CLOUDSIM BASED SIMULATOR FOR DISTRIBUTED VIDEO TRANSCODING

Habtegebreil Kassaye Haile
ABSTRACT

This report provides details of a new implementation of the simulator used for the paper [1]. The original simulator was implemented on SimPy discrete event simulation tool. Driven by some shortcomings in the SimPy tool and the underlying python programming language, a new simulator on a framework better suited for modeling and simulation of cloud services was required. The aim of this project is to provide that improved simulator. The algorithm implemented provides way to allocate and deallocate virtual machines dynamically on a collection of transcoding servers.

Keywords: CloudSim, Video Transcoding, Distributed Transcoding, Dynamic Provisioning
CONTENTS

Abstract i

Contents ii

List of Figures iii

Glossary iv

1 Introduction 1

2 The Simulated System 3

3 CloudSim 6
  3.1 CloudSim Architecture ................................. 6
  3.2 CloudSim Core ........................................ 8

4 Components of the Simulator 10
  4.1 TranscodingMain ....................................... 10
  4.2 TranscodingBroker ................................... 15
  4.3 TranscodingProvisioner .............................. 17
  4.4 Segment ................................................ 20
  4.5 Stream ............................................... 20
  4.6 TranscodingVm ...................................... 20

5 Outputs and Performance of the simulation framework 21
  5.1 Transcoding rates, Play rates and Number of VMs .......... 21
  5.2 Simulation Time performances .......................... 24

6 Future work and conclusions 26
  6.1 Conclusions .......................................... 26
  6.2 Future work .......................................... 26

Bibliography 27
## List of Figures

2.1 Simulated system architecture .................................................. 4
3.1 CloudSim layered architecture .................................................. 7
3.2 CloudSim Core ........................................................................ 8
4.1 Class diagram of the simulator .................................................... 11
5.1 Plot Total transcoding rate and total play rate (Top) and Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (Bottom), with allocation/deallocation performed every 10 sec .................. 22
5.2 Plot Total transcoding rate and total play rate (top), Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (bottom), with allocation/deallocation performed every 20 sec .................. 23
5.3 Plot Total transcoding rate and total play rate (top), Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (top), job length = 10 secs ................................................................. 25
GLOSSARY

DES  Discrete-event simulation

Vm  Virtual Machine. a service running on the servers and hosting the transcoding requests.

MIPs
   Millions of Instructions per second. A measure in cloudsim used to processing power.

RAM
   Random Access Memory.

BW  BandWidth.
1 INTRODUCTION

Cloud computing has emerged to become an important means of providing computing services to customers. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are the broad group of services provided in a cloud. The emergence of such clouds has relieved individuals and companies from undertaking, setting up and maintenance of the hardware and software platforms they need.

With this growth in cloud services there is also a growth in the need for platforms providing precise estimations of how an actual cloud system would perform under certain specifications and demands. At the moment a number of simulation frameworks to test the performances of cloud related applications exist. These platforms are expected to support the variability of cloud patterns, the heterogeneous nature of those demands and to dynamically adapt to those demands. Some of the simulators used to test cloud system behaviors are simple discrete event simulation tools that would require developing a simulator starting from the low levels, with one example being SimPy. Other tools have been developed with cloud and/or distributed computing in mind. An example of such simulators is CloudSim [2].

The first section of this document is a description of the system the simulator aims to reproduce. The system descriptions briefly put the purposes of the system components.

The second section of this project gives a brief introduction on the tool used to develop the simulator. The framework of choice is CloudSim [2]. It starts out with general architectural view of CloudSim with three broad layers. Relevant portions of those layers will also be described. A brief description of the core layer and the associated management of generated events are then given.

The next section will explain the parts of the simulator code that is specifically written for this project. Some of the files are a modification on the provided classes of the CloudSim library by overriding some functions or introducing new data mem-
bers. There are also new classes and methods that provide some of the functionalities required to realize a prediction based dynamic resource allocation of distributed transcoding servers. The source code components will be discussed in decreasing order of content, starting from the main class where the simulation begins.

The fourth section presents the results of tests obtained from running the simulator with different parameters. The first part of this section presents plots showing the dynamics of transcoding rate, play rate and number of VMs for the duration of the simulated time. The second part of this section presents tabulated data of the recorded simulation time for different simulation times and for different mean job sizes during stream segmentation.
2 THE SIMULATED SYSTEM

The simulated system is given in Figure 2.1. The system performs video transcoding service based on a dynamically scalable cluster of servers and it is the same system modeled and tested in the paper [ref paper, but with a different DES tool].

The system tested on this report is the same as the one on the paper [1]. The advantage of the system demonstrated in 2.1 is that segmenting streams into smaller jobs and reassembling them after completion would result in a more efficient and fast transcoding service. This becomes very significant when observing transcoding operations of streams with different transcoding demands. Suppose two streams where one needs a much higher transcoding time than the other arrive. The two streams are then sent to two separate transcoding servers with the same capabilities. In such cases one would finish much later than the other even if they both have the same output play rate. While the segmenting aims to alleviate such problems, the prediction based dynamic allocation enables the required number of transcoding servers to be available for the jobs acquired by segmenting the streams.

Video requests arrive and responses are sent out through the streaming server. The video repository holds a copy of the transcoded video for a certain period of time. The video streams are split into smaller segments in the video splitter. These segments that are referred as jobs in [1] are then placed in a job queue. The load balancer distributes the jobs among the transcoding servers. The load balancer selects the appropriate transcoding server based on the number of jobs waiting in the transcoding servers queue, i.e., one with the shortest queue length is selected. It then sends the first job in the queue of jobs to the selected transcoding server.

The master controller manages the system and determines resource allocation based on the prediction based dynamic resource allocation and deallocation algorithm. The load predictor is responsible for supplying the master controller the prediction data. The cloudprovisioner is responsible for starting and closing VMs upon the request of
Figure 2.1: Simulated system architecture
the master controller. The video merger produces a video stream output from the job outputs belonging to that stream.
A number of frameworks were considered to be used as a tool to develop the simulator. As the streams will be arriving at certain moments in time to be processed by the simulator, the frameworks that were considered were all discrete event simulation frameworks. In addition to stream arrivals a number of stream and stream segment parameters will also trigger certain actions to be taken.

The framework of choice here is CloudSim. CloudSim is a discrete event simulator with features and functionalities that enable easy modeling and simulation of cloud infrastructures and services. In particular, CloudSim offers approaches to define and create cloud entities in ways that are easier to visualize and understand. In this section some of the important components of CloudSim useful to understand the simulator are discussed. It is important to note that not all the components in CloudSim are presented, but only those directly related to the simulator. For detailed understanding of the Cloudsim framework refer to [2] and the frameworks API.

3 CloudSim

In Figure 3.1, it shows the components of CloudSim grouped in different layers. The user code layers are where the user/coder defines simulation specifications, like the hosts and their characteristics, the applications and their requirements and VMs. In general the layers in the CloudSim group label support the modeling and simulation of the virtual datacenters. The discussion for the core layer is given in the next subsection.

Through out this report an entity is any instance of a component and a simulation entity will refer to an instance of a derived class/component of the CloudSim component ‘SimEntity’. A host is a component that represents a real world computing device. Multiple processing elements (PEs) can be associated to it to represent a multicore device. A number of hosts can be managed by a data center entity. A number of VMs
can be associated with a single host. How multiple VMs operate on a host is governed by the VM allocation/provisioning policy component. Custom methods can be applied to this component to allow implementation of new policies. By default, a simple first come first serve VM allocation policy is provided.

In CloudSim applications are modeled by extending the Cloudlet class. This class models a basic application services and can be deployed on instances of VM class. A service allocation policy will take care of assigning VMs to the defined application services. A host allocation policy is used to determine the distribution of VMs among multiple cores of a host instance. The host allocation policies provided in CloudSim are space shared, where a CPU is exclusively mapped to a VM, and time shared, where a core is dynamically shared among multiple VMs with on demand assignment of cores to VMs.

The DataCenter class models the hardware hosting the cloud service providers infrastructure. It can hold a diverse set of host machines on which the VMs will be run. It contains instance of a provisioning component that implements policies for resources to hosts and VMs.
3.2 CloudSim Core

CloudSim originally relied on GridSim that in turn relies on SimJava library for event handling and communication between entities. However due to scalability issues, a new discrete event management framework was developed [2]. Figure 3.2 shows the class diagram for the core of the simulation framework [2].

The CloudSim class is the main class controlling the event queues and execution of these events. Events are represented as instances of the class SimEvent. Some of the information stored in event instances are time of occurrence, time of delivery to the destination simulation entity, finish time, source and destination id, tag of event and data for destination entity. Generated events are ordered based on their time parameters and then added to the future queue. When the scheduled time of events arrive the events are removed from the future events queue and moved to the deferred events queue. All Simulation entities, which are instances of classes derived from the class SimEntity, will then run their event handling procedures. These procedures will perform the appropriate action based on the events found in the deferred queue.

The class SimEntity is an abstract class. Instances of derived classes of SimEntity are able to pass messages between each other and issue and handle events. Classes that extend this function must implement three functions: startEntity(), shutdownEntity() and processEvent(). The startEntity() and shutdownEntity() are procedures that run when a simulation entity is initialized and destroyed. The processEvent() is the procedure that handles events in a simulation entity.

CloudSimTags contains list of tags for events. The tags are used to identify the kind of action to be taken. It is also possible to use other tag definitions in events if there is a need for a new custom event and a new action corresponding to that event. Another
important entity is CloudInformationService (CIS), which is responsible for registration of resources. The CloudSimShutDown is the last entity that runs and indicates the simulation has come to its end.

In a nutshell, upon start of simulation, datacenter entities are created and registered in the CIS. Then brokers will request about the available resources from the CIS. After getting the resource characteristics the brokers will then create VMs and associate application/jobs/cloudlets to the VMs.
4 Components of the Simulator

As described before, the simulator is built upon the CloudSim framework and uses various predefined classes from the framework. In addition some already provided classes of the simulator were redefined and adjusted to suit the need of the simulator. New classes containing required functions have also been defined and integrated. The following are a list of java classes that have been either redefined or written from the scratch. Figure 4.1 shows the component classes of the simulator and the relationship between the classes.

4.1 TranscodingMain

This is the place for the highest layer controls and where the simulation will be started. Creation, initialization and coordination between the different entities involved in the simulation is done here. The above is accomplished through functions in the classes and also methods of external classes. The TranscodingMain class contains the main function and also a static class derived from CloudSim SimEntity to handle some high level events. The functions of the main class are discusses one by one below;

createVM(…) – this function is responsible to instantiate VMs the simulation and returning a list of the instantiated VMs. In order to create a standard CloudSim VM a number of parameters need to be specified. Those parameters specify characteristics like image size of the VM, how much RAM the VM should have, execution speed of the VM in MIPS and scheduling method for the cloudlets/tasks to be executed on the VM. In addition to these typical members of the VM, a new member specifying the renting time of the VM is included. The createVm(…) function takes three arguments. The first argument specifies the Id of the entity that will be managing the VMs. In this case the VMs are managed by a simulation entity called 'Broker', which is responsible for allocating jobs to VMs and destroying the VMs when they are no longer needed,
Figure 4.1: Class diagram of the simulator
among other things. The second argument specifies the number of VMs to be created and the third argument is the starting point for assigning Id numbers for the newly created VMs. All the VMs created in the simulation are identical.

createCloudlet(...) – The jobs to be executed on the VMs are instantiated in this function. As arguments it takes the Id of 'Broker' entity, the creation time for the Jobs/cloudlets, the starting point for assigning Ids, the stream to which the created jobs belong to and an array containing the size of jobs in terms of execution time of each job to be created. The function returns a list containing job instances (referred also as cloudlets). To create the basic Cloudlet instance through the provided CloudSim libraries one needs to specify the job length in terms of millions of instructions, size in memory of the input and output of the job, utilization models for the CPU, RAM and BW(bandwidth) and the execution speed of VMs in terms of MIPs. In the simulator, simple utilization models where the cloudlet uses full capacity of the resource are used. The job length is acquired from the function argument that contains a list of job length. Subsequently, the input and output file sizes, which are expressed in terms of number of video frames, are calculated from the job length and stream characteristics in the way given below;

\[
\text{filesize} = \frac{\text{length} \times \text{stream.filesize}}{\text{stream.transcodingtime}} \quad (4.1)
\]

\[
\text{outputSize} = \frac{\text{filesize} \times \text{stream.outputfps}}{\text{stream.inputfps}} \quad (4.2)
\]

The filesize represents the input file size. It is obtained by multiplying the total input size of the stream by the ratio of the job length (job transcoding time) with the total transcoding time of the stream. The output filesize is determined by multiplying the play duration of the portion of the stream belonging to the particular job with the output frame rate of the stream. Since the play duration is the same for the stream before and after transcoding, it is obtained by dividing the input number of frames by the input frame rate.

In addition to the above values associated with the default definition of cloudlets, values specifying the stream to which the job belongs to, the input and output frame rate of the stream and the starting time (although not used in the simulation) of the cloudlets are also specified.

createStreams(...) – This function handles the creation of streams based on data
acquired from files specifying the number of streams to create and characteristics of those streams. As an argument this function takes a buffer reader object, associated with a file used to determine the number of streams to create, and the total stream count until that instant. The function returns a list of stream objects. Once the function extracted the number of streams to be created from the associated file, it will then go on to randomly select streams. After a stream is selected from the file containing the list of video streams and their associated characteristics, some characteristics are extracted and used to create a stream object. Those selected characteristics are transcoding time of the stream, number of I frames in the stream, input frame rate of the stream, output frame rate and file size in terms of number of video frames. The transcoding time of the stream is expressed in terms of MI by multiplying the duration in time by the MIPs value of all the VMs, which is universally 1000 in this case.

splitStream(...) – Once the streams are created the splitter function takes the responsibility of dividing the stream to jobs based on a Gaussian distribution with a given mean transcoding time and standard deviation. The choice of the mean transcoding time for the jobs is important in determining how many jobs will be created. This function is called for each stream that require splitting and it returns a list that contains the job sizes for the stream. The list of job sizes will be later used in the createCloudlet(...) function.

createDatacenter(...) – Creates the DataCenter class instance that represents a real life data center. The datacenter object will contain a list of host object. Characteristics such as PEs(processing elements) with specific MIPs and VM scheduling policy for PEs, Id, RAM size and RAM provisioning policy, storage capacity and Bandwidth capacity and Bandwidth provisioning policy are specified during instantiation. All provisioning and provisioning policies used are rudimentary ones where the resource in question is fully/exclusively used by entity being serviced and freed once the entity completes its need. The MIPs and RAM values of the host are set in such a way that a host can only support one VM at a time. A very high number of hosts are created so that hosts will always be available to support any number of VMs that might exist in the simulation.

When creating the datacenter object a datacentercharacterstics object, which specifies the various costs associated with the datacenter and other characteristics like name, operating system and time zone, is specified. In addition an allocation policy for the host list in the datacenter is also specified. The allocation policy for the host list
is a simple one that chooses the host with the lowest number of PEs in use. Since there
is only one PE created per host, declaration of this allocation policy is just a formality
to concur with the predefined syntax.

printCloudletList() – this function is used to print out the list of completed job-
s/cloudlets after the simulation is completed. This is taken from the example codes
given for CloudSim with some minor modifications introduced to display information
regarding new data added in the simulator. However this function will only be used
for short term simulations where the amount of jobs is limited. This is because the
function makes use of a list that accumulates completed cloudlets, but this list will
be consuming memory for longer simulations and will not be allowed to accumulate
objects for such simulations.

Main() - the main function does the task of initializing the simulator with the ap-
propriate flags and parameters and create all required the simulation entities. One of
the simulation entities created is an object identified by the object globalBroker cre-
ated from the class GlobalBroker. An instance of DataCenter class is also created here.
Then the simulation is started and at the end of the simulation the end time is recorded
to determine how long the simulation took to complete.

The highest level component handling high level events in the simulation and
performing some coordination between other event processing entities is the Glob-
alBroker. It is an extension of the SimEntity class. As described before this means
that GlobalBroker has to reimplement the three functions: startEntity(), processEvent
(SimEvent) and shutdownEntity().

The startEntity() is called when the simulation is started, therefore any initialization
or initial scheduling of events will be placed in this function. In this particular case
the startEntity() serves a simple purpose of opening the stream arrival rate file and
scheduling the initial tasks of creating jobs and VMs. The shutdownEntity() closes all
the file readers that has been opened in the simulation.

The function that is responsible for calling out the appropriate procedures when an
event is to be processed by the GlobalBroker is processEvent (SimEvent). When an
event is scheduled a tag used to identify the event is associated with it. Based on the
tag associated with the arriving event the function calls the corresponding procedures.
There are two events handled by the GlobalBroker class. Those are events that call
for creation of new jobs and calculating the amount of VM provisioning (allocation of
deallocation). When the creation of jobs event arrives then the function processNew-
Jobs() is called, and when VM provisioning event arrives the function processProvisioning() is called.

The processNewJobs() function is responsible for calling the functions that create streams, split the stream and create the actual cloudlets/jobs. If the function is being called for the first time it will initialize all the required variables, cloudlets and VMs. It will also set an initial number of VMs and submit them to the next layer for creation on the datacenter and create a file reader stream associated with the file containing the arrival times. The function will then proceed by calling the function for the creation of streams. Then for each stream it will create an array of job sizes by calling the stream-Split() function and create the jobs for the stream based on the job size array acquired. After that it will update the job/cloudlet count and submit the list of cloudlet to the next layer broker to schedule their execution. At the end the function will schedule the next job creation time by calling the scheduling function with the appropriate tag.

---activity diagram

The processProvisioning() function is called when an event for adjusting the number of VMs occurs. The number of additional VMs to be created is obtained by calling the method of the instance of TranscodingProvisioner class. If more VMs need to be created, the method will return the number of VMs to be created. But if instead the number of VMs need to be reduced the function will return zero. The detailed processing and implementation of the TranscodingProvisioner class is discussed in a separate subsection. If more VMs need to be created then the appropriate number of VMs will be created and submitted to the next layer, an object of the class TranscodingBroker. The class TranscodingBroker is also described in a separate section below. Finally multiple methods to access the data members of the class are also provided.

### 4.2 TranscodingBroker

This class is an extension of the TranscodingBroker class provided in the CloudSim library examples. The basic TranscodingBroker is an extension of the SimEntity class. The TranscodingBroker is responsible for negotiating resources on the cloud providers (DataCenters) to the needs of the demanding users (applications running by VMs). Some of the roles of this class are management of cloudlets and VMs, keeping track of important information used when determining provisioning and distribution of jobs among the existing VMs.
Just like any other class extended from class SimEntity, this class also re-implements the function processEvent(SimEvent). In this function all the events that are to be handled by this class are forwarded to the appropriate function. In the following paragraphs some of the important functions that have been modified from the basic version and other new functions are discussed.

The first important function in this class is the submitCloudlets() function. This function is not called in direct response to an event from the processEvent(SimEvent) function but instead from inside other event handling procedures. The function serves the purpose of allocating the appropriate VM to a cloudlet. The appropriate VM is selected by making use of a function that returns the index of the VM with the lowest number of cloudlets scheduled to be executed on it, i.e. the VM with the smallest queue is selected.

Once the VM for a cloudlet is selected then the transcoding time is calculated from the length of the cloudlet specified in terms of MIPs and the speed of the VM, which is also in MIPs. Right now this process is more of a formality since the cloudlet length was obtained by multiplying the time value obtained from a file with a universal VM MIPs value, as described in the createCloudlet() function description. The delay between a jobs/segments play time and transcoding time is determined and the segment is held from submission to the Vm until this delay expires. This is meant to reduce accumulation of transcoded jobs at the output buffer resulting from streams with higher transcoding rates but smaller play rate. All the necessary tracking variables and data members are updated to reflect the current state of the simulator. These state data members are responsible for keeping track of the total number of cloudlets in the system, the cloudlet count per VM and the cloudlet count per stream.

The processVmCreate(SimEvent ev) function is responsible for handling a response event from the Datacenter informing the Broker that a VM has been successfully created in the system. The function will map the VM, whose successful creation is confirmed, to the datacenter where the VM is created in. Then the job count of the VM is initialized and the VM is added to a list containing the currently running VMs. Once all the required VMs are created, unassigned cloudlets will be submitted to be assigned to a VM. This function also contains scripts that manage the case when all the requested VMs cannot be created in the available data centers.

When a job is completed the generated event is handled in in this class by the processCloudletReturn (SimEvent ev) function. In general the purpose of this class
is to make all the necessary state changes and updates reflecting that a job is on its way out of the transcoding VM. The principal updates are removing the cloudlet from the list of submitted cloudlets, decreasing the job count in the associated VM to job count map and decreasing the job count from the associated stream to job count map. If the cloudlet happens to be the last job of the stream it belongs to, the stream will be removed from the system. If the job is the last job to be processed on the associated VM, then the VM will be scheduled for deletion at the end of the renting time and all the parts that are keeping track of the state of the VM will be removed. If there are no more cloudlets left in the system then the simulation is ended.

In addition to above functions the class included many functions inherited from the parent class and a simple function that returns the index of best suited VM. The functions discussed above are those that have been overridden from the basic implementation for the purpose of this simulator.

### 4.3 TranscodingProvisioner

The other major class in the system is the MyProvisioner class containing the functions to calculate the number of VMs to allocate or deallocate. The allocation and deallocation is supposed to closely follow the transcoding and play rate demand. The functions of this class will be discussed in this section in the order in which they will be called.

The calculateTotalTR(MyDatacenterBroker broker) function is responsible for calculating the current transcoding rate of the system. A cloudlet can be in one of different states in the system. In CloudSim constant field values are defined to differentiate these states. The cloudlet states that are already defined in the CloudSim library are canceled, created, failed, failed resource unavailable, in execution, paused, queued, ready, resumed and success. Each cloudlet will also have a status variable. During the course of a lifetime of a cloudlet the status will go through some of the above defined states. When computing the current transcoding rate of the system these cloudlets, who are currently being executed in a VM, i.e. status is in execution, are considered. The transcoding rate of all those executing cloudlets is added up to give the total transcoding rate of the system. This value is then returned by the function.

The calculated total transcoding rate is then used as an argument for the function getPredicted(double TR_curr) that calculates a prediction of the transcoding rate. The algorithm is a direct reimplementation of the procedure used in [1], which is in turn
based on [3]. The method employs a two-step approach. The first step consists of a
load tracker, that implements exponential moving average (EMA) over a previously
gathered set of n values. Simply put the EMA computed for the latest transcoding rate
measurement is the weighted sum of the current measurement and EMA obtained from
previous measurement.

\[ EMA(t_i) = \alpha \times s_i + (1 - \alpha) \times EMA(t_{i-1}) \] (4.3)

Where \( \alpha = n + 1 \), \( t_i \) and \( t_{i-1} \) are current and previous time instants and initial
EMA is average of the first n measures. A set of q values obtained from load tracker
at q past instants of time is then used by the second stage (load predictor). The load
predictor outputs a future value after a prediction window h. In this approach the linear
predictor output is calculated using the expression given below;

\[ l = \gamma_0 + \gamma_1 t \] (4.4)

Where \( \gamma_0 \) and \( \gamma_1 \) are can be estimated at run time based on the LT values and are
called regression coefficients.

The next function in this class is the allocateDeallocateResource(...) function. This function is responsible to determine if additional VMs need to be created or some
need to be removed. Then it will determine the number of new VMs to add or the num-
ber of VMs to remove. More VMs will be added if a weighted transcoding rate, which
is calculated from the total calculated transcoding rate and the predicted transcoding
rate, is less than the play rate threshold for allocation. The play rate threshold for
allocation used here is the actual total play rate in the system. The total play rate is
calculated in the same way as the total transcoding rate, adding up play rate of all cur-
rently executing cloudlets. If the above condition holds and new VMs are to be added
the equation below is used to determine the number of VMs to add.

\[ \text{number of vms to add} = \frac{\text{total play rate allocate} - \text{weighted TR}}{\text{total transcoding rate \over \text{vmList.size()}}} \] (4.5)

If the weighted transcoding rate is greater than the play rate threshold for dealloca-
tion then this implies there are more VMs than needed. The deallocaation procedure
is a little more complicated than the allocation one. Instead of simply determining the
number of VMs to remove, the procedure also identifies the best suited VMs to be removed and marks them for deletion so they can be removed when their renting time expires. The first step the procedure does is identify the VMs in the 'deletion range' of their lifetime. The deletion range is a time interval within a renting lifetime of a VM, that allows a VM just created and a VM almost about to be expired not to be deleted. The idea behind overlooking almost expiring VMs from deletion is the assumption that renewal process will have already started for a VM very close to expiry. After the VMs eligible for deletion have been identified then we determine how many of these VMs need to be deleted. The number of VMs to be deleted is calculated using the expression given below:

\[
\text{number of vms to remove} = \frac{\text{weighted}_{-}\text{TR} - \text{total}_{-}\text{play}_{-}\text{rate}_{-}\text{deallocate}}{\text{total}_{-}\text{transcoding}_{-}\text{rate}_{-}\text{deallocate}} \frac{1}{\text{vmList}_{-}\text{size}} \tag{4.6}
\]

The above steps are followed by a procedure that sorts the VMs eligible for deletion on increasing order of remaining renting time. This is accomplished by a quicksort algorithm implemented by the function quicksort(\ldots). The quicksort(\ldots) function returns an array with the index of VMs in the VM list arranged in the order described. Once the VMs are arranged then the ones to be removed will be selected and a Boolean value indicating they are going to be deleted is set. This Boolean value is checked in the submitCloudlets() function of broker object of TranscodingBroker class to see if more cloudlets could be added to a VM. If this value is true (indicates the VM is close to removal) then no new cloudlets are sent to the VM.

The allocation and deallocation function returns a number value that contains the number of VMs to create. If more VMs are to be created it will return the number of VMs to create else it returns zero.

The final function of the MyProvisioner class is the calculateProvision(\ldots) function called from the GlobalBroker in the main file. This function starts up the provisioning process since all the discussed functions are subsequently called from within this function. It starts out by determining the total play rate, and then the total transcoding rate and the predicted transcoding rate are obtained by calling the corresponding functions. Then the function to determine the number of VMs to allocate and deallocate is called. The values for play rate, transcoding rate, predicted transcoding rate and the number of running VMs are then all recorded on an output file. Finally the function
returns the appropriate value indicating the number of VMs to provision.

4.4 Segment

This is a simple extension of the Cloudlet class given in the CloudSim library. Cloudlet class represents an application service and in the context of [1], this class is equivalent to jobs to be executed on VMs. A job is a segment of a stream requiring a certain transcoding service. In addition to the parent class members this class has additional members to contain important characteristics of the job. Some of those important characteristics are input frame per second (inputfps), the transcoding rate of the stream or job and output frame per second (streamPlayRate). There are also members that hold the identification number of the cloudlet globally and within a stream. Functions to get access to the above data members are also defined.

4.5 Stream

This represents a video stream in the real world. The class holds important characteristics of the stream. Some of those characteristics are the total number of frames in the stream, the transcoding time of the stream, input and output frame per second of the stream, number of I frames in the stream and value used to identify the stream. Functions to get access to some of the above data members are also defined.

4.6 TranscodingVm

This is an extension of the VM class defined in the CloudSim library. It represents a virtual machine with a defined capacity. The additional data members included in the extension are the time values to hold the start time, renting time and remaining time and a Boolean value to indicate if a VM is set for completion (removal). Functions to get access to the above data members are also defined.
5 Outputs and Performance of the Simulation Framework

A number of simulations with different parameters were run and the following are plots from the data of the simulations. In each of the plots above the actual time in seconds of simulated time is the time unit given between two consecutive measures/recordings of transcoding data multiplied by the value in the horizontal axis. The 3 hour simulations are based on a sequence of Poisson random numbers, which represent the number of stream arrivals, but the 6 hours and above are simply a replication of the 3 hours arrivals.

5.1 Transcoding rates, Play rates and Number of VMs

Various plots are given below for different simulation scenarios, the parameters of the simulations of each plot is given. For all the simulations Lower and upper remaining time threshold parameters are 0.2 and 0.02 respectively and the LT and LP parameters n, q and k are 5, 10 and 30 respectively and the simulated time is 3 hours with Vm renting time = 600 secs. For each simulation the video streams for the simulation are picked randomly, i.e. no two simulations are simulating exactly the same video stream sequence.

Figure 5.1 is obtained from a simulation with a simulated time of 3 hours and with job length of 20 secs.

Figure 5.2 is obtained from a simulation with Simulated time of 3 hours, with a job length of 20 seconds.

Although the last two simulations on Figure 5.1 and Figure 5.2 are not run on exactly the same data set the expectation would be that the simulation with frequent allocation/deallocation updates would tend to be less smooth and more ragged in the
Figure 5.1: Plot Total transcoding rate and total play rate (Top) and Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (Bottom), with allocation/deallocation performed every 10 sec.
Figure 5.2: Plot Total transcoding rate and total play rate (top), Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (bottom), with allocation/deallocation performed every 20 sec.
number of VMs allocated plot.

Figure 5.3 is obtained from a simulation with 3hr simulated in, but with mean job size of 10 secs allocation/deallocation made every 20 secs.

All of the above simulations with the exception of the last one in Figure 5.3 were performed with a job length of 20 seconds. The simulation on Figure 5.3 is with 10 seconds jobs.

5.2 Simulation Time performances

The Table 5.1 shows the time it took to simulate different runs. The data below was collected by with a mean segment size of 20 seconds

The Table 5.2 shows the time it took to simulate for different mean job sizes. The tests simulated 12 days.
Figure 5.3: Plot Total transcoding rate and total play rate (top), Total transcoding rate and Predicted transcoding rate (middle) and number of VMs (top), job length = 10 secs
6 Future work and conclusions

6.1 Conclusions

In this work a new simulator for prediction based dynamic resource allocation distributed transcoding is built and tested for certain simulated times and job sizes. The need for a new simulator arose from the deficiencies in the previous simulator that was based on SimPy discrete event simulation tool. The framework chosen for the new simulator was CloudSim, which is a discrete event simulator that is specifically developed with cloud applications in mind. From the results of the tests it can be observed that the amount of resources required for transcoding (VMs) is changing in time following the change in the play rate/transcoding demand. With regards to performance the observations reveal that linear relationship exists between the time simulated by the system and the simulation time of the program. However, similar relationship was not observed when witnessing the simulation time for different job sizes.

6.2 Future work

A number of extensions can be done by using this simulator as a starting point. So far the work described in this report has been mainly based on the pre-existing first version python simulator. The next versions of the prediction based distributed video transcoding simulator, like the one with access control, can be extended on this simulator. The Vm selection method used in the simulator is shortest queue, that is a Vm with the shortest number of jobs is selected to process the next job requiring service. Different mechanisms of Vm selection can be tested either by replacing the shortest queue approach or integrating the shortest queue method with other mechanisms. One example is the duration of jobs in the queues of the Vms. This consideration could be useful in cases of jobs with significantly varying durations.
BIBLIOGRAPHY

