CHAPTER 4

VIDEO ON DEMAND MODEL IN CLOUD ENVIRONMENT

4.1 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 QoS 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 (Moraes et al 2008).

Lau et al (2010) proposed a new strategy to integrate cloud solutions to curb the rising IT costs, the complexity of network management, and infrastructure inefficiencies for Internet Protocol Television (IPTV). Along with cost, scalability is the other vital issue in the current Video on Demand system. 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 centre does not provide the dynamic scalability.

Cloud Computing has become increasingly affordable and dynamic scale up and scale down has become practical. Throughout the world, the concept of cloud has made its emergence and it is booming in every area. Due to the increase in VoD services, there is greater need to control and manage multiple video files requested by clients at the same time. Major
companies like Yahoo and Microsoft have already made most of their data and information already available in the cloud and many IT industries like IBM and TCS are making their foot prints in this field. Many industries are adopting ‘Storage as a Service’ for storing their numerous data and ‘Platform as a Service’ for running their applications.

HP is offering a cloud based music and video services to Touchpad which is not been managed by Apple (Dixon 2011). Apple is gearing up to launch a cloud based video service, focusing on delivering Video on Demand movies to connected TVs and Blu-ray players, and it is planning to expand the services to PCs by allowing subscribers to watch movies on Web browsers (Donohue et al 2011). The 8x8 Inc., provider of business communications and managed hosting solutions, now offers small and medium businesses (SMB), a set of affordable cloud based video conferencing services through private cloud that enable the organizations to benefit from visual collaboration (Vazquez 2011).

The demand for video service is very high among the video providers and the video clients. Hence, to solve some of the problems faced by the existing VoD models, the study explores a new architectural design for a VoD system that will enable reliable, secure and smooth streaming of videos for very large audience and it is proposed to develop a secure and scalable model for Video on Demand in cloud environment thus making Video on Demand as a Service (VoDaaS). It is also aimed to reduce or eliminate the movie piracy issues and to provide a smooth and lively streaming of videos to the clients in a cost-effective manner.

The work attempts to define a cloud based model for Video on Demand 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. Thus, all the clients who have requested for a video will see the same scene at a given time. Figure 4.1 shows the block diagram representation of the proposed Video on Demand model in the cloud environment.

![Proposed Cloud based VoD model](image)

**Figure 4.1 Proposed Cloud based VoD model**

The proposed Video on Demand as a Service model utilizes cloud hosting and the cloud storage services. The video files are stored as Binary Large OBjects on remote host by adopting replication strategies to ensure reliable operation of VoD system. The cloud based Video on Demand system is designed to meet the following criteria:

- **Scalable**: The Video on Demand system needs to be stable to support any number of users and should provide with video, which is non-invasive. Thus, VoDaaS has to scale upward during peak demand and downward while lighter demand.

- **Cost-Effective**: As the cloud has the ability to pay for use of computing resources on a short-term basis as needed and to release them when the task is completed, thereby rewarding conservation by letting machines and storage go when they are no longer used (Armbrust 2009). The VoDaaS is cost-effective to both end users and the video providers.
- **Reliable**: The VoDaaS model is 99.99% reliable, as it is very important to increase the number of users to the service and does not allow the potential clients to go elsewhere.

- **Secure**: The VoDaaS is intended to provide proper privacy preferences to all of the videos and also setting preferences for future uploads, thereby avoiding piracy of videos.

In order to meet all these requirements and to confine to the Web service standards, the design options such as delivery, platform, media formats and development language as suggested by Attewell (2005) are incorporated into the proposed VoD system. The adopted technologies for the proposed Video on Demand system are given in Figure 4.2. All the major video file formats can be uploaded to and downloaded from the cloud storage. The videos can be accessed from different gadgets including PCs and iPhones and the languages used for developing shall be Java, ASP.NET, C# and XAML.

![Figure 4.2 Technology Selection for VoD system](image-url)
The cloud environment is capable of providing required OS instance and the appropriate media player to support the format of the media file requested by the client. The proposed model also provides the support for language independency for client side applications and device independent environment for accessing the response.

4.2 WORKFLOW DESCRIPTION OF THE PROPOSED VIDEO ON DEMAND MODEL

A cloud based Video on Demand model is proposed to enable the clients to access the services anytime anywhere. The primary focus of introducing cloud environment is on reducing the cost of providing efficient Video on Demand services. The proposed model serves the video to the client as per the workflow description given in Figure 4.3 as it is intended to simulate the real theatre environment. When the video requester enters the VoDaaS, a new instance is initiated and the user can basically do three different tasks, namely Register himself to view a particular video file, marked as 1, 1A or if he is a registered user, then, he can directly login to watch the video, for which he has registered that has been marked as events 2, 2A and 2B or directly view the list of videos available marked as 3. The video providers are those who will upload or submit the videos to the cloud storage. VoD Requesters are those who will access the videos for a stipulated time. Once the client logs in to the VoDaaS as provider, it is possible for him to upload or see the videos marked as events 4, 5 and 6 in Figure 4.3.

If the client is a consumer, then he is given permission only to see the videos and he cannot upload any video. Before watching the video, the client has to login and to confirm his registration for the video, which are marked as events 7 and 7A. Once he is logged in and confirms the registration, the requested video stored in the cloud storage is fetched to the
user marked as events 8A and 8B. Depending upon the time at which the client arrives, three different tasks are carried out, that is marked as 9. The video is not streamed to those who arrive earlier than the video start time and those who arrive after the video end time (marked as 10A and 10C). Those who arrive on time can watch the video from the beginning, which is marked as 10B. Those who come late, will be screened only from the scene that other viewers are screened with, which is marked as 10C and those who come early, has to wait till the starting time to watch the video.

Figure 4.3 WorkFlow for Video on Demand as a Service
To design, develop and implement the cloud based Video on Demand as a Service that is scalable, secure and cost-effective, few requirements are essentially needed. These requirements are matched with the features that are used to implement the service, as shown in Table 4.1.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Requirements</th>
<th>Features</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Development Environment</td>
<td>Compute Emulator</td>
<td>The development fabric included in Azure SDK is used to develop the VoDaaS.</td>
</tr>
<tr>
<td>2</td>
<td>Data storage</td>
<td>Table Storage, Client Library</td>
<td>A Windows Form is used to store the data into TableStorage using Client Libraries.</td>
</tr>
<tr>
<td>3</td>
<td>Video upload</td>
<td>Blob storage, REST API</td>
<td>A Windows Form is used to upload the videos into blobs using REST API.</td>
</tr>
<tr>
<td>4</td>
<td>VoD Client portal UI</td>
<td>WebRole</td>
<td>The VoDaaS is developed using Java, ASP.NET, C# and XAML.</td>
</tr>
<tr>
<td>5</td>
<td>Security</td>
<td>Access Control</td>
<td>The identity service Access Control is provided based on authentication and authorization.</td>
</tr>
<tr>
<td>6</td>
<td>Video Delivery</td>
<td>WebRole, Silverlight</td>
<td>The video is streamed to the client using Silverlight player.</td>
</tr>
<tr>
<td>7</td>
<td>Debug mode</td>
<td>Diagnostics</td>
<td>The events are traced and the logged events are retrieved from the Table Storage.</td>
</tr>
<tr>
<td>8</td>
<td>Deployment</td>
<td>Hosting Service, Storage Account</td>
<td>The VoDaaS that has been developed in the Compute Emulator is delivered as a Service using Hosting Service and Storage Account</td>
</tr>
</tbody>
</table>
4.3 IMPLEMENTATION OF THE PROPOSED VoDaaS

In the proposed model, the videos are stored in distributed locations effectively through content delivery network. The VoD system is implemented in Windows Azure public cloud as shown in Figure 4.4. The server in the cloud environment along with the administrator (distributor) has the control and can manage the videos stored in the “Storage as a Service” layer. It supports 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 upto 200 GB in the form of block blobs to the cloud storage that can be accessed from anywhere at anytime with provider’s permission. The details of the providers and consumers are stored in the structured cloud table.

Figure 4.4 Cloud Based Architectural Model for Video on Demand
The VoDaaS is deployed in the Azure cloud environment and it is implemented 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 instance as shown in Figure 4.4, which is driven by an User Interface. The user interacts with a webpage or service and the WebRole instance helps in processing the request and response. Thus, the WebRole is the single HTTP endpoint for the external clients. The VoD Service model is characterized 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 “livecloudcinemas” and it will be used to access the storage service via Storage Library and Representational State Transfer (REST) APIs. The service definition of the VoDaaS shown below, 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>
```
The size of the required Virtual Machine is defined as “small”. The size of the compute node enables the individual instance to scale up or scale down dynamically according to the number of requests. The cloud generates an account sharedkey when the VoDaaS is registered with the cloud storage service in the name “cloudmovies”. Only using the account shared key the storage account “cloudmovies” can be accessed.

The service configuration shown below for the LiveCloudCinemasWebRole includes the 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.

```xml
<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+5S0JybmI4gG7tCJ6e8wAl716jtphtCOliUp3KafwiXFhls90BXH+J2r8F5sVQVqw=="/>
            <Setting name="BlobStorageEndpoint"
                value="http://blob.core.windows.net" />
            .... ....
        </ConfigurationSettings>
    </Role>
</ServiceConfiguration>
```

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 number of instances to be created for each WebRole is defined in the service configuration. The load balancer therefore matches the requests across these specified number of 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 have been deployed into multiple Windows Azure role instances, thereby achieving multi-tenant / multi-instance model and thus increasing the reliability.

The Service Definition and the Service Configuration files are used in the setting of the necessary parameters for the successful deploying and running of VoDaaS. The implementation of the Video on Demand model is ensured in a secure and scalable manner. The VoDaaS accesses the cloud table storage to store the details of the customers and videos. The video file is divided into small blocks and is uploaded to the cloud blob storage using the REpresentational State Transfer API. The security of video is guaranteed by not allowing any anonymous clients accessing the video files. The videos are streamed to registered users using Silverlight by the way of progressive download. The VoDaaS is deployed and its performance is measured based on the parameters like upload and download times and throughput.
4.4 CLOUD TABLE STORAGE AS A SERVICE

As the video files will be delivered to the registered users; the details of the booking and the VoD provider and consumer themselves are stored in the cloud table storage. The following Figure 4.5 shows the details of two tables stored in the “cloudmovies” storage.

Figure 4.5 Cloud Table Storage Model for VoDaaS

To store the VoD provider and VoD consumer details in the cloud table storage, the entity class named “ClsConsumer” is inherited from ‘TableServiceEntity’ class and defines all the table entities (columns), such as ConsumerName and ConsumerPwd. 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 the type of client (VoD consumer or provider) is stored in the PartitionKey. Both PartitionKey and RowKey are used to identify a unique client. The definition of ClsConsumer class that encapsulates all the entities of the table such as consumer name and password is shown below:

```csharp
public class clsconsumer : TableServiceEntity {
    private string _strConsumerName;
    private string _strConsumerPwd;
    public string ConsumerName { get; set; }
    public string ConsumerPwd { get; set; }
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
}
```
To insert all the user details into the cloud storage, the class “ClsConsumerDataContext” is written which is shown below. The data context class will upload all the entities that are defined in the entity class to the cloud table storage. The constructor ensures the validation of the Table Endpoint represented as base address and storage credentials, which will be passed by the provider during insertion of a record.

The interface ‘IQueryable’ is used to create table named “Consumers” in cloud table storage, whose columns will be the entities defined in the entity class. The ‘addConsumer()’ method takes a parameter which is the reference to the instance of the entity class. The object is then inserted to the cloud table storage using the ‘addObject()’ and ‘saveChanges()’ methods of the data context.

```csharp
public class ClsConsumerDataContext : TableServiceContext {
    ..... 
    public IQueryable<clsconsumer> Consumers {
        get {
            return this.CreateQuery<ClsConsumer>("Consumers");
        }   
    }   
    public void addConsumer(ClsConsumer objconsumer) {
        this.addObject("Consumers", objconsumer);
        this.saveChanges();
    }   
    ..... 
}
```

From the frontend, when the user clicks on the Submit button “Register User”, the following 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 the system checks if there is required permission to work with the cloud storage account. TheClsConsumerDataContext constructor is triggered by passing the Storage Credentials and TableEndpoint Address. If the client is a new user, the user details are added to the cloud table by calling the addConsumer() method.

```csharp
var Consumer = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
var ConsumerContext = new clsconsumerDataContext(Consumer.TableEndpoint.ToString(),
Consumer.Credentials);
ClsConsumerobjconsumer = new ClsConsumer();

......
objconsumer.PartitionKey = RadioButtonList1.Text;
objconsumer.RowKey = TxtEmailId.Text;
objconsumer.ConsumerPwd = pwdpwd.Text;

......
ConsumerContext.addConsumer(objconsumer);
```

Similarly, the details of the user registering with the video are also stored in the cloud table storage and it is counterchecked when the user is requesting to view the video contents.

4.5 **BLOB STORAGE USING REST API**

The videos are stored in the form of blobs that are divided into block blobs of varying size and stored in a container like storing a file in a
local system folder. Figure 4.6 shows how the block blobs are clustered based on the category and stored in a container.

![Blob Storage Model for VoDaaS](image)

**Figure 4.6 Blob Storage Model for VoDaaS**

The REST protocol is used to upload the video files to the blob storage, as it emphasizes 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, etc.). 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 following code gives the procedure to upload the video file (denoted as filename) by the user and the way to convert the video data 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+5S0JybmI4gG7tCJ6e8wAl716jitptCOliUp3KafwiXFhls90BXH+J2r8F5sVQVqw==";
......
for (int i = 0; i < numBlocks; ++i){
    String block1 = "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 - streamsetPosition);
......
```
After dividing the video files into required number of blocks, each block blobs have to be uploaded to the cloud table storage, which is performed through HTTP Request that is built using the REST API. In the code below, the URI includes the URL of the blob storage along with the container name and filename in which the blob has to get uploaded. 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 block blob is to be uploaded and the Id is given as second parameter namely ‘blockId’.

The following code shows details of the header and the body of the HTTP PUT request that is built using REST API to upload a block blob to the blob storage.

```csharp
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;
```

The created HTTP Request contains the raw byte array of the video that is to be uploaded to the private container named “tamilmovies”. The authentication details such as account name and account shared key are added to the HTTP Request. The HTTP Request that contains the actual streams of
video data and the storage credentials is then executed, as given in the following code.

```csharp
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);
......
```

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.

After executing the HTTP request, the corresponding response is obtained. When the response is successful (200 OK), then the uploading of the video block blob to the cloud blob storage is successful. But, the uploaded block blob data will be committed only after executing the HTTP PUT request with the Blocklist argument. The content of this request contains blockIds of all the block blobs in an XML format.

The XML is created using XMLTextWriter, whose head element is defined as “BlockList” and the child element as “Block”. The value of the `<Block>` element would be the Id of each block blob, which is represented as
“blockId”. After executing the HTTP Request, the corresponding response obtained is captured and checked for the status code. If the response status code is 200 OK, then all the block blobs that are uploaded using HTTP PUT Request are committed successfully. The following code shows the creation of the XML document, execution of the HTTP Request and the verification of the response obtained.

```csharp
using (Stream requestStream1 = request1.GetRequestStream())
{
    using (XmlTextWriter xw1 = new XmlTextWriter(requestStream1, Encoding.UTF8))
    {
        xw1.WriteStartDocument();
        xw1.WriteStartElement("BlockList");

        for (int i = 0; i < blockIds.Length; i++)
        xw1.WriteElementString("Block", blockIds[i]);

        xw1.WriteEndElement();
        xw1.WriteEndDocument();
        xw1.Flush();

        using (HttpWebResponse response1 =
            (HttpWebResponse) request1.GetResponse())
        {
                ....
        }
    }
}
```

Using XmlTextWriter, an XML document is attached to the HTTP Request that contains all the block Ids to be committed. The HTTP Request patterns for both PUT BLOCK and PUT BLOCKLIST methods are given
below. These requests are created when uploading a video file of size 64MB
to the blob storage service. The following is the sample format when the
HTTP PUT BLOCK Request is captured when a block blob of 1MB size from
a 64MB video file is uploaded to the cloud storage.

**HTTP PUT REQUEST - BLOCK**

```plaintext
PUT
http://cloudmovies.blob.core.windows.net/tamilmovies/rithu.mp4?
comp=block&blockId=TGF2aSBDbG91ZE1vdmllczA= HTTP/1.1
x-ms-date: Sat, 08 May 2010 02:44:03 GMT
Authorization: SharedKey
cloudmovies:A3ZGR2hGlbsiDHI6VxaaB6cGTuOg0hbuJGb8Cbm
ka38=
Host: cloudmovies.blob.core.windows.net
Content-Length: 1048576
Expect: 100-continue
Connection: Keep-Alive
```

When a video file of 64MB size is uploaded to the cloud storage,
the above HTTP PUT Request would be executed for 64 times as the size of
each block blob is 1MB. The following HTTP Request for PUT BLOCKLIST
transports the Block Ids of all 64 block blobs in order to successfully commit
a single 64MB video file.

**HTTP PUT REQUEST – BLOCKLIST**

```plaintext
PUT
http://cloudmovies.blob.core.windows.net/tamilmovies/rithu.mp4?
comp=blocklist HTTP/1.1
x-ms-date: Sat, 08 May 2010 03:05:46 GMT
Content-Type: text/xml; charset=UTF-8
```
Authorization: SharedKey
cloudmovies:EBGzSPgdcSHFBZakKk48o7gcYFZHRfRvMjvHqoWIDmI=
Host: cloudmovies.blob.core.windows.net
Content-Length: 1559
Expect: 100-continue
<BlockList><Block>Q2xvdWRtb3ZpZXMw</Block>
<Block>Q2xvdWRtb3ZpZXMx</Block><Block>Q2xvdWRtb3ZpZXMy</Block><Block>Q2xvdWRtb3ZpZXMz</Block><Block>Q2xvdWRtb3ZpZXM0</Block><Block>Q2xvdWRtb3ZpZXM1</Block>
.... 58 such blocks
</BlockList>

The response obtained for the above HTTP PUT BLOCKLIST Request is shown below:

HTTP RESPONSE - BLOCKLIST

HTTP/1.1 201 Created
Transfer-Encoding: chunked
Content-MD5: 5iuBQN7doG4fWe5vXAWLng==
Last-Modified: Sat, 08 May 2010 03:06:09 GMT
ETag: 0x8CCBC728AFEEE05
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: b69c08b0-a39c-4fd1-b330-6bda8e1a8e6
Date: Sat, 08 May 2010 03:06:08 GMT

From the above output, the HTTP status code 201 indicates that the HTTP PUT Blocklist Request is successful. Hence all 64 blocks of blobs each of 1MB size are committed, thereby uploading the complete video file in the
Storage as a Service layer. The response contains the list of resource characteristics like Server details, Content-MD5 and request-id. It also contains the ETag header field which indicates the current value of the entity tag that is created for the given request. The ETag is used to provide cache validation.

4.6 VIDEO SECURITY

When the video is played in the client browser, there is a high possibility that the video will be pirated due to cache. Even when “Video on Demand as a Service” is offered to larger audience, the caching is a problem. Hence, 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 for a specified time duration that the provider permits. A provider can upload, read and delete a video data depending on his permission scope. Access control to the user is given based on the scope of the roles. Each scope inherits roles, permissions, and business rules from their parent. Figure 4.7 illustrates a sample access control provided to the clients.

Figure 4.7 Authorization – Access Control
The signed access signature method is used to provide the read permission to the video files for those who have registered for the video in a stipulated time. In this method, only after specifying the private key of the service provider, the public key will be created, which is used by the client to view the video contents for the stipulated time. The following URL depicts clearly that, only read permission is given for the video blob named ‘rithu.mp4’ to the user, which is specified as ‘sr=b and si = readaccess’

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

The unique string-to-sign denoted as ‘sig’ is constructed from the fields like starting and ending time of the online classes, signed identifier, account name and private key that are verified in order to authenticate the request. The signature obtained is an HMAC (Hash-based Message Authentication Code) computed over the string-to-sign and a public key is obtained by using the SHA256 algorithm (Secure Hash Algorithm), which is encoded using Base64 encoding scheme. The following code shows partly a few attributes for setting the permissions for the video files.

```javascript
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)
        });
    .......
```
For each and every blob’s URL in the cloud blob storage, a security key is generated and attached. This URL will be provided to all the clients and only those clients who have the read permission (registered users) will be able to view the video files.

4.7 VIDEO FETCHING

The Silverlight application, which was initially released as a video streaming plug-in application, 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 after the end time of the video, 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 the same 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 \( M_{ST} \) and \( M_{ET} \) respectively. Let \( T_c \)
be the time when the client logs in and $T_T$ be the total duration of the movie in seconds.

$$\mathbf{MP}_{\text{pos}} = (M_{ET} - M_{ST}) \times \left( \frac{T_c - M_{ST}}{T_T} \right) \quad (4.1)$$

The following protocol segment shows the sample HTTP GET request submitted to the server to access the video file.

**GET REQUEST**

```
GET
http://cloudmovies.blob.core.windows.net/tamilmovies/rithu.mp4?sr=b&si=readaccess&sig=f3M37MRO%2BuWs4d2%2FLaCDGLfPYP%2BAxjsfeUgSm2ihAmQ%3D HTTP/1.1
Accept: */*
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; GTB6.4; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.21022; InfoPath.2; .NET CLR 3.5.30729; .NET CLR 3.0.30618)
Host: cloudmovies.blob.core.windows.net
Connection: Keep-Alive
```

The HTTP response from the server for the above request is given below. The response 200 OK indicates that the video file which is requested can be viewed successfully. Only when the client accesses video content at the given time, the specified video file will be unlocked.

**RESPONSE**

```
HTTP/1.1 200 OK
Content-Length: 67582743
Content-Type: text/xml; charset=UTF-8
```
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 end user is on a slow Internet connection, then it is 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 that requested it, thereby improving performance, scalability, and cost efficiency, to end users. Currently Azure Cloud Environment provides data centres in 18 different locations (Calder 2009). The CDN has been enabled for affinity group ‘Anywhere Asia’, therefore, it caches the video blobs at strategically placed locations to provide maximum bandwidth for delivering the content to users.

4.8 CLOUD DEPLOYMENT

The Video on Demand as a Service utilizes both Storage and Hosted services from the Azure cloud environment. Storage Service is to store the video contents and customer details, which are stored in the form of
tables. The hosted service is created to upload all the files in the WebRole including configuration file. When creating a hosted service, details like hosted service identifier, public service name, description and affinity group have to be specified.

The public service name created for VoDaaS is “cloudcinemas”, thus the service can be accessed using the URL, “http://cloudcinemas.cloudapp.net/” and the affinity group is specified as “Anywhere Asia” that allows to specify where the VoDaaS will be accessed. This helps in faster execution as the services are available within the same data centre where the clients reside most. Thus, there will be fewer hops in the network for communication between the services, thereby speeding up the request and the response time.

4.9 PERFORMANCE ANALYSIS

The performance measures of the proposed Video on Demand as a Service (VoDaaS) are based on the following criteria:

- The upload time and download time of the video files of varying sizes are divided into block blobs.
- The average upload and download time for a chunk of different block blob size of different video files.
- The throughput for both uploading and downloading of video files of varying sizes to the cloud storage.

The following profile settings are considered for testing the performance of the proposed VoDaaS model. A blob is uploaded / downloaded in small chunks of block blob size as specified in the “UnitSizeInBytes” parameter. In case of any failure in upload / download of block blob, the number of times to repeat the task is specified in MaxRetries. The values
defined for the profile setting, when uploading and downloading a video file of size 15MB that is divided into block blobs of 32KB are shown below:

DataMeasurementUnit : Megabytes
BlobSizeInBytes : 15728640
BlobType : Block
ContainerPrefix : entertainment
MaxRetries : 5
NumberOfBlobs : 1
NumberOfThreads : 1
UnitSizeInBytes : 32768

On successful test using for the above profile setting, the various performance measures obtained are given in Table 4.2. Each chunk upload / download is named as Work Item. The throughput is estimated as the total size of the video data transferred divided by the duration of the test run.

Table 4.2 Performance measures obtained for a 15MB video file

<table>
<thead>
<tr>
<th>Performance Measures</th>
<th>Upload</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Work Item Duration In Milliseconds</td>
<td>1258.795978</td>
<td>1183.625822</td>
</tr>
<tr>
<td>End Time (UTC)</td>
<td>8:13:15.3</td>
<td>8:22:44.1</td>
</tr>
<tr>
<td>First Work Item Duration In Milliseconds</td>
<td>1121.022</td>
<td>633.7485</td>
</tr>
<tr>
<td>Max Work Item Duration In Milliseconds</td>
<td>2465.6625</td>
<td>3283.9695</td>
</tr>
<tr>
<td>Min Work Item Duration In Milliseconds</td>
<td>708.939</td>
<td>238.266</td>
</tr>
<tr>
<td>Start Time (UTC)</td>
<td>8:03:11.0</td>
<td>8:13:16.0</td>
</tr>
<tr>
<td>Throughput (Megabytes/sec)</td>
<td>0.024823545</td>
<td>0.026400927</td>
</tr>
<tr>
<td>Total Data Transferred In Bytes</td>
<td>15728640</td>
<td>15728640</td>
</tr>
<tr>
<td>Total Items Transferred</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total Retries</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The different video files of various sizes starting from 1MB to 75MB are uploaded to cloud storage by dividing it into block blobs of size 32KB, 64KB and 1MB. The time taken to upload the video files of varying sizes to Windows Azure cloud storage is shown in Figure 4.8.

![Figure 4.8 Upload time for storing the blob data to cloud storage](image)

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 32KB is higher when compared to other uploads in the given scenario.

It is mandatory to divide the video files that are above 64MB 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 size of a particular block blob as higher as it can be accommodated.
The time taken in seconds to download video files of varying sizes from the cloud storage is shown in Figure 4.9.

![Figure 4.9 Download time for fetching the blob data from cloud storage](image)

Similar to the upload time, the download time of the blob data increases as the size of video file increases and in most 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 32KB is higher when compared to other downloads in the given scenario. Downloading time is marginally lesser when compared to the uploading time of the video file. The percentage difference between the upload and download time for the stored video files of varying sizes are given in Table 4.3.

**Table 4.3 Percentage difference between upload and download time**

<table>
<thead>
<tr>
<th>Video are divided into Block blobs (size)</th>
<th>Percentage increase in upload time</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 KB</td>
<td>5.155</td>
</tr>
<tr>
<td>64 KB</td>
<td>4.780</td>
</tr>
<tr>
<td>1 MB</td>
<td>3.529</td>
</tr>
</tbody>
</table>
The factor of variation for upload and download time between block blob sizes of 32KB and 1MB is calculated to understand the association between them. The factor of variations for upload time ($F_{vu}$) and download time ($F_{vd}$) are estimated as follows:

$$F_{vu} = \frac{(U_{T32} - U_{T1})}{U_{T1}}$$  \hspace{1cm} (4.2)

$$F_{vd} = \frac{(D_{T32} - D_{T1})}{D_{T1}}$$  \hspace{1cm} (4.3)

where $U_{T32}$ and $D_{T32}$ are the average upload and download time of various video files that are split into block blobs of 32KB, and $U_{T1}$ and $D_{T1}$ are the average upload and download time of various video files that are split into block blobs of 1MB.

The following Figure 4.10 shows the factor of variation for various video files of different sizes versus the time in milliseconds.

![Figure 4.10 Factor of variations between 32KB and 1MB Block blobs for both upload and download time](image-url)

It is found that with respect to factor of variation, the download or the upload time for the video files that are divided into block blobs of 32 KB or 1MB, there is a minimal variation and the variation reduces as the video
file size increases. Therefore, when the size of the video file is higher, the upload and the download time does not differ much based on the size of the block blobs. The average time taken to upload and download a single chunk of 32KB and 64 KB block blobs respectively are shown in Figure 4.11 and for a chunk of 1MB size is given in Figure 4.12.

Figure 4.11 Average upload and download time for 32KB and 64KB block blobs

Figure 4.12 Average upload and download time for 1MB block blob
The average time taken for uploading a chunk of 32 KB and 64 KB data are below 4000 milliseconds and to upload a chunk of 1 MB to cloud storage is approximately ten times higher than uploading a 32 KB and 64 KB block blob. Even though, the upload time of the full video file that are divided into 1MB of block blobs is less, the average time to upload a chunk of block blobs 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. For example, to upload a 1MB file, which is divided into block blobs of 32KB each, on an average it takes fifty eight seconds and if the same 1MB file is uploaded as such, on an average it takes twenty nine seconds, which is approximately fifty percent lesser in time.

The upload and download throughputs are estimated while uploading and downloading the video files of different sizes from Windows Azure cloud storage are shown in Figures 4.13 and 4.14 respectively.

![Figure 4.13 Upload Throughput](image-url)
Many real time applications, especially Video on Demand needs a minimum rate of throughput to provide a satisfactory result to VoD clients. Therefore, the clients can obtain the minimum continuous traffic rate that the network devices can handle without dropping any single packet.

### 4.10 COST ANALYSIS

Estimating the cost of infrastructure is a key step towards a quantitative analysis of issues such as ROI (Return on Investment) and SLAs (Service Level Agreements). The organizations, which minimize the total operational cost and maximize the innovative investments, have higher revenue and profitability performance. Through the proposed VoDaaS model, the VoD providers can reduce the cost associated with hardware and maintenance and thereby increasing their profit. The distributors or the video providers need not fear about the computing resources, storage capacity, compatibility and about the video piracy issues. The clients can watch videos
not only in the theatres but also in the home theatres and hence this approach becomes a secondary way of watching videos.

The comparison of the computing capacity and cost versus time between the traditional hosted Video on Demand model and the proposed ‘VoDaaS’ model is carried out. In the traditional hosted ‘VoD’ model, the required hardware is installed in advance by forecasting the number of requests, the workload and the growth.

These service providers ensure that they invest on the hardware 40 to 60 percentage more than the forecasted need. As the load increases, the capacity is also added ensuring over provisioning. The virtualization helps to decrease the over provisioning, but it has to be done manually by adding the required virtual machines or the physical server. Even by doing so, 20% of the capacity is over-provisioned, which not only increases the initial cost but also increases the maintenance cost. The comparison is considered including the inbound and outbound bandwidth, storage space, VM instances and other miscellaneous expenses such as maintenance, setup and delivery charges.

The profile settings considered for the comparison of traditional Video on Demand model with the proposed Video on Demand as a Service is with respect to the Total Cost of Ownership (TCO) are as follows:

- **Application type:** Web Applications - Stored media streaming distribution
- **Development of the service:** New Services
- **Hardware Resources:** Small (< 4 servers / VMs)
- **Integration:** Not integrated with other applications, either on-premises or in the cloud
- **Request - Traffic:** Heavy user logins
- **Profile of the service over time:** Unpredictable Spikes
  - Application is used over extended period of time with sudden spike(s) of capacity (e.g. sudden spike due to instant popularity)
- **Average use hours per day:** 24 hours
- **Average use days per year:** 365 days
- **Number of Virtual Machines per Server:** Three

For the comparison, the cost for initial on-premises infrastructure is included. The on-premises infrastructure is configured with virtualization. The Azure TCO calculator is used for cost analysis of both traditional VoD and proposed VoDaaS and the result shows that the Return of Investment for the proposed VoDaaS is 60 percentage. The graphical representation of the above results is depicted in the Figure 4.15.

![Figure 4.15 Cost analysis between traditional VoD system and VoDaaS](image)

**Figure 4.15 Cost analysis between traditional VoD system and VoDaaS**

In spite of these high investments, there are chances of unpredictable high growth in the load, which at times is very difficult to handle immediately, thereby causing poor performance in the server. There are also chances that the server might go down. This will surely reduce the client’s trust on the service provider. The other disadvantage with the
traditional model is that most of the time, the computing and the hardware capacities are not fully utilized, leading to excessive and needless investments.

The computing capacity and cost incurred over time in the traditional Video on Demand model is shown in Figure 4.16. Here, the dotted line (in blue color) refers to the capacity forecasted, the dashed line (in green color) denotes the established, over provisioned capacity and the line in the red color denotes the actual computing capacity that has occurred.

![Figure 4.16 Cost versus Time in traditional VoD](image)

In the proposed ‘VoDaaS’ model, the concept of over provisioning and under provisioning is drastically reduced by providing or reducing the capacity immediately when required. This model reduces the capacity when the load reduces and dynamically increases to the required capacity when the load increases drastically. As the hardware utilization is fully automated, the idle resources can be used efficiently.

The computing capacity and costs incurred over time for the proposed Video on Demand as a Service is shown in Figure 4.17. Here, the
blue dotted line refers to the capacity forecasted. The dashed line (in green color) denotes the computing capacity utilized by the proposed VODaaS, which is always excess to the actual computing capacity used, which is denoted by the line in red color.

![Figure 4.17 Cost versus Time in proposed VoDaaS](image)

**Figure 4.17 Cost versus Time in proposed VoDaaS**

Comparing the computing capacity and cost versus time between the traditional Video on Demand service and the proposed model, the net savings with regard to the cloud environment for a particular case is approximately 20 to 40 percentage of hardware and maintenance cost during the period of 36 months. The other advantages of providing ‘Video on Demand as a Service’ model in cloud environment is that, the provider need not worry about the storage requirement, capacity planning and security management.

### 4.11 CONCLUSION

An enhanced and secured cloud computing architectural model for Video on Demand has been developed. This model ensures cost-effectiveness and scalability of the service. The VoDaaS provides a smooth and lively
streaming of movies to the clients thereby trying to reduce the video piracy issues. The Content Delivery Network (CDN) is enabled to provide the lowest latency and best connection speed for the requester. When a client requests for a particular video, the video blob data is read directly from the Cloud Storage. Instead, when a CDN is configured and a request is made using the CDN, the request is redirected to the CDN endpoint closest to the location from which the request has been made to provide access to the video blob.

The developed “Video on Demand as a Service” in Cloud Environment offers scale up and scale down environment which is best suited for a scenario that does not know the number of clients accessing the video, thereby ensuring dynamic scalability. In addition to it, this model is hosted on top of the IaaS, which provides additional scalability and secure environment. To fetch the videos faster they are stored as blobs that are divided into multiple blocks of 1MB each. To avoid piracy, the videos are encrypted during runtime using the private key. As the application is scalable, it provides benefit to the distributor in terms of faster money back. Further, this model can also be utilized not only for entertainment purpose but also in the field of education.