4 Case Study – the Big Brother Reality Show

4.1 Introduction

We define public, open ended, submission systems the class of software systems in which the public at large can submit a contribution in the format of a digital object. Typical examples of such systems are conference management and user generated content websites, where users can register and submit their contribution in the format of digital files. Such systems are, undoubtedly, a great improvement over manual systems that usually require the use of physical media and additional services, e.g. mail, at a much higher cost.

Recent technological advancements and the popularization of digital media capturing devices, specially photo and video cameras, have made it possible for the common citizen to produce multimedia files routinely [BREITMAN 2010]. The YouTube phenomena attest to this fact. Recently, it has been common to find applications, such as head hunting systems that require video footage in addition to the candidate's resume. As a potential drawback of this trend, multimedia files may require a lot of processing power, to encode, process and compress submissions to some desired standard or file format at the receiver's end.

Thus, the combination of public, open access systems, that allow users from around the globe to make submission at very little cost and the fact that these submissions may very well contain large multimedia files, poses a great challenge to the conception and design of such systems. Estimating storage space and processing power to enable such applications has become harder and demand non-trivial solutions.

In what follows we argue that Cloud Computing technology provides the necessary requirements for a viable solution. It provides the necessary infrastructure to develop submission applications in which both storage and processing needs can be dimensioned as needed. For this purpose we propose a general cloud-based architecture for open, public submission of user-generated content.

This paper is organized as follows. Section 4.2 summarizes the major characteristics of cloud computing environments. Section 4.3 introduces a general architecture in which to implement scalable submission system applications. Sections 4.4 and 4.5 demonstrate the viability of the proposed approach with a case study. In Section 4.6 we present our concluding remarks.

4.2 Cloud Computing

This section briefly summarizes Cloud Computing aspects relevant to this paper.

The Cloud can be both infrastructure and software, meaning it can be an application that can be accessed through the Web or some servers that can be provisioned exactly when needed. It's a paradigm of computing in which dynamically scalable and usually virtualized resources are provided as a service over the Internet.

From the infrastructure point of view there are some interesting aspects in Cloud Computing [ARMBRUST 2009][VOGELS 2008][MILLER 2009][VOUK 2008]:

• The perception of infinite computing resources available on demand, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning;

• The mitigation of an up-front commitment by Cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs;

• The ability to pay for use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful [REESE 2009].

There are a few commercial companies, e.g., Amazon, Google, HP, and IBM that offer Cloud Computing services to the public in general. For the purposes of this paper we'll focus on Amazon Cloud services, which can be split in different infrastructure services as detailed bellow.

Amazon S3 – Amazon Simple Storage Service is cloud-based persistent storage and operates independently from other Amazon services. [AWS S3] It can be used to upload data in the cloud and pull it back out. Its infrastructure is very primitive when compared to traditional file systems [CHANG 2008][GHEMAWAT 2003].

Amazon EC2 – Amazon Elastic Compute Cloud is a web service that provides resizable compute capacity in the cloud. [AWS EC2] It provides an API for provisioning, managing, and deprovisioning virtual servers inside the Amazon cloud. It's the heart of the cloud and allows remote deployment of virtual server with a single web service call.

Amazon SQS – Amazon Simple Queue Service is a highly scalable, reliable, hosted queue for storing messages as they travel between computers. It can be used to move data between distributed components of an application that perform different tasks, without losing messages or requiring the components to be always available.

Amazon SimpleDB – Amazon SimpleDB is a web service that provides core database functions such as data indexing and querying in the cloud. It functions as a very simple relational database.

Please refer to [AWS S3][AWS EC2] for more complete descriptions.

4.3 Proposed Architecture

In this section, we describe the proposed architecture for a system to process user generated content [JENSEN 2008] that must be stored and later reviewed using a Content Management System. The system should be able to receive video files encoded in any format, uploaded by users using a web-based application.

The content received is stored in Amazon S3, and for each video received an input message is written in SQS Queue to instruct EC2 instances to process a new job. The EC2 job must take care of transcoding the video to a standard format (mpeg4/h.264/aac) that could be easily reproduced by any video player (e.g. flash player). To guarantee that our storage can scale horizontally, the output video file is stored at Amazon S3 once again.

The queue (SQS) is consumed by an EC2 instance that monitors for input jobs and uses FFmpeg [TOMAR 2006] to do the transcoding. Output video files are stored in S3 and a new message is written on SQS with data of the job that just completed.

Figure 4.1 illustrates the basic architecture.

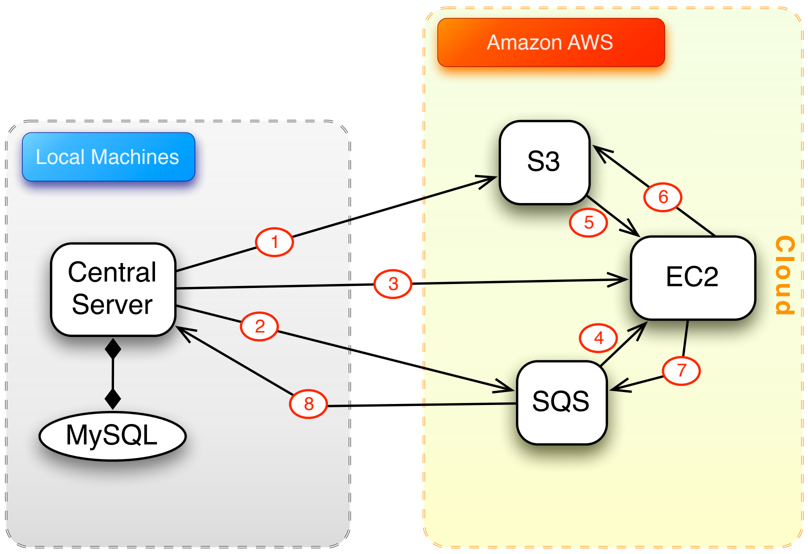
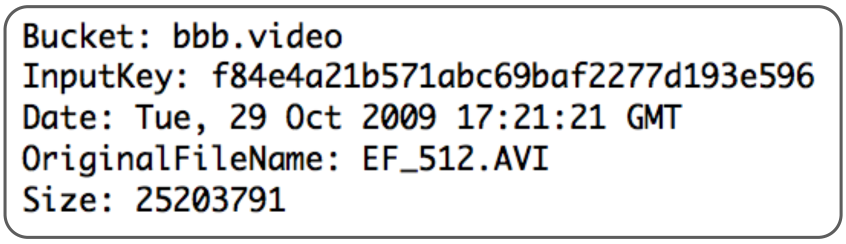


Figure 4.1. Proposed architecture for the public submission system.

We detail the process in the following basic steps:

1. Video submitted by user is stored in Amazon S3;
2. Local server writes the message in the input queue of SQS detailing the job to be done;
3. Local server creates a new EC2 instance to process the job;
4. EC2 instance reads the message from the input queue;
5. Based on the data of the message the input video is retrieved from S3 and stored locally in the EC2 instance;
6. Video is transcoded by EC2 and the generated output is stored in S3;
7. EC2 instance writes a message in the output queue describing the work performed;
8. Confirmation of the work completed is read by the local server from SQS output queue.

SQS messages use the basic structure format adopted by mail messages and HTTP headers and defined in RFC-822 [CROCKER 1982]. Input messages are as follows:

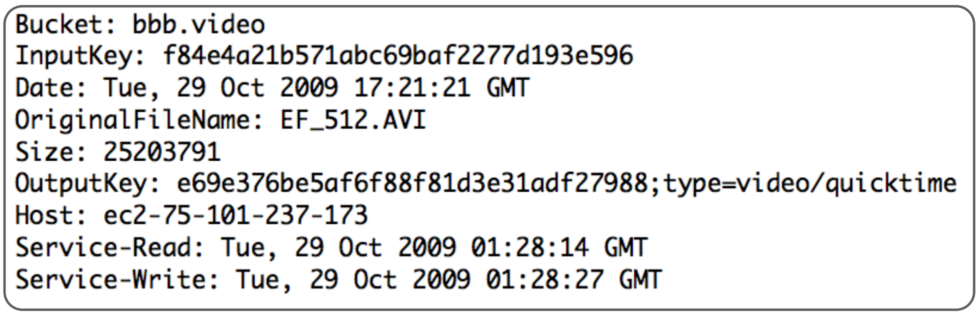


```
Bucket: bbb.video
InputKey: f84e4a21b571abc69baf2277d193e596
Date: Tue, 29 Oct 2009 17:21:21 GMT
OriginalFileName: EF_512.AVI
Size: 25203791
```

Figure 4.2. Example of data in the input message used by the application.

Where Bucket and *InputKey* are the identifiers of the file in the S3 infrastructure, and *OriginalFileName* is the source filename. Web services were implemented using the Boto library [GARNAAT 2006].

The output message is defined as:



```
Bucket: bbb.video
InputKey: f84e4a21b571abc69baf2277d193e596
Date: Tue, 29 Oct 2009 17:21:21 GMT
OriginalFileName: EF_512.AVI
Size: 25203791
OutputKey: e69e376be5af6f88f81d3e31adf27988;type=video/quicktime
Host: ec2-75-101-237-173
Service-Read: Tue, 29 Oct 2009 01:28:14 GMT
Service-Write: Tue, 29 Oct 2009 01:28:27 GMT
```

Figure 4.3. Example of data in the output message used by the application.

Where we also define the hostname of the EC2 instance that processed the job and the timestamps when the job was received and when it finished.

There should be noted that the use of the queue does not constrain the system from processing multiple files simultaneously. The system is capable of receiving simultaneous files from different users, process and upload them to S3. Each process will have its own job messages associated in the queue. The only limitation regarding multiple file processing will be the upload bandwidth while sending the files to the cloud infrastructure (bandwidth will be shared).

The EC2 instance that is launched uses a specific Amazon Machine Image (AMI) created with all the dependencies necessary to process the video. That includes an updated version of the Linux kernel, *git* to retrieve the latest source code available for this framework, python and the FFmpeg software, which does the actual video transcoding.

Once the AMI image is instantiated it reads a configuration file that keeps parameters as:

• FFmpeg command line and arguments;

• Maximum processing time before marking the job as dead;

• Input queue name to read SQS messages;

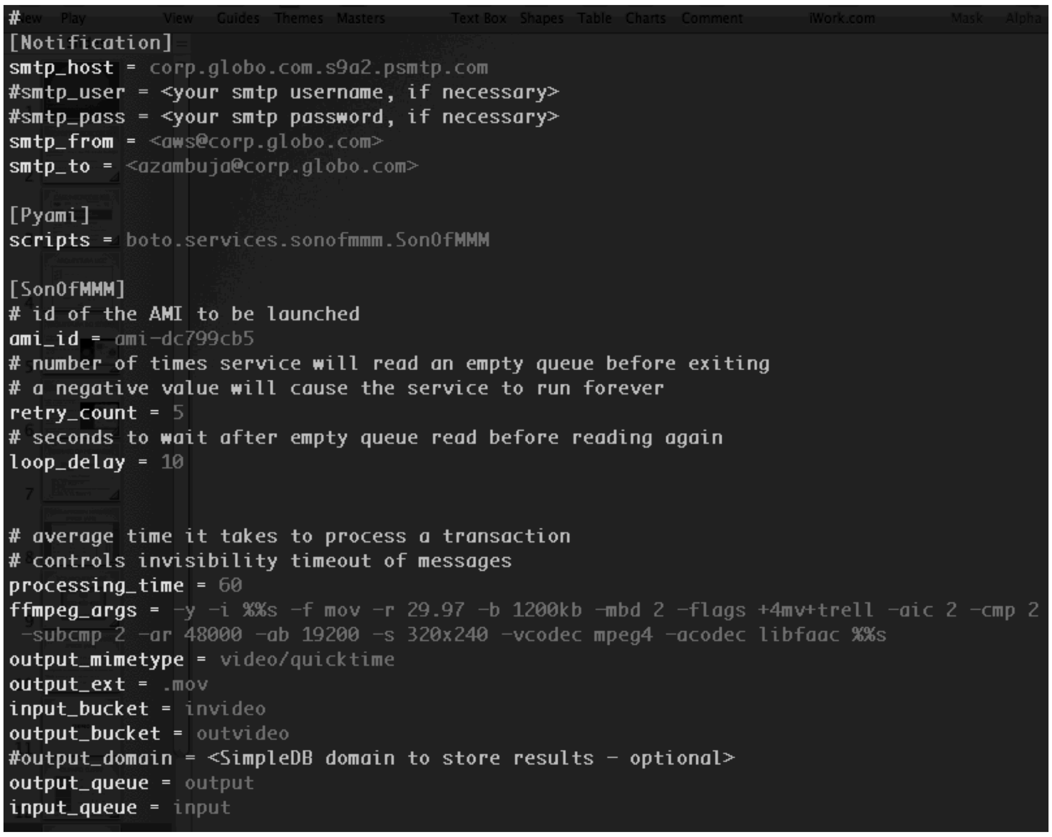
• Output queue name to store SQS messages;

• Maximum number of retries in case of error;

• Notification e-mail (for debugging purposes);

• Python class to be invoked for the processing.

The configuration file is shown in figure 4.4.



```
#
[Notification]
smtp_host = corp.globo.com.s9a2.psmtp.com
#smtp_user = <your smtp username, if necessary>
#smtp_pass = <your smtp password, if necessary>
smtp_from = <aws@corp.globo.com>
smtp_to = <azambuja@corp.globo.com>

[Pyami]
scripts = boto.services.sonofmmm.SonOfMMM

[SonOfMMM]
# id of the AMI to be launched
ami_id = ami-dc799cb5
# number of times service will read an empty queue before exiting
# a negative value will cause the service to run forever
retry_count = 5
# seconds to wait after empty queue read before reading again
loop_delay = 10


# average time it takes to process a transaction
# controls invisibility timeout of messages
processing_time = 60
ffmpeg_args = -y -i %%s -f mov -r 29.97 -b 1200kb -mbd 2 -flags +4mv+trell -aic 2 -cmp 2
 -subcmp 2 -ar 48000 -ab 19200 -s 320x240 -vcodec mpeg4 -acodec libfaac %%s
output_mimetype = video/quicktime
output_ext = .mov
input_bucket = invideo
output_bucket = outvideo
#output_domain = <SimpleDB domain to store results - optional>
output_queue = output
input_queue = input
```

Figure 4.4. Configuration file for EC2 instance.

4.4 Case Study

The Brazilian Big Brother reality show is broadcasted by free-to-air TV network with an audience of more than fifty million people simultaneously. The idea behind this reality show is to portray the life of 16 random anonymous people while living together under the same roof, for a total period of three months. They are isolated from the outside world but are continuously monitored by television cameras. The housemates try to win a cash prize by avoiding periodic evictions from the house.

The application process to participate in the reality show is open to any resident in the country, and usually requires that candidates send a videotape (by mail) with a small video clip introducing them. In the last edition (2009) participants were also required to create a profile in a social network web site.

With technological advances the application process evolved from sending a videotape by mail to uploading a digital video using the Internet. Due to legal reasons, videos can not be hosted in websites such as YouTube or Vimeo. Applicants are allowed to send videos in the video format of their choice. These must be stored until the end of the selection process (three months). All the videos need to be transcoded to a standard format, so that the production team is spared from the hassle of having to deal with a plethora of video formats and different codecs.

The system should be able to receive a very large number of videos during the three-month application process. With the new digital process it is expected that more than 200,000 videos, about 60% of the total submission is uploaded during the last week before the deadline.

A few specific characteristics leverage the use of Cloud Computer architecture for this project in particular:

• Uncertainty in how much storage and processing capacity will be needed;

• Resources will be needed during the application and selection processes only. After this period they can free the resources;

• Few but extreme high peak situations where the infrastructure will need to scale up – 60% of the videos are expected to be sent in the last week.

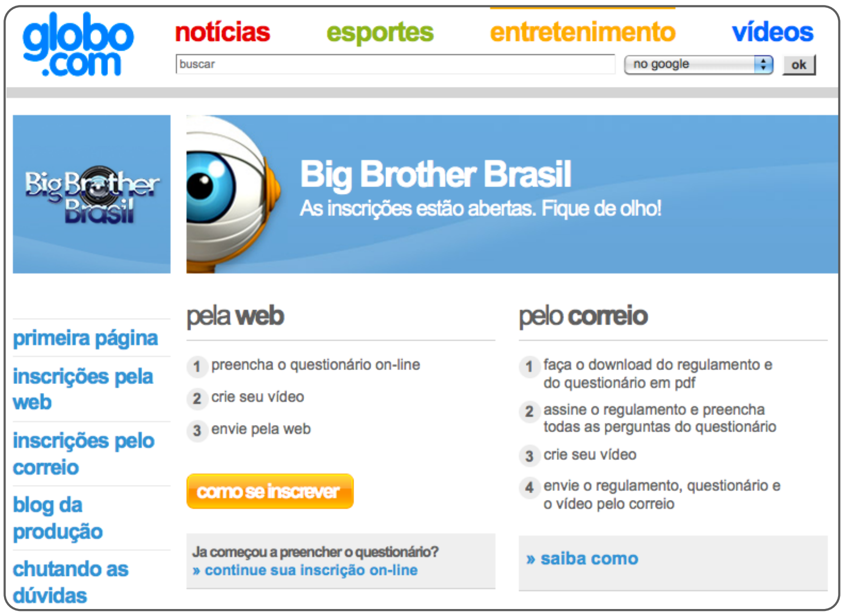
The proposed architecture uses the cloud to store and process all this content, and to provide storage availability and scale resources as needed.

4.5 User Generated Content Architecture

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

In this section we describe the already existing infrastructure to support the reality show. The cloud architecture described in section 3 is a new component of this existing system.

To start the process the user must go to the reality show's website [BBB 10] and create a user account.



Big Brother Brasil
As inscrições estão abertas. Fique de olho!

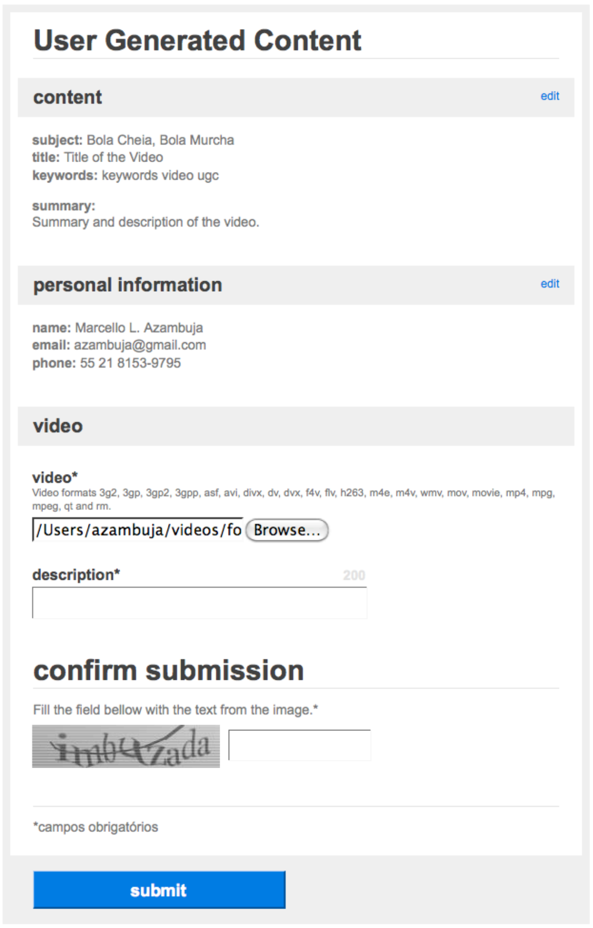
primeira página
inscrições pela web
inscrições pelo correio
blog da produção
chutando as dúvidas

pela web
1 preencha o questionário on-line
2 crie seu vídeo
3 envie pela web
como se inscrever
Ja começou a preencher o questionário?
» continue sua inscrição on-line

pelo correio
1 faça o download do regulamento e do questionário em pdf
2 assine o regulamento e preencha todas as perguntas do questionário
3 crie seu vídeo
4 envie o regulamento, questionário e o vídeo pelo correio
» saiba como

Figure 4.5. Web site for the Big Brother reality show.

Once logged in, some meta-data information related to the video must be filled in, as shown in figure 4.6.



User Generated Content

content
edit

subject: Bola Cheia, Bola Murcha
title: Title of the Video
keywords: keywords video ugc

summary:
Summary and description of the video.

personal information
edit

name: Marcello L. Azambuja
email: azambuja@gmail.com
phone: 55 21 8153-9795

video

video*
Video formats 3g2, 3gp, 3gp2, 3gpp, asf, avi, divx, dv, dvx, f4v, flv, h263, m4e, m4v, wmv, mov, movie, mp4, mpg, mpeg, qt and rm.
/Users/azambuja/videos/fo Browse...

description* 200

confirm submission

Fill the field bellow with the text from the image.*

*campos obrigatórios

submit

Figure 4.6. Meta-data information for the UGC.

The User Generated Content architecture is illustrated in the figure bellow.

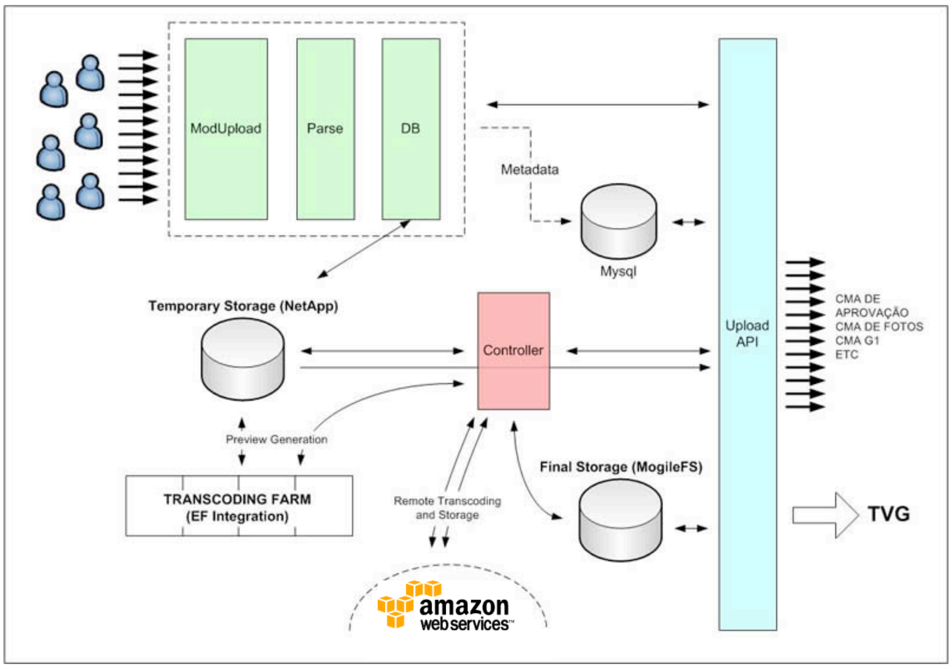


Figure 4.7. User generated content architecture.

In the following paragraphs we describe how the system used to work before the cloud component was added.

Once the received the full raw HTTP POST used to send both video and its meta-data (video title, description and user information) is stored in the local file system. This job is done by an apache module called ModUpload.

A daemon monitors the local file system for recent uploads and parses the raw HTTP POST data separating metadata information from the video file. Metadata is then written in a local MySQL database and the video file is stored in a temporary local storage.

These three steps are represented in figure 5 by the green boxes (ModUpload, Parse and DB).

The video would then be transcoded by a local transcoding farm and stored in the local final storage using MogileFS.

With the integration of the Cloud component, as described in section 3, there's no longer the need to have a local storage and a local transcoding server farm. As already described in the previous sections the content is stored using Amazon S3 and transcoded using EC2 instances.

4.6 Conclusion

The main contribution of the paper is twofold. Firstly we characterized a class of systems that pose great design and implementation challenges, thus make them excellent candidates for Cloud Computing solutions. Among those challenges are the near impossibility of estimating total number of submissions, storage required, and the punctual needs for a large amounts of processing power. The second contribution of the paper is describing a scalable architecture, and implementation, that can be generalized to similar web-based, public submission systems.

If we were to calculate how much money would be spent to process 100,000 videos with an average size of 15MB, using the small EC2 instance we can process a video in 50% real time, that would require 834 hours of CPU running.

Storage	1.5 TB	U$0.15 / GB	U$ 225.00
Transfer	1.5 GB	U$0.14 / GB	U$ 210.00

Messages	200,000	U$0.01 / 10,000 requests	U$ 0.20
Computer Resources	834 hours	U$0.085 / hour	U$ 70.90
Total			**U$ 506.10**

Table 4.1. Cost analysis of transcoding solution in the Cloud for 100,000 videos.

A total of U$506.10 for transcoding and storing 100,000 videos – that's not even a penny for each video.

Adding to that the fact that no up-front investment and deployment of infrastructure was needed, neither a precise estimation on the expected load of the system we can conclude that Cloud Computing is an excellent solution in this specific scenario. It is important to remark that economical viability of the proposed architecture is such that enables it to quickly deploy at great reduction of the TCO, typical of Cloud Computing implementation [WALKER 2009].