# Large Lab Exercise Cloud Computing: Video Converter on AWS

13-11-2012

*Author:*
N. Singh
N.R.N. Timal
n.singh-2@student.tudelft.nl
n.r.n.timal@student.tudelft.nl

*Support cast:*
D.H.J. Epema
B.I. Ghit
A.Iosup
d.h.j.epema@tudelft.nl
b.i.ghit@tudelft.nl
a.iosup@tudelft.nl

**Abstract**

## 1 Introduction

This document is constructed in response to WantCloud BVs interest in moving a new application to the cloud. To this end the CTO of WantCloud wants us to create a (relatively) simple application to identify what types of trade-offs exists when deploying an application on the cloud.

We chose Amazon Web Services (AWS)[1] as our cloud provider because it is one of the major companies providing resources to external parties to date. If WantCloud BV will follow through with their wishes to deploy an application in the cloud most likely AWS will be one to seriously consider. As part of AWSs policy it offers new users of the cloud a free usage tier[2]. For our system we solely used the accommodations provided by the free usage tier, in particular the free use of Elastic Cloud Computing(EC2) micro instances and Elastic Load Balancer.

We created an application which allows users to convert a video file to a desired format using a client-server model[3]. Although this may seem to be a simple application its workflow can contain very demanding operations. That is client(/user) needs to upload a video file to the server, the server in turn processes the uploaded file and converts and returns the converted file to the client. The client then stores the converted file to its local file system. We therefore find this application to be fitting for assessing the tradeoffs inherent in cloud based applications.

Our cloud based system will implement the following. Automation, which means that a servlet will automatically be assigned to a EC2 micro instance. Auto-scaling, that is depending on the amount of client requests addition (when CPU usage of existing instances are high) or removal (when CPU usage of an instance is below a low threshold and can be compensated by an existing instance) EC2 micro instances. Load balancing, equally dividing the work between instances using Round Robin scheduling[4]. Reliability is achieved attaching by every micro instance to a persistent storage volume. And last but not least monitoring which will provide a comprehensive view of e.g. the amount of resources used by each live micro instance. The implementation of the aforementioned features will be accomplished by interfacing AWS with Pythons boto framework. The code of this framework can be found at this location (http://code.google.com/p/boto/). This interface extends among other components of AWS to EC2 and Simple Storage Service(S3) which will be used by our cloud based system.

The rest of the report is organized as follows. Section 2 gives a more in-depth view of the application that is implemented. Section 3 discusses how the features of the cloud based system are implemented (automation, auto-scaling, load balancing, reliability and monitoring). The experimental setup and the experiments conducted are discussed in Section 4. Any limitations of our cloud based system based on the experimental results from Section 4 appear in Section 5. Finally the paper concludes in Section 6.

[1]http://aws.amazon.com/
[2]http://aws.amazon.com/free/
[3]http://en.wikipedia.org/wiki/Clientserver_model
[4]http://en.wikipedia.org/wiki/Roundrobin_scheduling

# 2 Application

We have constructed a video converter tool in Java[5] which allows an user to specify what file needs to be converted and to what format it will be converted. The conversion from one format to another is done in the cloud. Using a client-server model, the client uploads a file and additionally provides the output format. Whereas the server, which is provided by the cloud, deals with converting the uploaded file. When the conversion is done the converted file is written to the local file system of the client. We have used the Jersey[6]Framework to create a servlet which is constructed using a REST[7] based approach. This servlet contains the algorithm for converting a video and is deployed on a Tomcat server in the cloud. The algorithm uses several methods from the Xuggle[8] framework. This framework is free and open-source and is used for audio/video manipulation. As for the client we used Jersey Client Framework which provides a comprehensive interface for communicating with a server.

## 2.1 Requirements

In this paragraph we will elaborate what requirements are essential for our application. The requirements can be subdivided into two categories: *functional* and *nonfunctional requirements.* A fitting *functional requirement* for our application is that it needs to convert to and from all commonly known codecs/video formats e.g. Windows Media Video(WMV)[9] and QuickTime MOV[10]. As a *nonfunctional requirement* the servlet needs to be deployed on the cloud, to achieve this goal we will be using the facilities provided by AWS. In particular Amazon Simple Storage (S3) for storing the WAR file created by the Jersey Servlet and Amazon Elastic Compute Cloud(EC2). An EC2 instance will be comprised of our servlets computational logic, that is once the servlet is launched on an EC2 instance, the instance will be able to execute a video conversion.

[5]http://www.java.com

[6]http://jersey.java.net/

[7]http://en.wikipedia.org/wiki/Representational_state_transfer

[8]http://www.xuggle.com/

[9]http://en.wikipedia.org/wiki/Windows_Media_Video

[10]http://en.wikipedia.org/wiki/QuickTime

# 3 System Design

## 3.1 Resource Management Architecture

### 3.1.1 System Policies

# 4 Experimental Results

## 4.1 Experimental Setup

## 4.2 Experiments

## 5 Discussion

## 6 Conclusion