upload or download a larger file than to upload or download multiple small files to the cloud storage. For example, to upload a 1 MB file, if it is divided into block blobs of size 32 KB, on an average it is fifty eight seconds. Similarly, of the same 1 MB file, if it divided into block blob of 1MB, on an average it takes twenty nine seconds, which is almost double the time.

5. CONCLUSION

An effective Video on Demand service is developed and implemented in Cloud Environment to provide a smooth and lively streaming of video files to the clients. The performance measure like upload time and download time are analyzed with video files of varying sizes from 1MB to 75MB each divided into block blobs of 32KB, 64KB and 1MB. The results pertaining to the above are reported which shows that the time taken to upload or download a single large file is lesser than the time required to upload or download multiple small files. It is also found that the download or the upload time for the video files that are divided into block blobs of 32 KB or 1MB has only slight variations and the variations also reduces as the video file size increases.

REFERENCES

[1] Igor Monteiro Moraes, Miguel Elias Mitre Campista, Marcelo Duffles, Marcelo Rubinstein, Luis Henrique Costa and Otto Carlos Muniz Bandeira Duarte, "Peer-to-peer Video Streaming: Architectures, Mechanisms, and Challenges", *Brazilian Symposium on Computer Networks and Distributed Systems*, 2008.

[2] P. Seethalakshmi and V. Ramachandran, "RMI Based Load Sharing and Caching for Media on Demand" in the proceedings of the International Conference *Digital aided Modeling and Simulation DAMS2003* January 6th – 8th, India, 2003, pp. 109.

[3] K. Komathy, P. Vivekanandan, V. Ramachandran, "Secure SOAP-Based Web Services", *International Journal on Computer Systems, Science and Engineering*, Vol. 18, No. 1, 2003, pp. 27-33.

[4] Grid Computing Focus Group, "Distributed Video on Demand – A Grid based VoD Solution", *Software Engineering and Technology Labs*, April 2006.

[5] Ronald L. Krutz and Russell Dean Vines, "Cloud Security", *Wiley Publishing Inc*., 2010.

[6] Phooi Yee Lau, "Pay-As-You-Use On-Demand Cloud Service: An IPTV Case", *International Conference on Electronics and Information Engineering*, 2010.

[7] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic."Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, Elsevier Science, Amsterdam.

[8] Benslimane, Djamal; Schahram Dustdar, and Amit Sheth. "Services Mashups: The New Generation of Web Applications", *IEEE Internet Computing*, Vol. 12, No. 5, 2008.

BIOGRAPHIES

Lavanya Rajendran received her Bachelor's degree in Computer Applications. She completed her M.Sc. in Electronic Media and Informatics. She is currently working as Assistant Professor in College of Engineering, Anna University, Chennai. Her areas of interest include Cloud Computing, Computer Networks, Web Designing and Video Production.

Ramachandran Veilumuthu received his Masters degree and Ph.D. in Electrical Engineering from College of Engineering, Anna University, Chennai, India. He is currently working as a Professor in the Department of Information Science and Technology, College of Engineering, Anna University, Chennai. His research interest includes Cloud Computing, Network Security, Soft Computing and Web Technology.