Quoc Quan Duong

Distributed video processing system on cloud servers

Helsinki Metropolia University of Applied Sciences

Bachelor of engineering

Media engineering

Bachelor's thesis

November 4, 2016

Author(s) Title Number of Pages Date	Quoc Quan Duong Distributed video processing system on cloud servers 54 pages + 2 appendices 4 November 2016
Degree	Bachelor of engineering
Degree Programme	Media Engineering
Specialisation option	Mobile programming and .Net programming
Instructor(s)	Aarne Klemetti, Researching Lecturer

With the rapid growth of the internet, connecting with other people has never been so easy. Every second, millions of information snippets are transmitted through the internet, which contains huge amounts of image, audio, and video. To handle large amounts of data, it requires a complex system and powerful hardware.

Big companies such as Facebook and YouTube are applying their own solutions to handle many video files daily. The purpose of this study is to provide a solution to build a distributed video processing system on cloud environment.

With the help of cloud computing and many tools available such as NodeJS, FFmpeg, Amazon web services, this study will provide a complete solution to develop a distributed processing system. Moreover, the aim is that achieving a solution for secure video processing which is high performance and easy to scale.

The purpose of this study is to set up a common approach for processing media files as well as handling a large number of tasks for servers. This study not only used for the research purposes but also can be applied directly to production.

Keywords	Distributed system, video transcoding, parallel processing, cloud computing, Amazon web service, FFmpeg, real-time web socket, NodeJS

Contents

Abbreviations and Terms

AAC	Advanced Audio Coding
API	Application Programming Interface
AWS	Amazon Web Service
CORS	Cross-origin Resource Sharing
CPU	Central Processing Unit
DNS	Domain Name System
EC2	Elastic Compute Cloud
ELB	Elastic Load Balancing
GPL	General Public License
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model View Controller
NoSQL	Not only SQL
RAM	Random Access Memory
S3	Simple Storage Service
SDK	Software Development Kit
SQL	Structured Query Language
SSD	Solid-state Drive
SSH	Secure Shell
TCP	Transmission Control Protocol
UML	Unified Modelling Language

1 Introduction

Today, rich media content on the internet is very popular. Accorded to Cisco's report, by 2020, the video will account for 82% traffic internet traffic worldwide [1]. Social network and entertainment channels such as Facebook and YouTube have created a revolution of human communication. Clearly, the demand for video content is huge. When someone is using their device to record a video and then uploads that video to a certain website, the process may sound simple from the user side. However, there is a complex processing chain behind it. The reason is that the format of the video to be displayed in a web environment has its own regulations, which sometimes differ from regulations of the recording devices. Therefore, they will be processed before officially streamed to the viewers. On the other hand, the application for online video editing starts launching. This work requires a complex system; EdVisto is an application discussed in this study. EdVisto is a product of DiSel21 Oy, a startup in the field of education. This service provides new learning methods for students and teachers with an idea "Learn through storytelling". Technically, in this case, the story is created by videos.

The difficulty is to handle various formats of video before they are transmitted to the user. In addition, EdVisto provides many video editing tools such as adding effects, collage, add background music, add text, transitions and a lot of other features. There are many similar software installed on the user's device, which users can apply to handle video independently, using the capabilities of the device, such as laptops or mobile. The difference with EdVisto is an application running on the cloud, which means all processing tasks are performed on the server. This is beneficial for the users as they do not need to have a good hardware device, all data is stored safely in the cloud and they can easily share the results to everyone. But those features also bring many challenges to the technical team.

The image processing is very expensive, especially coupled with the large files. With a powerful computer, the processing can take hours. So, the problems are how to reduce the processing time down to a minimum and to be able to serve many users. The three main questions for the research are:

- How to handle multiple video formats?
- How to improve performance of processing?

- How to serve a very large number of concurrent users?

To answer those questions, distributed computing has been selected as a solution. This solution will be presented in detail in the thesis.

2 Distributed system

Distributed processing

Distributed System is a network of independent computers, which communicate to each other by dispatching messages to solve a common task. One system to be considered as distributed must have these characters: concurrency of components, lack of a global clock, and independent failure of components [2, 1].

The term concurrency in computing is the number of work executed independently and not in an order, but the partial outcome will be ensured remain in the order as income [3, 1]. This allows parallel processing which decreases the total processing time. For example, a client has a big file which needs to be downloaded from the internet. The big file has been divided into several smaller parts which can be transmitted at the same time. Practically, the smaller parts may finish in different order, but the results of the big file always stay the same as the original. This process is presented in figure 1.

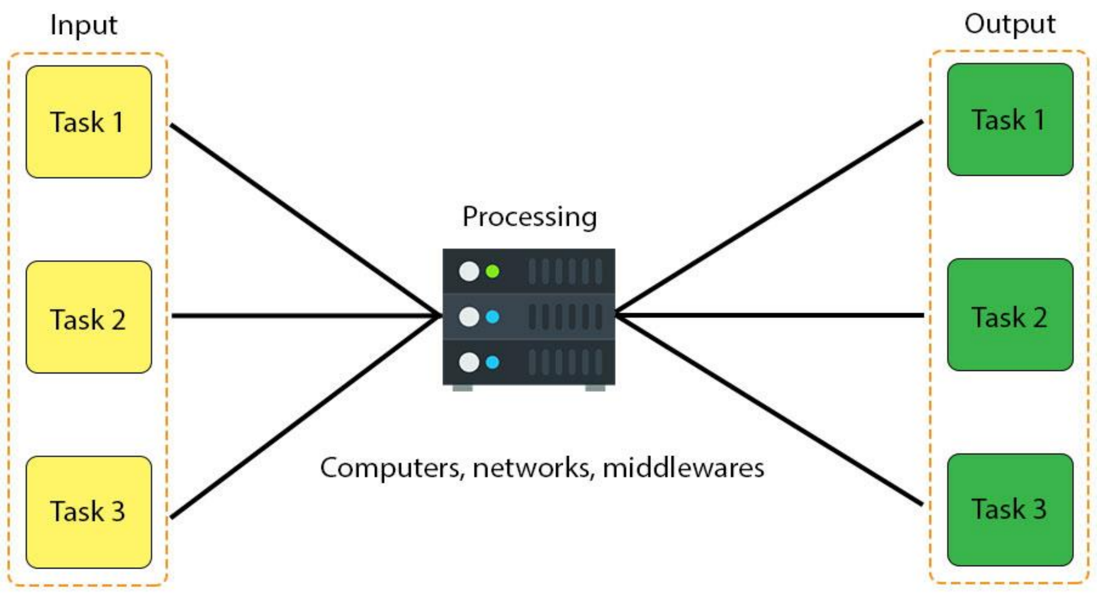


Figure 1: Concurrent processing

In a distributed system, each node may have a different time rate, there is no global clock for all nodes [4, 1]. The system is designed to run the tasks in sync. However, in a single node, the clock is not the same. For instance, many people have their own watch in the same location, but the time in each watch will never be 100 percent identical to others. To be able to cooperate, the nodes communicate by exchanging messages.

One important thing in distributed system is handling the failure of the component. A component can be crashed and causing failures unpredictably and independently. Meanwhile, the other components are still running the process and may not be able to recognize the failures [2, 2]. Figure 2 shows about 45% of machine restart in six months and in one week, there is approximately 1% of all machine restart more than once at Google.

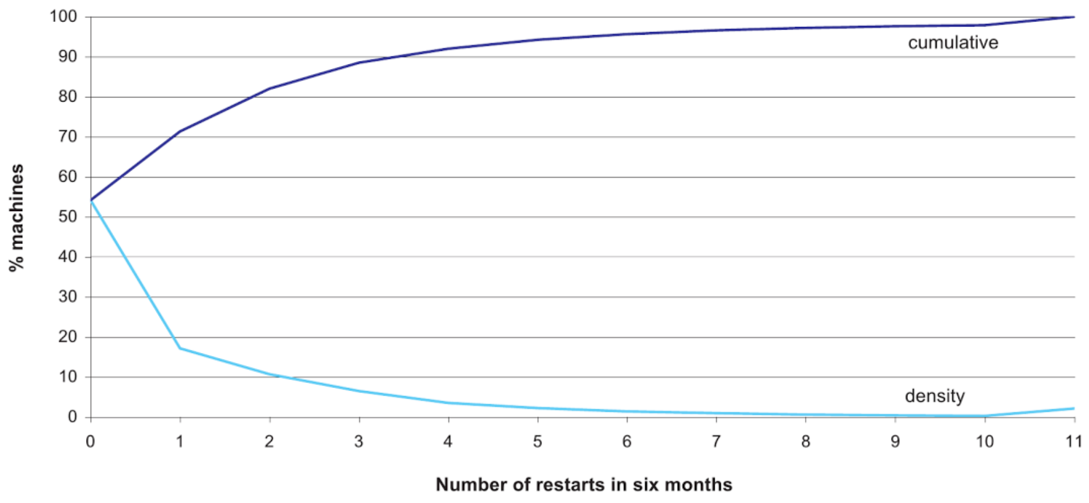


Figure 2: Number of restarts of machine in six months. Reprinted from Luiz (2009) [5,84]

There are many reasons for the downtime. Based on Google report, the main reasons are human activity and software. Hardware fault only takes less than 10% of system failure [5,80-84]. The system needs to handle the failure automatically to ensure the whole processes is not interrupted by a single failure.

Distributed file system

In a single computer, data are represented into binary format, a number system that uses zero and one stored on the hard drive. A group of binary pieces tracked and identified by the system is called file. Files can be organized in the folder hierarchy. The main duty of this file system is to store and retrieve data in the local storage device. On the other hand, a distributed file system is designed to manage data not only in the bounds of computer hard disk, which consists of multiple storage nodes that are connected to a network. There are many distributed file systems implemented today, one of the most popular frameworks is Hadoop, an open-source software developed by Apache Software Foundation, which has its own filesystem called HDFS (Hadoop distributed file system). HDFS is designed to manage data similarly to the traditional file system. The strong advantage of HDFS is highly faulted tolerant, which is suitable to be used in common hardware. Furthermore, HDFS allows to store and stream big size of data which can be counted by gigabytes or terabytes [6]. This study will not go into detail of Hadoop, but will mention a typical feature of a distributed storage is data replication, thus HDFS is a good example.

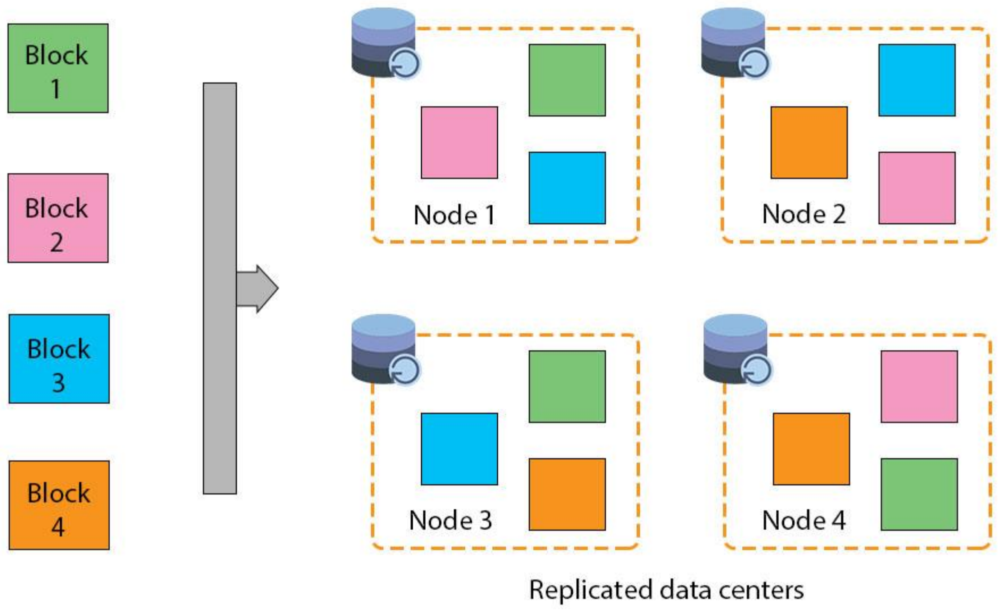


Figure 3: HDFS replicated data

Figure 3 shows HDFS structure. The file is stored in a sequence of blocks; the blocks are replicated in several nodes for fault tolerant. This means that a copy of data block is stored in multiple nodes, thus in case one node is in downtime, the integrity of the file is guaranteed. The failure rate of hard disk driver is high. According to the report about hard drive reliability in 2015 by BackBlaze, the average rate of failure is about 5.84% of total hard disks [7]. Therefore, more blocks are replicated, more data are safe. Data stored in HDFS then can be accessed by multiple nodes, accessing speed will depend on the hardware bandwidth and the physical distance between the nodes. The free space of storage depends on how many nodes connected to the networks. With the same storage hardware there are more machine nodes in the network, more space is available to use.

Advantages

There are many benefits that the distributed computing contributes. Following are the basic advantages of the distributed system:

- Availability: A system is supposed to be availability when it can ensure the maximum uptime. In the operation, if the incident happens for one part, a replacement quickly is necessary to ensure the system is not be down. Of course, this process must be automated in a quietly, but in the user side there is no notice. A network with multiple redundant nodes in the distributed system can ensure this requirement.
- Performance: This is one of the main purposes of the distributed computing. When the number of tasks to a computer is overloaded, the distributed system can complete the same tasks much faster. The works will be split up and processed simultaneously, or optimized by a computer network. Because it takes advantage of the processing power of multiple computers at the same time, therefore it can provide high performance significantly.
- Reliability: If data is stored by a computer, when the incident occurred, all the information might be lost. However, with the distributed system, by replicating data to multiple machines, in many different locations, data is always ensured be safe. This reduces the probability of losing data.
- Scalability: In the way to upgrade a system we usually have two concepts, vertical and horizontal upgrading. For a computer, the upgrade can only apply the first way, which is to upgrade the hardware. For example, if a 2TB storage drive is not enough, we can buy another hard drive with a larger capacity installed, assuming 10TB. The problem can be solved well in some cases, but what happens if the increased demand is 100TB or millions of TB. Obviously, the downside of the vertical upgrading is depending on the limitation of the hardware device. Thus, the distributed system is designed to solve this problem. When the system needs to have an upgrade, just simply add a computer unit into the network. By the horizontal upgrading, there is no limitation of the hardware.

- Economy: As noted above the distributed system upgrades are simple and not limited by hardware. Instead of having to buy the high-end hardware with a high price, we can fully use the common hardware. The increase in power for the system to be adjusted based on demand. It is understood, for example, instead of having to pay for a high configuration computer, where demand is not yet meet the capacity of the machine, we can fully build a system with cheap computers and expand based on demand. Thus, capital cost is minimized. [8.]

The above is the list of the advantages of distributed system, the next chapter will discuss its challenges.

Challenges

Benefits come with challenges. The design and management of the distributed system are more complicated than a single system as discussed below.

- Heterogeneity: In a network, the computers may be different in terms of hardware, operating system, software, protocols, programming languages, but the design must ensure that they can communicate to each other.
- Transparency: For users to services provided as a single system, all operations of the distributed computing are done silently in the background.
- Fault tolerance: The system must be able to handle the failure. If one node fails, they will be separated from the system, without affecting the whole system, and the job is to continue processed.
- Concurrency: The tasks and the resource to be accessed and processed simultaneously and the results must synchronize with each other in the original order.
- Security: Data must be kept confidential, preventing unauthorized access. In addition, the system must be able to withstand the attacks by denial of service (DDOS). [8.]

Although there are a lot of challenges, with the help of cloud computing and good design, those issues can be solved.

2.1 NodeJS

NodeJS is an open source JavaScript runtime environment built on Google's V8 engine. Although JavaScript beginning was designed for running on client's browser, but by NodeJS, it can be also deployed in the server environment. This makes JavaScript more powerful.

The main idea of NodeJS is asynchronous event-driven and non-blocking I/O. In some other programming languages based on handling the threat, operations are executed synchronously. This means the next operation must wait until the current one is done, which is not efficient for the tasks to be executed parallelly. Instead, NodeJS uses non-blocking I/O model, which means the next operation can continue to be executed while the current one still in the process. This idea makes NodeJS lightweight and more efficient [9]. Figure 4 shows the benchmarks of NodeJS compares to PHP and Python, higher is better:

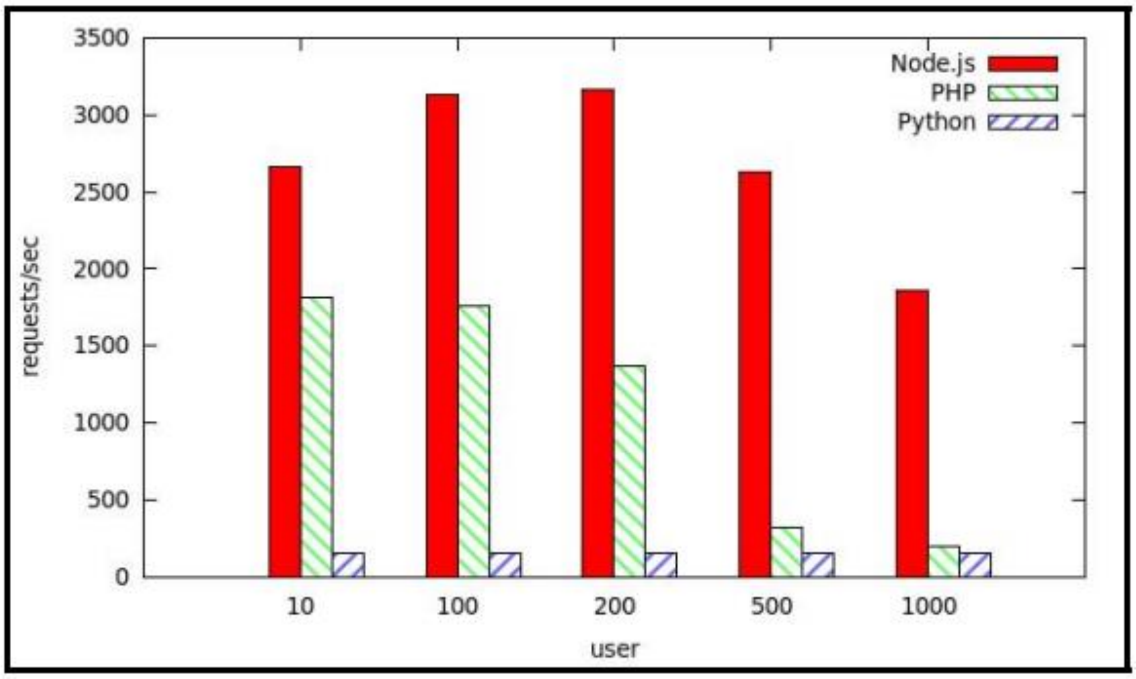


Figure 4: Requests per second in different languages. Reprinted from Kai (2014) [10]

NodeJS is a good choice to build scalable distributed system with intensive I/O requests like video streaming service, game online, or real-time application. There are many big companies who are using NodeJS for their projects, for instance: Microsoft, PayPal, Uber, Netflix, and LinkedIn [11].

WebSocket and socket.IO

WebSocket is a protocol to make a full-duplex communication channel with a single TCP connection [12]. Before WebSocket, the long-polling solution has been used to maintain a two-way connection between client and server. By sending interval HTTP requests, it is wasted network bandwidth for the header each time a request is sent. Another downside of this method is the delay time between two requests. Fortunately, WebSocket comes up with the idea to avoid unnecessary header payload, also improve the performance and provide the real-time connection. Moreover, WebSocket can use the same port 80 with HTTP and 443 with HTTPS which are the common TCP ports [13]. Therefore, WebSocket is not blocked by a firewall in the highly secured environment like in the library or office network.

For integrating WebSocket to NodeJS, there is an open source library called socket.IO which is running on multiple platforms. Socket.IO provides the module packages for both client and server. The setup as well as calling API of socket.IO is handy. API allows to emit, receive or broadcast an event between server and client. Data payload can be of any type, from JSON, text, and blob. With a combination of NodeJS and socket.IO, it is faster and more convenient to build a real-time application. Many popular applications are using socket.IO such as Microsoft Office, Slack, and Trello [14].

3 Cloud computing

Concept

Cloud computing is services which provide the access to data or programs via remote network instead of the local one [15]. With cloud computing, the client can use the computing resource and access to data storage on demand with minimum configures. The idea of cloud computing resource sharing to provide high capacity, scalability, manageability and economic cost. Figure 5 illustrates the cloud computing.

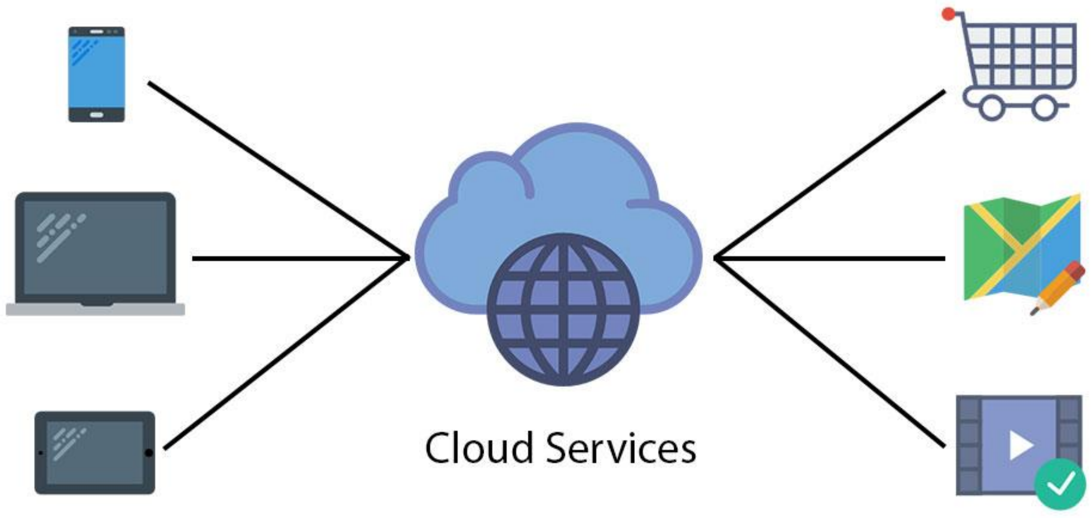


Figure 5: Cloud computing

There are models of cloud computing services:

- Software as a Service: similar to the common software services, but is deployed and delivered through cloud network. With this service, the client can access the software through multiple devices, not depend on configuration and operating system of the device. Users control software via a network or the internet. All tasks will be handled in the cloud system, where users only need a thin interface to access, such as a web browser.
- Platform as a Service: providing an environment for the development and administration including operating systems, libraries, programming languages and tools. Clients have no right to interfere in the operating system, such as network hardware, CPU, memory, but will be able to have the configuration, certain software setup environment.
- Data as a Service: this service provides a data storage solution with high capacity. Typically used the distributed data storage technology. There are many types of the database can be provided as SQL, NoSQL, or file storage.
- Infrastructure as a Service: providing services related to hardware such as CPU, storage, memory, network or another computing resource. Users can self-install the operating system, applications, customize the setup related to hardware, without the permission to modify the hardware. There are existing services such as Amazon's Elastic Compute Cloud or Google Compute Engine. [16.]

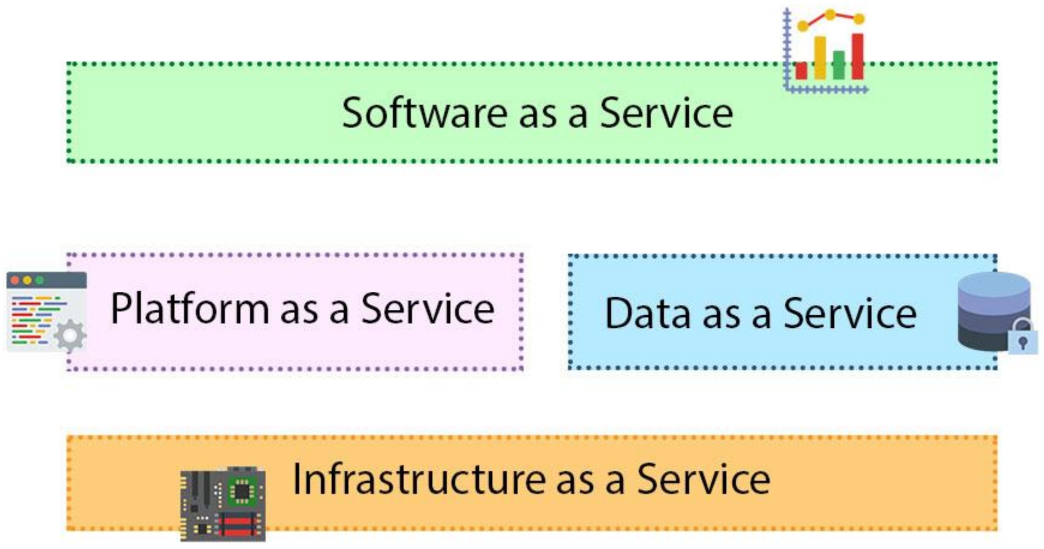


Figure 6: Cloud services. Data gathered from Qusay (2011) [16]

Figure 6 illustrates the cloud service models. These models can be supplied in separate or combined to provide users with a complete solution.

Benefits

Cloud computing brings a lot of benefits. Firstly, the flexible scalability of the cloud system helps the enterprise save the expense. Clients only need to pay for resource be used. While demand increased, the cloud can be easily upgraded. Moreover, the client does not have to invest in a full system with real hardware or hire technical staff to manage the team, which is very expensive. Software licensing fees are also problematic. Another very important issue is reliability. What will happen if the system having a problem? The data might be lost. However, the data is stored safely in the cloud, the data center is backed up in many different locations, and the risk of information loss is minimized. The cloud services managed by the professional third party will also help the company save time and focus more on its products.

3.1 Amazon Web Service

Amazon web services (AWS) is a product about cloud computing of Amazon Inc. Launched in 2006, AWS has provided infrastructure services to enterprises with thousands of servers spread across 13 territories [17]. Currently, Amazon has data centers in the U.S., Europe, Brazil, Singapore, Japan, and Australia. There are many cloud services integrated into the system, typically, Elastic Compute Cloud (EC2), Simple Storage Service (S3), Elastic Load Balancing (ELB), Simple Queue Service (SQS) and Auto Scaling. [18.]

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) service is an important key in the Amazon cloud services. EC2 offers customers a virtual server solution with scalable, customized configuration. In a virtual environment, customers can install the customer operating system and freely set up the applications. All the necessary software, including operating systems, applications, libraries, data and configuration settings are wrapped in Amazon Machine Image (AMI) as a package and can be stored or deploy to EC2 later. Currently, Amazon supports many different operating systems including Linux and Windows.

EC2 types with different hardware options are specialized to serve different purposes. For example, General Purpose with T2, M3, M4 instance, Compute Optimized with C3, C4 and Memory Optimized with X1, R3 instance. In the running time, if the memory is not enough, it can be upgraded to another instance without losing data, or if the store is missing, we can attach additional storage block in the current instance. Moreover, with the combination of multiple instances, it is possible to create a distributed system through a high bandwidth connectivity and low network latency between the EC2 machine nodes. [19]

Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) provides a cloud storage solutions which is simple, secure and scalable. Amazon S3 stores is a big pool which stores data in the form of a key-based object in one place called bucket. The capacity limit of one file at the current time is 5 terabytes. However, there is no limit to the number of files stored. With amazon S3, users can store, access, download, copy, or delete files from a file system which is like a file system on a computer. The difference is that users can manage files from anywhere through a browser or provided APIs if there is the internet. Amazon S3 basically is a distributed data storage on the cloud.

Regarding reliability, the files are stored redundant, replicated in multiple data centers, which Amazon claims that 99.999999999% of durability and 99.99% of availability. This means if you store 10,000 objects the possibility of losing one object file is expected in a cycle of 10,000,000 years. [20.]

Amazon Elastic Load Balancing

Amazon Elastic Load Balancing (ELB) is a solution for auto distributing requests to multiples EC2 instances. ELB has two types of service: classic load balancer and application load balancer. Classic load balancer routes traffic based on application or network level or information and application Load Balancer operates at the application layer and content of the request [21]. For example, in Classic load balancer, multiple requests sending to server will be distributed to different EC2 instance node which are connecting to the ELB. On the other hand, in application load balancer type, it is able to filter the requests to a specific target. For instance, all request has prefix "/api" will only be transferred to a group of the EC2 machine which handling API requests. In general, ELB is a service to help to scale easily and limit failure during operation. Due to automatic checking mechanism of the instance health, ELB can limit the failure. At this time, ELB operates at Layer 4 (transport) and layer 7 (application) [22], support HTTP and WebSocket protocol [23].

Amazon Auto Scaling Group

As described, the task of the ELB is automatic distributing requests to EC2 instances. Combined with ELB, to manage the number of EC2 automatically, Amazon provides auto-scaling service. This service allows auto adjust the number of EC2 instances in a group. The increasing and decreasing rules are defined by the user. For example, one group of EC2 node has three machines running a heavy task in 30 minutes. The metric collected indicate CPU is high, around 90% at that moment, which meets the condition of auto scaling configured by the user before. Auto scaling will increase the number of EC2 instances based on the demand of workload. Later when the work was done, it will auto decrease the number of instances. With auto scaling, to build a scalable application is simplified and more cost efficient because the time using on EC2 instances is optimized on the demand. [24.]

Amazon Simple Queue Message

Amazon Simple Queue message (SQS) is used to manage the tasks which are waiting for processing. SQS is well suited to build the distributed application which processes a lot of tasks, but not necessary in real-time. For example, in video processing, a heavy work may take a while to process. Not all videos are needed to be processed at the same time, however, they can be processed in an order. Thus, the order of video is saved as a message and it is managed by SQS. SQS allows sending, deleting, and handle messages from delivery to targets (EC2 machine). The advantage of SQS is simple configuration, high availability and reliable. [25.]

4 Video processing

4.1 Codec and format

In the process of recording, because the raw video data takes up a lot of storage space, it needs to be compressed. The specific devices or software makes the compression by some algorithms called codec. The process of compression is also known as encoding and the process of decompression is called decoding. [30.] This is necessary to reduce the size of video files, which help easier to transmit and easier for processing later. There are plenty of different codecs which can be found in the recording device, operating system, software player, CD, DVD etc. There are some popular codecs:

- x264: A free library under GNU GPL-licensed for encoding video streams into the H.264/MPEG-4 AVC compression format. [26]
- Xvid: open source under GPL, support MPEG-4 ASP (advanced simple profile) format [27]
- FFmpeg (libavcodec): a library containing decoders and encoders for audio/video codecs, support many formats such as MPEG-1, MPEG-2, MPEG-4 ASP, H.261, H.263, VC-3, WMV7, WMV8, MJPEG. [28]
- Divx: software product of DivX Inc, support MPEG-4 ASP, H.264 format. [29]

Raw video data normally has a very large size, after being compressed by codecs, it will be of significantly smaller size. However, the image quality might be reduced. With the tools and algorithms, a video can be converted by many different codecs.

Video format, on the other hand, is a type of container that holds one or more codecs of video or audio. This container has information about the video, audio, and a data track. There are popular formats for video such as MP4, AVI, and WMV. [30.]

H264 Codec

H264 is a video codec which has a high definition quality and efficient compression. H264 can be contained in many types of formats like mp4 or mov. Moreover, it is also supported by many platforms and browsers. We can see H264 is used in many recording devices like a camera and mobile devices. With the great quality and compatibility, H264 is one of the most popular codecs used for video. [30.]

4.2 FFmpeg

FFmpeg is a powerful framework for handling video and audio. FFmpeg includes multiple libraries like libavcodec, libavformat, libavfilter likely encode, decode, mux, stream, play media content. FFmpeg provided a simple command line interface to access functionalities. It can be able to compile on many different platforms such as Linux, Windows, Mac OSX, Android, and iOS. FFmpeg is suitable for many free as well as commercial projects. This software is open source under GNU Public License. [31.]

Transcoding

On the market, the production of many recorder and player devices gives us more choice in the video quality as well as format. However, there are also difficulties with incompatibilities when a video is recorded by a device, and played on many different platforms. To solve this problem, the video needs to be converted into many different formats to be compatible with different players. Video transcoding is a process to convert one video file format to another file format. Transcoding process can be expressed in figure 7:

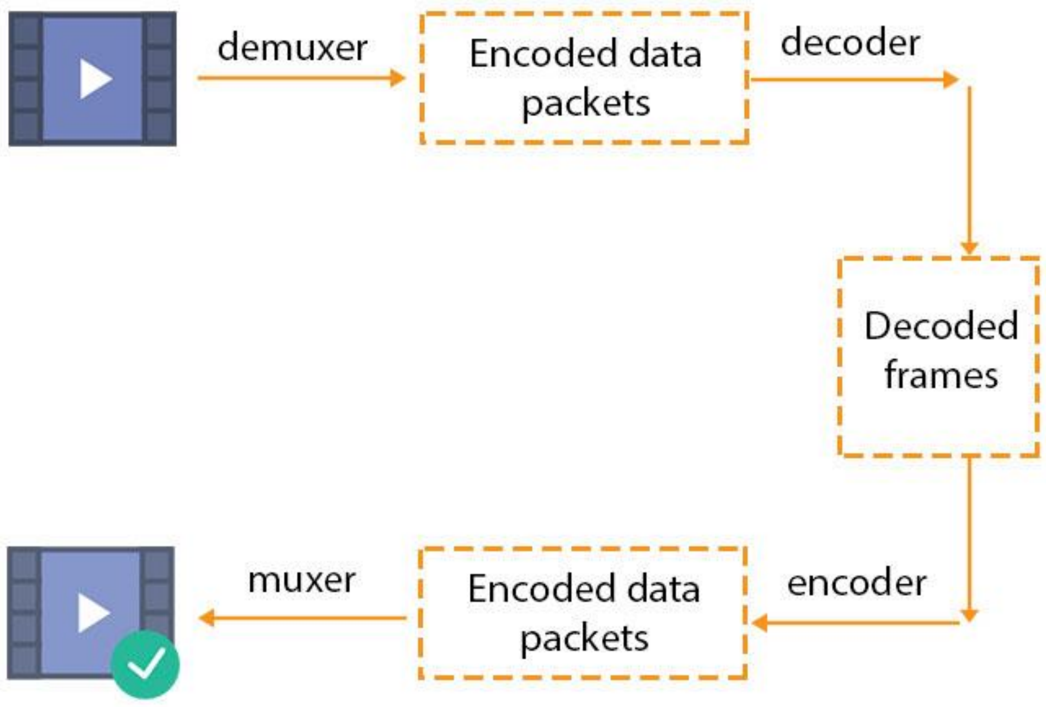


Figure 7: Video transcoding, data gathered from FFmpeg [28]

For example, Avi format can be converted to mp4 H.264 format, which is playable on the web platform.

```
ffmpeg -i input.avi -vcodec libx264 output.mp4
```

Listing 1: FFmpeg example command [28]

For the command line in listing 1, “ffmpeg” stands for the start of any command in this platform. “-i” is flag indicate input, and “input.avi”, “output.mp4” are input and output file path correspondingly. The extension tells FFmpeg what the format should be converted to, and libx264 tell the codec using to encode is x264.
Not only can FFmpeg transcode video, but it can also work with audio. The same idea is applied and easy to find in the documentation.

Manipulating video and audio data

With FFmpeg, image processing, such as changing resolution, scaling, filtering is simple:
For example:

To set output resolution, there is an option: "-s" flag as in listing 2:

```
ffmpeg -i input.avi -s 640x480 output.mp4
```

Listing 2: FFmpeg change resolution command

640 is the number of pixel for the width and 480 is the number of pixel for the height. To set video bitrate to 500k and audio bitrate to 64k as in listing 3:

```
ffmpeg -i input.mp4 -b:v 500k -b:a 64k output.mp4
```

Listing 3: FFmpeg changes the bitrate command

Flag ‘-b’ stands for bit rate ‘v’ and ‘a’ indicate video and audio.
To filter or scaling a video, flow the listing 4:

```
ffmpeg -i input.avi -vf scale=320:240 fade=in:0:30 output.avi
```

Listing 4: FFmpeg filter command

Flag "-vf" indicates video filters, there are many options for filtering, as can be seen in the example, scale video to 320 x 240 width and height, then create a fade effect in the first 30 frames of video.
For audio processing, there are similar options. In addition, video and audio stream can be divided and handled separately.

```
ffmpeg -i input.mkv -map 0:1 -map 0:2 audios_only.mkv -map 0:0
video_only.mkv
```

Listing 5: FFmpeg mapping [32]

As in the listing 5, flags "-map 0:1" and "-map 0:2" will map the audio stream in channel one and two, and then write to one single output audios_only.mkv file. Similarly, Flag "-map 0:0" indicates video stream and can be written to video_only.mkv file.

Concatenating

There is another popular need in video processing, namely concatenating files. Fortunately, FFmpeg provides several ways to do that.

Case one: when all video files have the same codec properties (codec, format, bitrate), there is a fast solution by using demuxing to merge the files:
Firstly, create a text file contain the list of the file path.

```
file '/path/to/file1.mp4'
file '/path/to/file2.mp4'
file '/path/to/file3.mp4'
```

Second, run the command to concatenate the files in the list:

```
ffmpeg -f concat -i videoList.txt -c copy output.mp4
```

Listing 6: FFmpeg concat demux command [33]

Listing 6 is the command to merge all the files by copying buffer data, without re-decoding and re-encoding, which means, the whole processing should be fast.

Case two: having different video files in different type of codec and format. It is impossible to merge those files without decoding and encoding, which means the copying is not allowed. FFmpeg gives the “filter_complex” options to solve this problem:

```
ffmpeg -i input1.mkv -i input2.avi \
-filter_complex "[0:v:0] [0:a:0] [1:v:0] [1:a:0] con-
cat=n=2:v=1:a=1 [v] [a]" \
-map "[v]" -map "[a]" -c:v libx264 output.mp4
```

Listing 7: FFmpeg filter complex command [33]

In the command of listing 7, [0:v:0][0:a:0] is the video and audio stream of input1 and [1:v:0][1:a:0] is the video and audio stream of input2.
The “concat=n=2” tells the number of input files is 2, then v=1:a=1 indicates there is only one video and audio output stream which is [v][a] in this example.
-map "[v]" -map "[a]" -c:v libx264 output.mp4

Finally, [v] and [a] stream will be written to the output by encoding with x264 codec. There are still many features of FFmpeg. However, in this research criteria, only the main features above are mentioned.

5 Case study

5.1 Introduction

As mentioned in the introduction, EdVisto is a product for education with the idea of learning through video story. The application allows users to create stories by combining multiple media files like videos, photos, and audios. The special requirement of the application is that all job will be processed by the server, users do not need to install software or to have a good hardware device to handle heavy workloads. Once media files are uploaded, the service will ensure they are stored securely and can be accessed anytime, anywhere. Therefore, in the same story, the users may use multiple devices to edit. Moreover, many users can join to edit a story at the same time. However, to obtain those features, the application is also encountered many difficulties. The main part and also the most challenging part of the project is video processing. The requirement is having all features yet ensure performance, reliability, and scalability.

5.2 Analysing problems

Main problems

The application can be divided into three specific issues. First, the video files uploaded can have different formats. Not only format, the video parameters like resolution, bitrate, and size ratio are also heterogeneous. This may result that the video cannot play on some platforms and is difficult to handle later. Therefore, the uploaded videos need to be converted into a common format and the same parameters.

Second, users can upload video files which have a big file size. To handle the large files will consume a lot of time. Practically, on the same computer configuration: Core i5 2410M CPU, 2.3 core duo. 8 GB ram, SSD storage, to convert 10 minutes of video takes about 5 to 8 minutes. Long processing time will affect the user experience. To achieve faster-processing performance, it can be applied by increasing the server's configuration.

For example, upgrading the CPU, memory, storage, which is called vertical scaling. However, this would be very expensive to have to invest in high-end hardware. In addition, it is not always needed to handle the heavy video. In different times of day, there are different amounts of users and the number of users at peak, and there is also sometimes very little the traffic is behind it. Clearly, the sustaining capital cost is unnecessary. Furthermore, when the demand exceeds the limits of the hardware upgrade, it will face a bottleneck, which cannot continue to scale up. Therefore, it needs to find a different solution which is more efficient and cheaper.

The last issue is concurrent users. A powerful server can serve a small number of users over a period. However, when the number of users at a time is too much, problems might happen. The server might be overloaded, users might have to wait longer for processing, and even if a problem occurs, the data might be lost. Therefore, it needs a system which can handle a huge concurrent request and scale up automatically.

Analyze video file format issue

With the first issue, to handle multiple videos with different format and codec, those videos need to be converted into one format. At this current time, on the web as well as the mobile environment, there are many video file formats. We need to choose a universal standard which is compatible with multiple platforms. Below is the table of media source supported by browsers and mobile.

Table 1: Browser compatibility with different media sources. Modified from Mozilla [34]

Video codecs / browsers	Chrome	Firefox	IE	Opera	Safari
VP8 and Vorbis in WebM	6.0	4.0	9.0	10.6	3.1
VP9 and Opus in WebM	29.0	28.0	N/A	yes	N/A
Streaming WebM via MSE	N/A	42.0	N/A	N/A	N/A
Theora and Vorbis in Ogg	yes	3.5	no	10.5	yes
H.264 and MP3 in MP4	yes	yes	9.0	yes	yes
H.264 and AAC in MP4	yes	yes	9.0	yes	3.1

Table 2: Mobile compatibility with different media sources. Modified from Mozilla [34]

Video codecs / browsers	Android	Firefox Mobile	Firefox OS	IE mobile	Opera mobile	Opera Mini	Safari	Chrome
VP8 and Vorbis in WebM	2.3	24.0	1.0.1	no	16.0	yes	no	29.0
VP9 and Opus in WebM	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Streaming WebM via MSE	N/A	42.0	N/A	N/A	N/A	N/A	N/A	N/A
Theora and Vorbis in Ogg	no	24.0	1.0.1	no	no	yes	no	no
H.264 and MP3 in MP4	yes	24.0	yes	10.0	16.0	yes	yes	29.0
H.264 and AAC in MP4	yes	24.0	yes	10.0	16.0	yes	3.2	29.0

Based on table 1 and 2, we can see the video files have MP4 format encoded with H264 video codec and AAC or MP3 audio codec compatible with all browsers and mobile platforms.

When it comes to audio, both AAC (Advanced Audio Coding) and MP3 (MPEG-1 Audio Layer 3) are the suitable format, however, using AAC brings more advantages. Flowing some benchmarks, when an audio file is compressed to the same bitrate, AAC file has a smaller size than MP3 file as shown in table 3. More than that, the audio quality of AAC format is also better than MP3 format at the same bitrate [36].

Table 3: AAC vs MP3 file size in different bitrate. Modified from Winxdvd [35].

Format / Bitrate	96Kbps	128Kbps	192Kbps	256Kbps
MP3	4.24M	5.65M	8.48M	11.3M
AAC	4.29M	5.71	6.76M	6.76M

In summary, a video file with H264 video codec and AAC audio codec in MP4 container is a suitable choice.

Another issue is the resolution and bitrate of the video. The video has higher resolution and bitrate will have bigger file size and vice versa. The idea is to find a medium number which ensures both the quality of video as well as the performance of the system. For the requirement, resolution 640x360p and 1280x720p have been selected as standard common resolution which is used widely. Reasonable bitrate will be the one ensure as well as the quality and the file size of output file. In this study, the bitrate 500k for 360p resolution and 1000k for 720p resolution are selected.

Analyze performance issue

Performance problems will appear when a video file to be handled has bigger size or duration. With vertical upgrading method for one computer, the hardware will be soon restricted due to technical limitation. Moreover, even with a high-end hardware, the way to handle a video file is reading segment by segment respectively. If a single server instead of multiple servers are handled the same video at the same time, work will be completed faster. The combined power of multiple computers to handle a task is known as parallel processing.

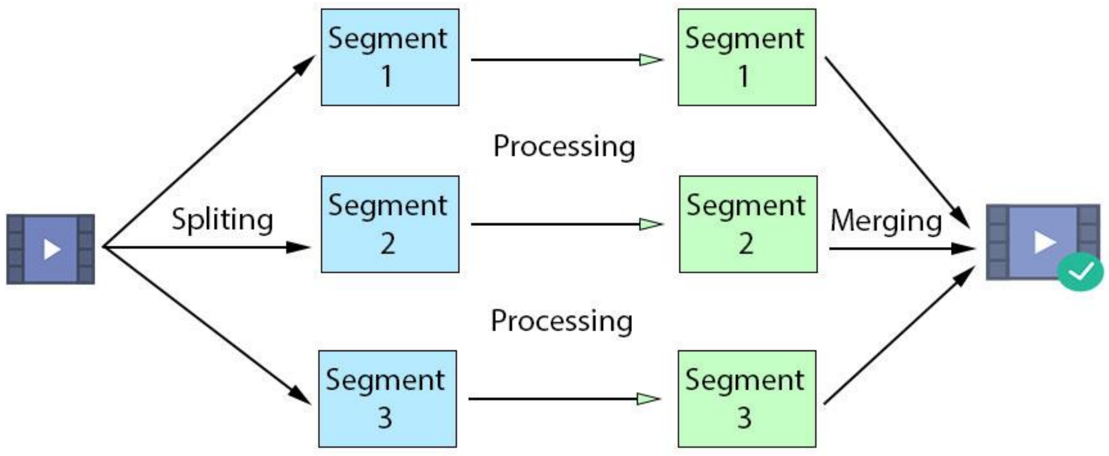


Figure 8: Parallel processing

As already mentioned in this study, parallel processing is part of the distributed system. In figure 8, a video file can be considered as one job, if we split video file into multiple smaller segments, we can send those segments to multiple machine nodes for processing independently. After all the nodes finished, smaller segments will be merged into one big file which follow the original order. This approach will bring the following benefits. First, the job is processed simultaneously, the video segment does not need to wait for the previous one to finish. Second, to gain more power of processing, just simply increase the number of nodes in the system. That is, instead of having to use an expensive computer, we can use less expensive computers, and increase the number of computer nodes depend on the needs. That would save a huge amount of operating costs. Also, the system is not interrupted when the problem occurs in a single machine.

Analyze concurrency issue

Even having the network of computers for a high-performance system, it still has another issue is concurrence. Practically, whether the processing capacity of the machine nodes is very high, it can still cause overload if too many requests are sent at the same time. To serve all requests at once is difficult, it is better if we can save the requests into a queue.

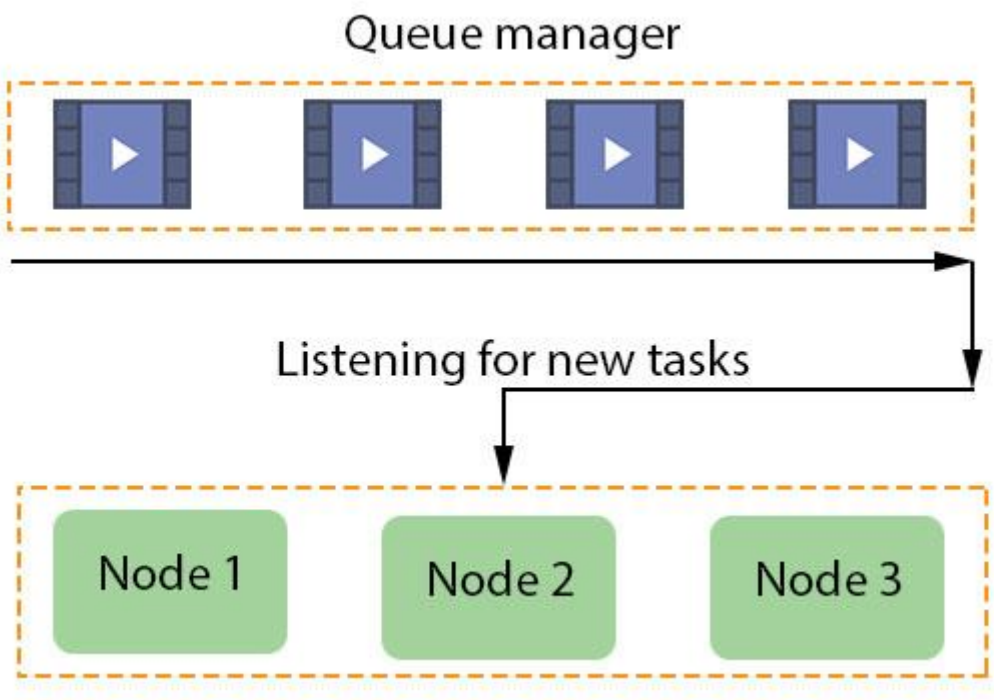


Figure 9: Queue system

Demonstrated in figure 9, each request will be processed in the order respectively and the waiting time will depend on the processing capacity of the machine nodes. If the number machine node is smaller than the demand, the request must wait longer, and vice versa, if the number of nodes is much higher than the needs, the cost will be a problem. Therefore, another question arises, how to adjust the number of machine nodes for matching the processing demand.

Basically, the server manager cannot monitor and adjust the number of machine nodes manually. Figure 10 illustrates an automatic tracking system to adjust the number of nodes in the network.

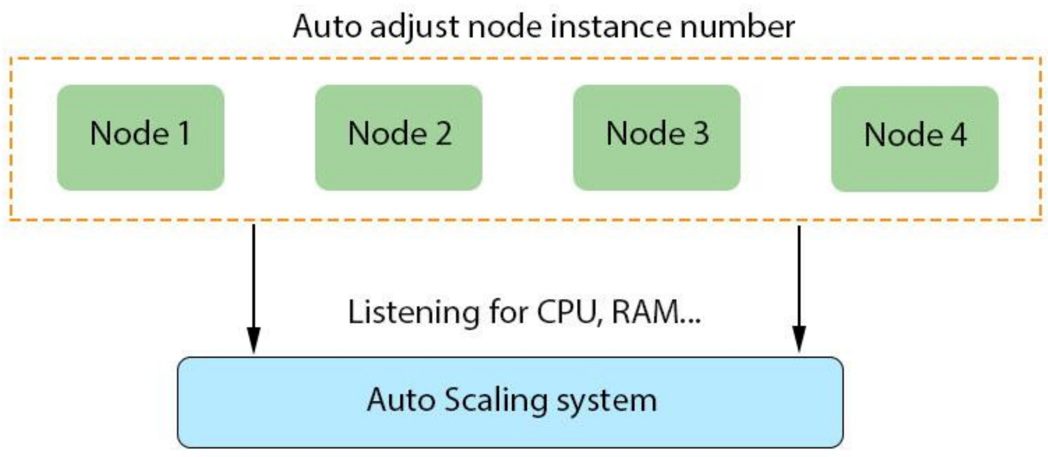


Figure 10: Auto-scaling system

The system monitors CPU, memory, and availability of each node. If the CPU or RAM is being used at a high percentage, the system can self-deploy additional machines, and vice versa, if the parameters are too low, the system will automatically turn off the machine to save costs.

5.3 Solution and Architecture

After analyzing the problem, the distributed system is a suitable solution. However, since it is difficult to build a system from scratch, so we can use cloud service from the third party for development. There are many cloud service providers today, one of which is Amazon. In this study, the distributed system will be designed based on the Amazon cloud infrastructure. As discussed earlier, there are services like Amazon EC2, S3, Load Balancing, SQS and Auto Scaling Group, and each service takes on the role in the system.

Master node design

First, we need a connection between the server and the client via the TCP protocol. Master node will take over this task for creating API. Master node is deployed on an EC2 instance which is already installed in the operating system, software, and necessary services. A web server and database are also set up on the master node.

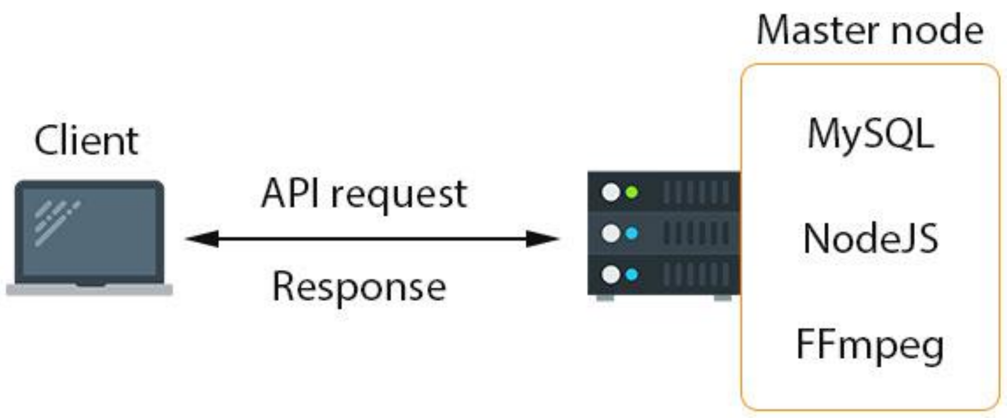


Figure 11: Master node

As in figure 11, the client sends requests to the server via the provided API, which is created by NodeJS. Data will be stored in MySQL. Node JS is responsible for communicating to the database to access and save data. After data is manipulated, it will be sent back to the client in JSON format.
FFmpeg is a video processing library, we will discuss it in detail in the following sections.

Slave node and Load Balancer design

After the client sends a request to the master server node, the video data will be pre-handled in the master node, after that, it will be divided into smaller parts then transferred to slave nodes to handle intensive tasks. In a slave node, no database is required. Instead, only the micro service is running to handle special task which in this case is the video processing. Figure 12 presents the master node and slave nodes relationship.

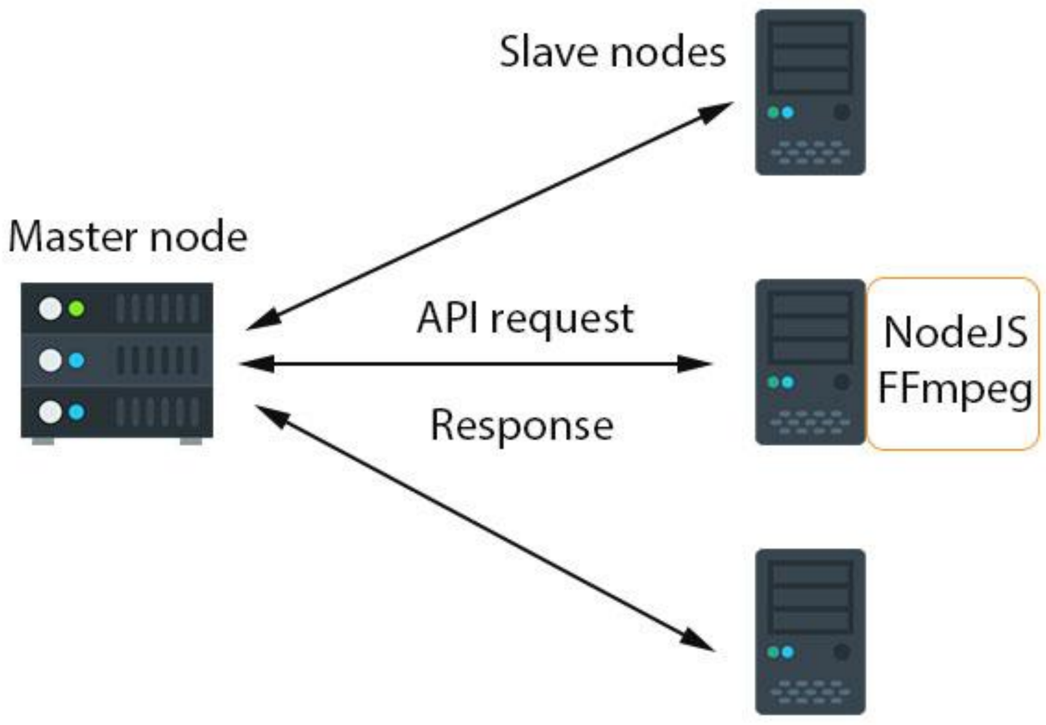


Figure 12: Slave nodes and master node

A master node can connect to many different slave nodes through the TCP/IP protocol. But in the Amazon cloud service, master node instance does not know the IP address of the slave nodes, especially if the slave nodes are deployed automatically. Fortunately, AWS provides a service called load balancer, which automatic delivery requests from the one node to others in the network.

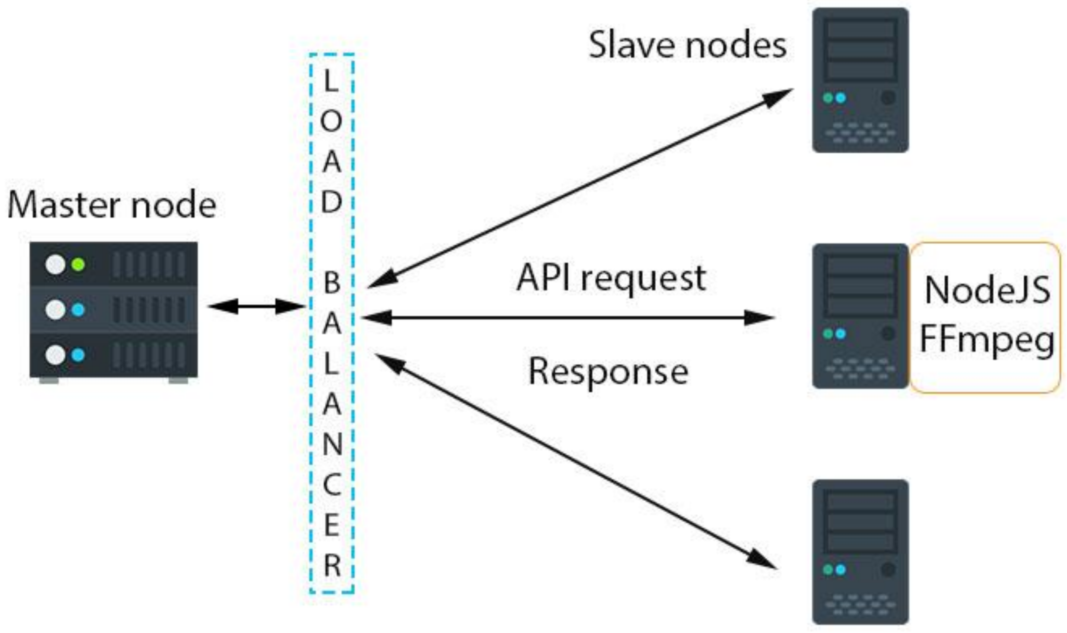


Figure 13: Load balancer for master node and slave nodes

As the figure 13 shows, the load balancer will decide which slave node should receive the request coming from the master node. The load balancer will keep track if a slave node is added or removed from the network.

Storage design

Information for processing will be saved in the MySQL database. However, video files saved should not be in the master node due to the limitation of storage capacity. To solve this problem, Amazon offers a storage solution called Amazon Simple Storage Service (S3). As introduced, Amazon S3 is a distributed storage system which can storage unlimited files as well as no restriction about capacity. S3 is well suited for storing large files, especially media files. Figure 14 shows how the S3 be accessed by machine nodes.

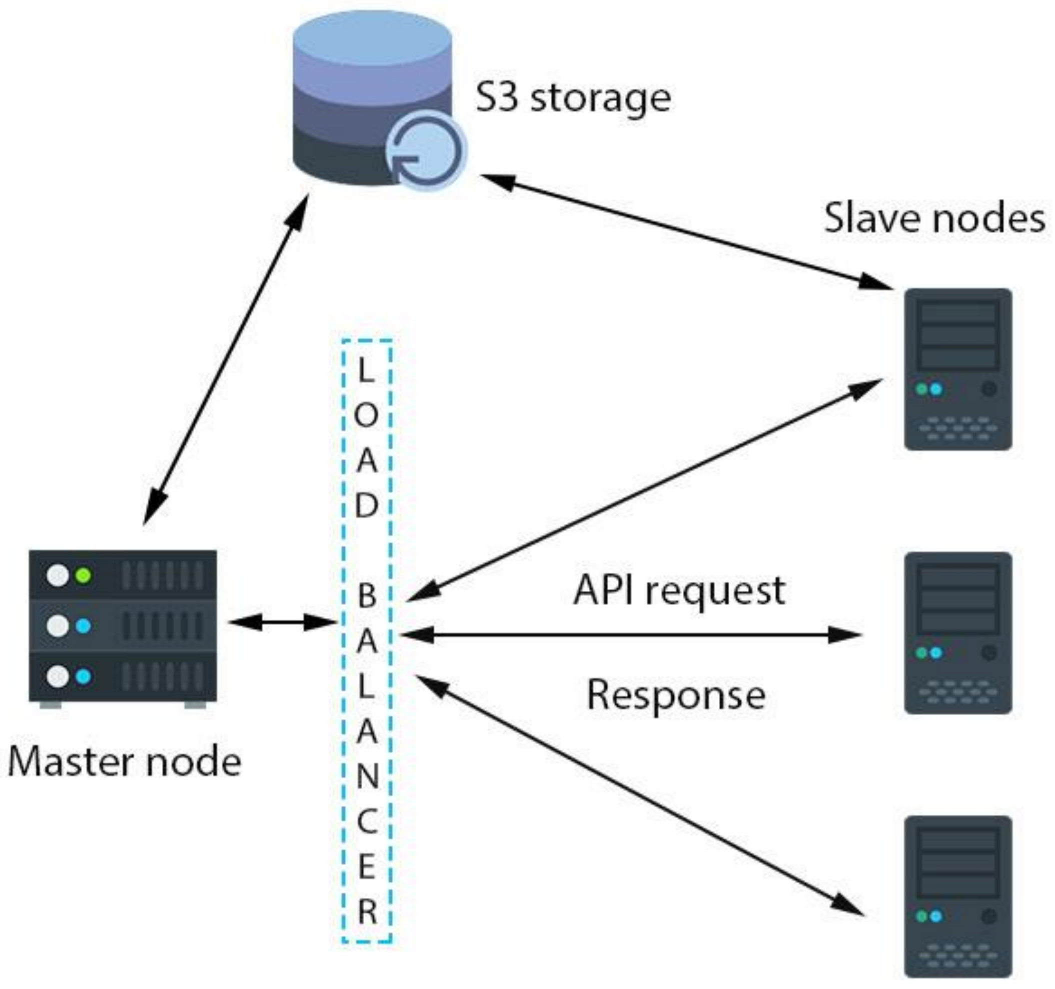


Figure 14: S3 storage, master node, slave nodes

With amazon S3, master and slave nodes both can access same data files easily. The nodes do not need to worry about managing files on its own storage, just focus on processing tasks.

When a video file is sent to the master node, it will be split into smaller segments by FFmpeg. Then, these files will be uploaded to S3. After ensuring all files have been uploaded successfully, the master node sends the requests to slave node which asks for processing the video segments. When the slave node completes its tasks, one message will be sent to the master. Master keep tracks of the work. When all the tasks have been done, the master will ask one of the slave nodes to merge all processed segments into one video file in a correct order. In finished, the files will be uploaded back to S3 and a successful notice will be sent to the client.

Queue system design

Master node can send a direct request to slave nodes through the load balancer. However, slaves must handle the request immediately. If there are too many requests, especially with video processing, the slave might get overloading. It would be better if all the request is saved in a queue and waiting for processing the turn. This can be done with Amazon SQS and presented by figure 15.

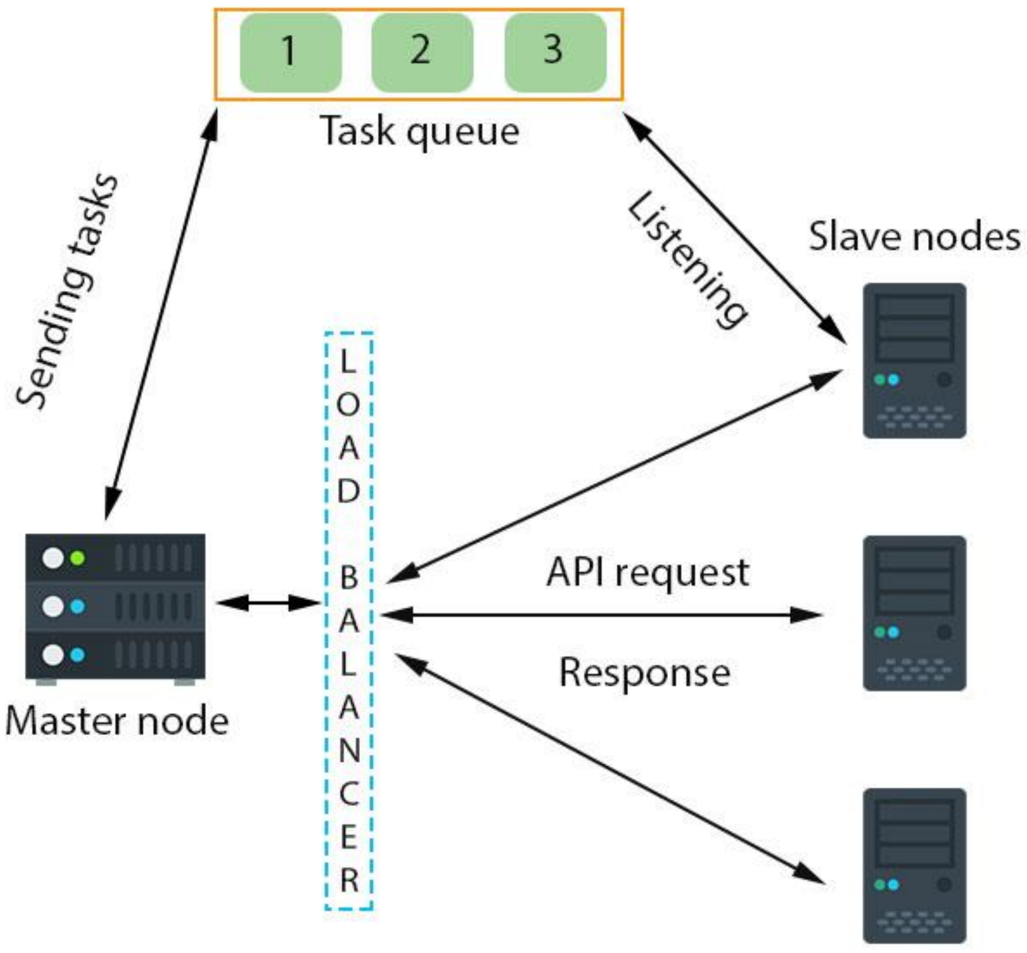


Figure 15: SQS, master node, slave nodes

Instead of sending a direct request from the master to the load balancer, the request is sent to the SQS in a form of message. SQS is responsible for managing messages, setup expired time and waiting time. Slave nodes are using pooling method to listen to SQS. If there is a new message and the slave node is capable of the handle, it will automatically retrieve messages from SQS. Meanwhile, SQS will temporarily hold the message until it receives confirmation from the slave to delete. Otherwise, if the waiting time ends, SQS will release the message for another slave node. Thus, the entire work is guaranteed if there is a problem with a slave node.

Auto scaling system design

To solve the problem of managing the slave nodes, Amazon provides Auto Scaling Group service. To be able to use this service, it needs the system image generated from a slave node. With this image, any newly launched EC2 instance will have full-service configuration and ready to run. It is similar to clone a slave to a new one. So, no need to setup again for new launched slave instance.

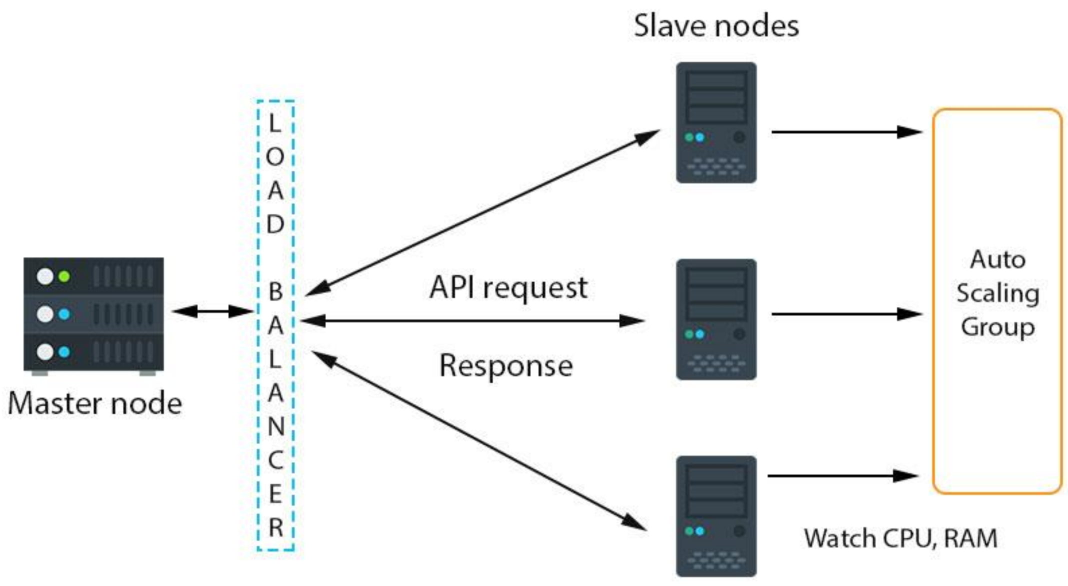


Figure 16: Auto scaling group, master node, slave nodes

As the figure 16 shows, Auto scaling service will listen to slave nodes status, for example (CPU, Memory, Storage driver). The establishment of the rule to launch or terminate a slave is simple. After launched a new instance, load balancer automatically recognizes and registers new node into the network. And similarly, when an instance is terminated, the load balancer will automatically remove it from service.

System architecture

Combine all the above, we have a complete architecture for the whole system presented in figure 17.

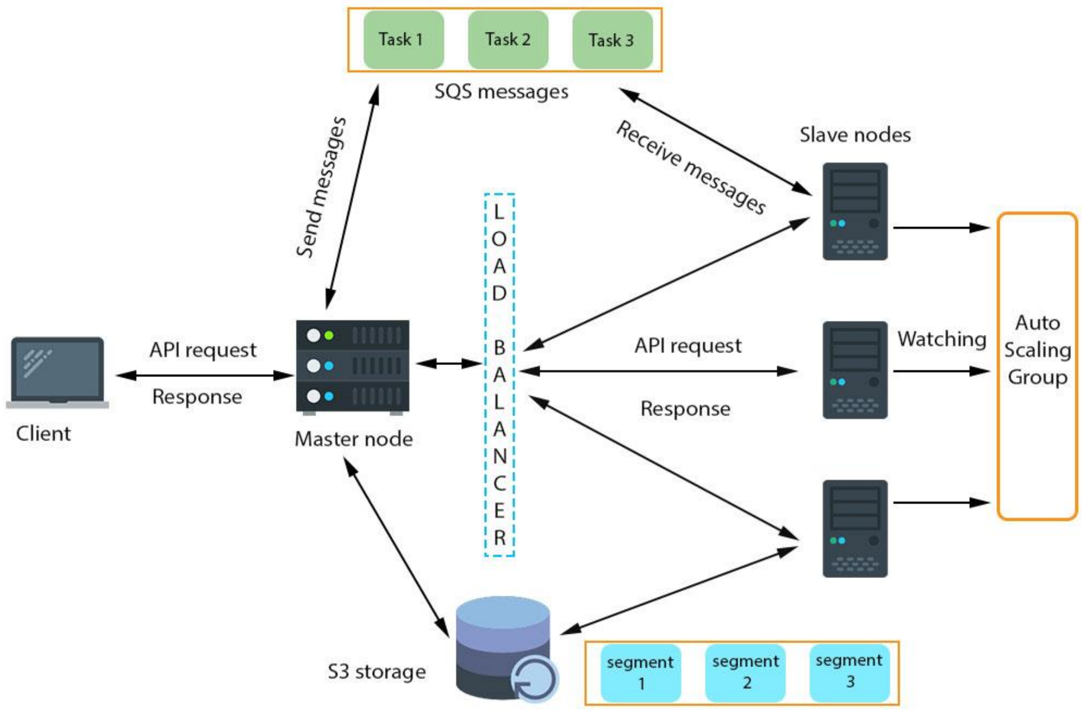


Figure 17: System architecture

Here is the work follows a request, included file uploading, has been sent from client to server until processing progress done and send back to the client a message event.

- Client upload a video file to server (master node)
- Master node receive the file, divide the file into smaller parts.
- Information about video files and its part saved to MySQL database.
- Master node upload file's parts to amazon S3.
- Master node create message contain necessary info, then send to SQS
- Slave node listens for a new message from SQS. When a message is taken SQS hold the message to prevent another node take it.
- By the message, video part is downloaded from S3 to the slave node.
- Slave node process video based on the requirement from message info.
- After process done, the slave sends a message to master, announce the work done.

- Master node keep track of the works, if all works have been done, the master will send a direct message to slave through the load balancer.
- Slave node receives the message from the master, start merging all the video part into one file then upload back to amazon S3. After that, it sends a success message to the master node.
- Master node save all the necessary information to the database, then emit an announcement to the client by real-time WebSocket.
- The client receives the socket event, display video to the view.

5.4 Implementation

Required tools

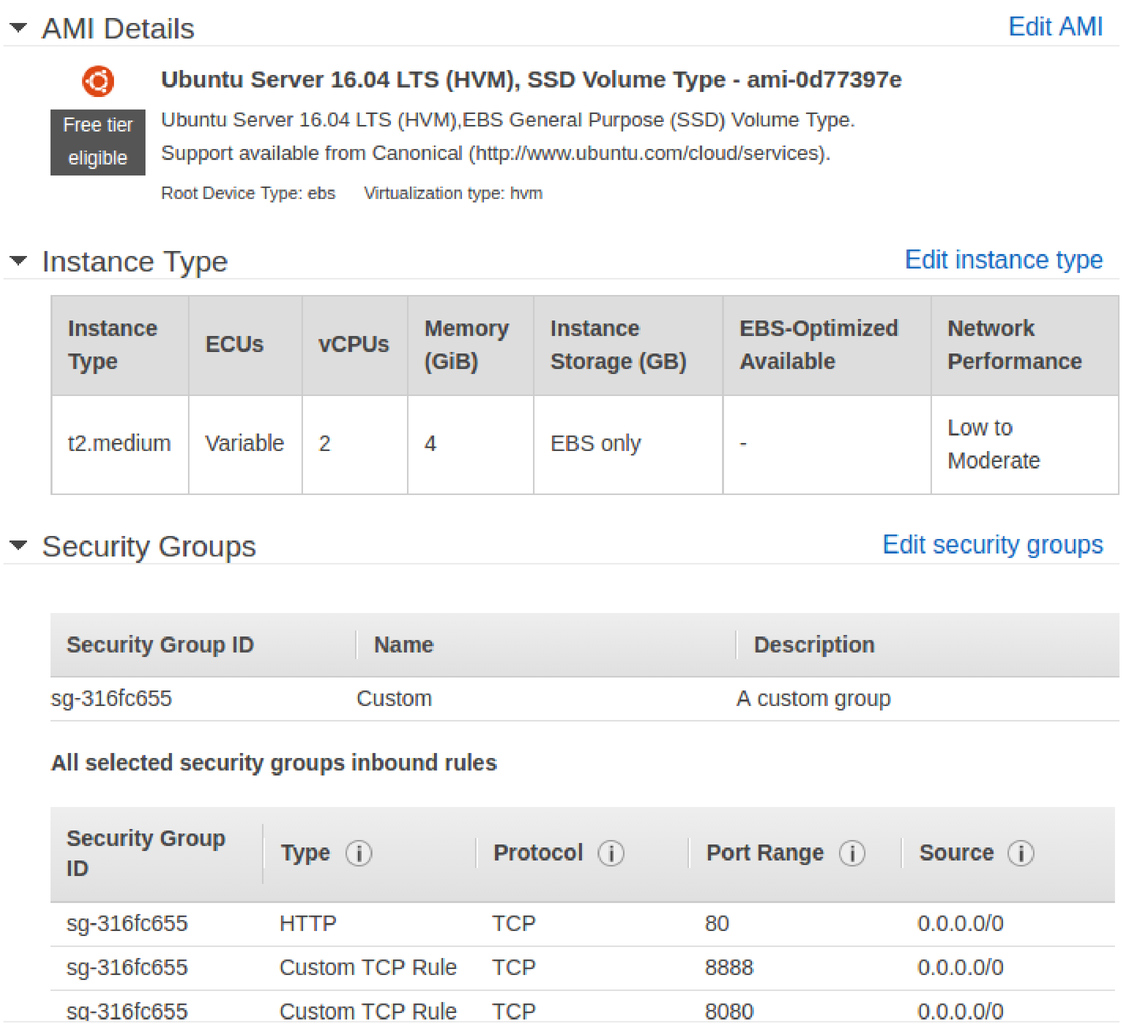
Before implements application, we need to setup development environment. In addition, the following tools were used in this project:

- Laptop running Linux mint 18
- WebStorm IDE, a code editor with lot of utility tools from Jet Brains
- Version control GIT, with Bitbucket as host repository
- MySQL workbench for database management

Setup Master node

Master node instance is running Linux operating system. It is simple to launch an EC2 instance by AWS management console, select the “EC2 instance” in the menu and launch a new one. There are many choices for the operating system. This project will use Ubuntu Server LTS 64-bits 16:04 for all EC2 instances.

There are many options for EC2 instance, however, the master node without consuming too high computational capability, it can be started with T2 medium type which has 2 cores of CPU and 4GB memory.
With optional storage, SSD drives would be more appropriate for I/O tasks due to faster access than HDD.
Another section needs to be considered is security rule. To be able to send request from a client to server, HTTP/TCP request in port 80 must be opened as showed in figure 18. Moreover, some other TCP port and SSH port 22 can also be added depending on the design of application port.


AMI Details

Edit AMI

Free tier eligible

Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-0d77397e

Ubuntu Server 16.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root Device Type: ebs Virtualization type: hvm

Instance Type

Edit instance type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.medium	Variable	2	4	EBS only	-	Low to Moderate

Security Groups

Edit security groups

Security Group ID	Name	Description
sg-316fc655	Custom	A custom group

All selected security groups inbound rules

Security Group ID	Type	Protocol	Port Range	Source
sg-316fc655	HTTP	TCP	80	0.0.0.0/0
sg-316fc655	Custom TCP Rule	TCP	8888	0.0.0.0/0
sg-316fc655	Custom TCP Rule	TCP	8080	0.0.0.0/0

Figure 18: Amazon EC2 instance

After launching a new EC2 instance successfully, we can access to EC2 command line interface via SSH. For security reasons, Amazon requires a security key pair when connecting, which can be downloaded from AWS management console. Open terminal, typing SSH command with input is a key file, the username is “ubuntu” and the IP address of the EC2 instance. The listing 8 shows the SSH command.

```
ssh -i key.pem ubuntu@56.41.30.278 -y
```

Listing 8: Connecting to EC2 by SSH

After successfully connect, a command line interface window will appear. In here, we can use the command to install, config and run the service.

The necessary software is installed in the master node includes:

- Node JS, official website: https://nodejs.org
- MySQL, official website: https://www.mysql.com/
- FFmpeg, official website: https://ffmpeg.org/

The installation of the software will not be listed in detail. However, it can easily follow the instructions from the homepage of the open source.

API server

In a NodeJS application, MVC pattern can also be applied. Below is the app structure:

- App/
    - server/
        - config/
            - config.js (configuration file for run environment)
            - constant.js (contains constant variables)
            - db.js (configuration for database connection)
        - controllers/
            - videoCtrl.js (control logic for processing video)
        - models/
            - videoModel.js (store, access data model for controller)
        - route.js (routing for API requests)
        - socket.js (connect, manage WebSocket connection)
    - index.js (main app file)
    - package.json (node module package manager file)
    - node_modules/ (contain node modules package)

In node_modules, the following package is needed for the handling of the video:

- fluent-ffmpeg: create a command builder tools by JavaScript for FFmpeg
- multiparty: Parse HTTP requests with content-type multipart/form-data
- aws-sdk: help API connect to AWS
- node-mysql: is node.js driver for MySQL written in JavaScript
- request: for sending HTTP request between nodes

- socket.io: Real-time WebSocket library for NodeJS

After installing the necessary software, the first task is to set up the server. With Node.js and Express framework, http server can run with few lines of code as in listing 9:

```
var express = require('express');
var app = express();
app.listen(config.port);
```

Listing 9: Node server setup

The port for running application service is in a config file, which can be set manually by the developer. In production, the port can be 80 if running without a proxy server.

To manage the routing to the application, create a route.js file and navigate a request to the relevant controller to handle:

```
var express = require('express');
var router = express.Router();
var videoCtrl = require('./controllers/videoCtrl);
router.post('/uploadVideo', videoCtrl.upload);
```

Listing 10: Routing in NodeJS

In the example in listing 10, API for uploading file has the format:
http://api.domain.com/uploadVideo
The request will be passed from router to upload function in the videoCtrl.js file. We will talk more about the upload function in the next section.

Database

The database has the duty to save the video-related information and helps keep track progress of the video. Figure 19 shows the table video design:

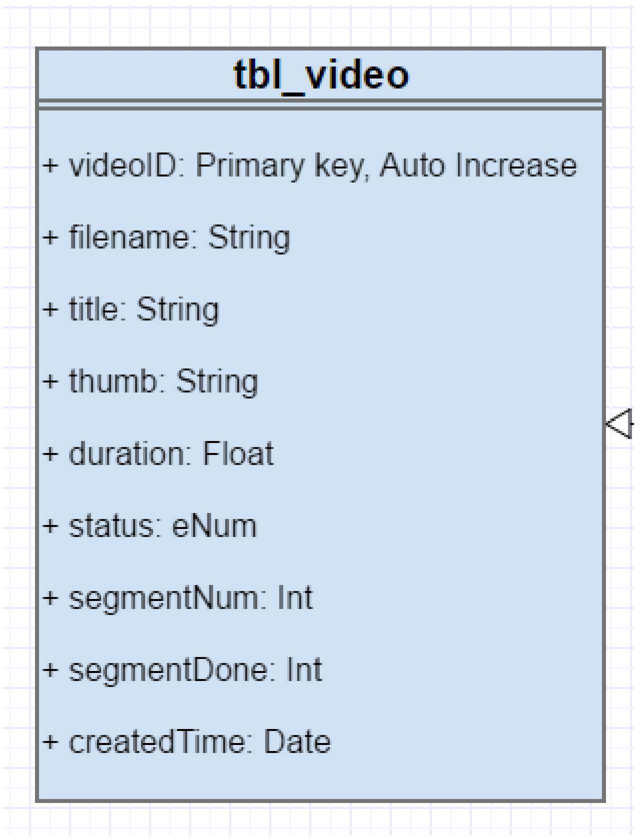


Figure 19: Database table video UML

The role of each column:

- videoID column is primary auto increase key
- filename column stores the unique name of processed video file
- title column stores the original name of the video file
- thumb column stores the filename of thumbnail photo of video
- duration column stores the duration info of the video
- status column indicates the status of progress, which is either pending, success or fail
- segmentNum column shows the number of total segment part created from video file.
- segmentDone column shows the number of processed segment successfully.
- createdTime column shows the date time file uploaded.

When a video file is divided into smaller segments by the master node, the number of the segment will be saved into the database. Some information like title, duration could also be saved with the status field is set to “pending”. At that time, the number of processed segment is 0. Every time, when a segment has finished by slave node, the master node will increase segmentDone by one. When all segments have been processed successfully, which means segmentDone equal to segmentNum, master node marks the status of video file to "success", also updates the file name and thumbnail fields. In case some error happens in progress, the master will mark the status of the video file is "fail".

Setup amazon S3

Amazon S3 will be a place to store video files. Also from the AWS management console, in S3 section, we create a storage pool called bucket. Bucket name can look like a subdomain name, which will have some benefit later when using amazon DNS.
After creating the bucket, we can set some security rules, if the file is public for the user. The reference security rule is as follows:

```
{
          "Sid": "AllowAccecssPublicFolder",
          "Effect": "Allow",
          "Principal": "*",
          "Action": "s3:GetObject",
          "Resource": "arn:aws:s3:::cdn.domain.com/*"
}
```

Listing 11: Configuration for access permission of S3

Above rule in listing 11 allows public folders GET by all users.
Or if we only allow access to a private user who can GET, PUT, DELETE, and COPY or have more permissions, there is an example in listing 12:

```
{
          "Sid": "AllowAccessWithSignedURL",
          "Effect": "Allow",
          "Principal": {
                    "AWS": "arn:aws:iam::8xxxxxxxxx:user/admin"
          },
          "Action": "*",
          "Resource": "arn:aws:s3:::cdn.domain.com/*"
}
```

Listing 12: Giving access for specific account

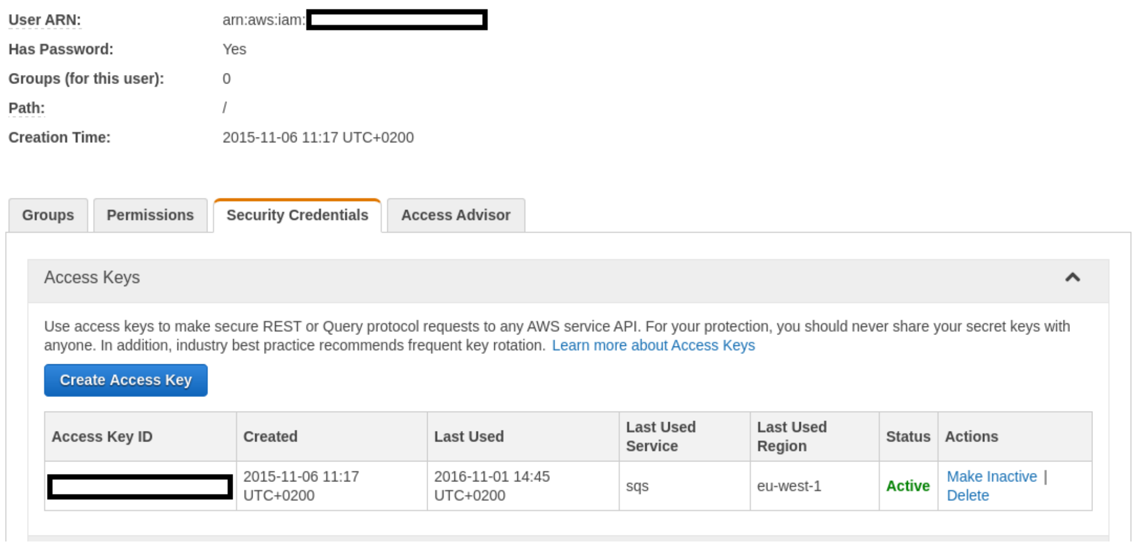
Another important part is the CORS configuration. Since the application will have a domain different to the domain of the Amazon S3, this configuration is necessary to allow cross-origin requests on the S3 HTTP server. In edit CORS configuration options, we can add:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-
01/">
   <CORSRule>
       <AllowedOrigin>*</AllowedOrigin>
       <AllowedMethod>GET</AllowedMethod>
       <AllowedHeader>*</AllowedHeader>
   </CORSRule>
</CORSConfiguration>
```

Listing 13: CORS configuration

The configuration in listing 13 allows all domain to access the files in S3.

Next is the connection between Amazon S3 and Node.js server. Amazon offers an SDK package for NodeJS to send AWS API requests, included API for S3.
For authentication, we need to add one new user account with the access key ID for SDK connection. Amazon will provide a secret access key, which must be saved carefully.



User ARN:	arn:aws:iam:[redacted]
Has Password:	Yes
Groups (for this user):	0
Path:	/
Creation Time:	2015-11-06 11:17 UTC+0200

Use access keys to make secure REST or Query protocol requests to any AWS service API. For your protection, you should never share your secret keys with anyone. In addition, industry best practice recommends frequent key rotation. Learn more about Access Keys

Access Key ID	Created	Last Used	Last Used Service	Last Used Region	Status	Actions
[redacted]	2015-11-06 11:17 UTC+0200	2016-11-01 14:45 UTC+0200	sqs	eu-west-1	Active	Make Inactive | Delete

Figure 20: Amazon access key

Figure 20 shows the place of Access Key ID. Once having the Access Key ID, we can set up the AWS in NodeJS as follows:

```
var AWS = require('aws-sdk');
AWS.config.update({
   accessKeyId: 'acces_key_id',
   secretAccessKey: 'secret_key_id',
   region : 'eu-west-1'
});
var bucket = "cdn.domain.com";
```

Listing 14: Amazon SDK configuration

Listing 14 shows how to config Amazon SDK in NodeJS.

After AWS configuration is done, to be able to upload, create an S3 instance, then using API provided in SDK documentation. For example, in listing 15, for listing all files are in S3 folder:

```
var s3 = new AWS.S3();
var param = {
    Bucket: bucket,    // bucket name
    Prefix: folder        // folder name
}
s3.listObjects(param, function(err, data) {
    if (!err) {
        console.log(data.Contents);
    }
});
```

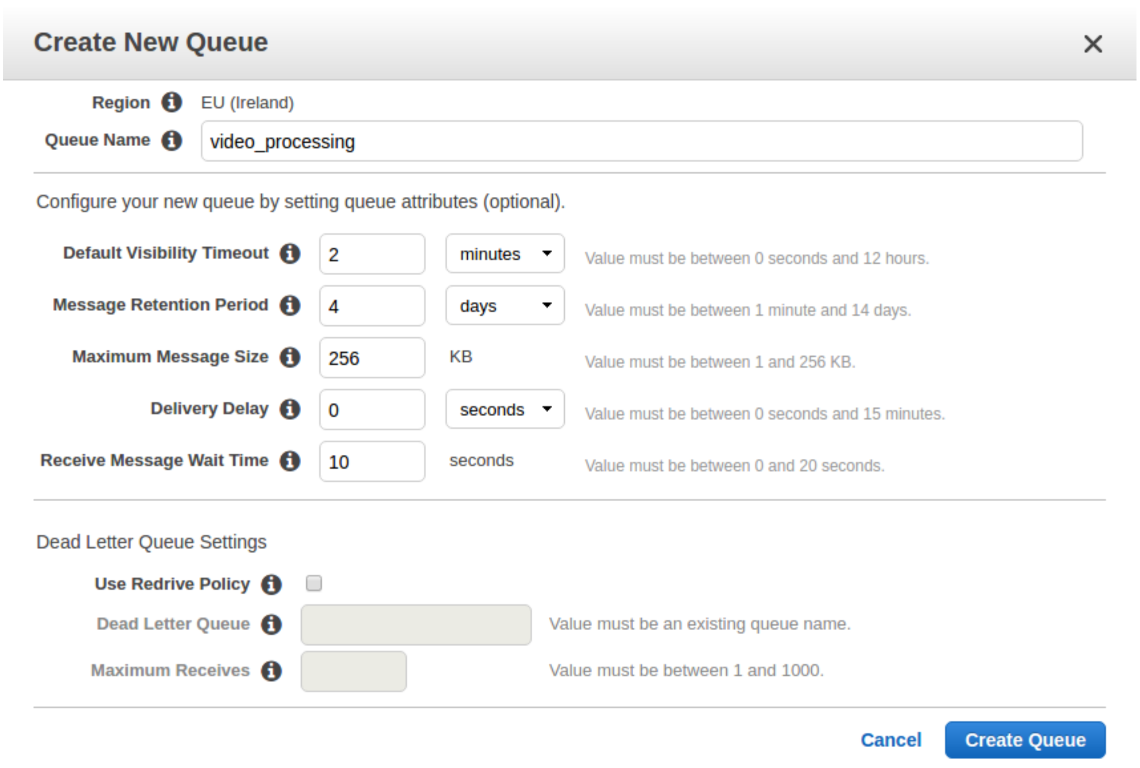
Listing 15: Amazon S3 request example.

The data return from S3 is an object has the “Contents” is an array of file object information.

Setup amazon SQS

To set up message queue service, in the management console, select SQS then create a new queue. After created a new queue, SQS will provide a link for connecting later. The link is in this format:

https://sqs.eu-west-1.amazonaws.com/xxxx/video_processing


Create New Queue

Region	EU (Ireland)		
Queue Name	video_processing		

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout	2	minutes	Value must be between 0 seconds and 12 hours.
Message Retention Period	4	days	Value must be between 1 minute and 14 days.
Maximum Message Size	256	KB	Value must be between 1 and 256 KB.
Delivery Delay	0	seconds	Value must be between 0 seconds and 15 minutes.
Receive Message Wait Time	10	seconds	Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy	☐	
Dead Letter Queue		Value must be an existing queue name.
Maximum Receives		Value must be between 1 and 1000.

Figure 21: Amazon SQS configuration

In the example shown in figure 21, the new queue has the name "video_processing". Default visibility timeout is the time when message received from a queue will be invisible to other listeners. Message retention period is the time message will be deleted if not received by any listener. Receive message wait time is the maximum amount of time for a long polling receive call to wait for a message to become available before returning an empty response.

To get a connection with node JS, similarly to S3, an instance of SQS created from AWS SDK.

```
var sqs = new AWS.SQS();
var params = {
    MessageBody: “Test message”,
    QueueUrl: SQS_URL,
    DelaySeconds: 0
};
sqs.sendMessage(params);
```
Listing 16: Create SQS message.

As showed in the listing 16, the message body is in string format. If the request data is in JSON format, we need to use JSON.stringify(message) to convert data object to a string.

Master node workflows

Combining the above implementations, we can present the master node workflow as follows:

1. File uploaded to master
2. Save video info to a new row in database
3. Split video files into segments
4. Upload segments to S3
5. Update the segment number to database
6. Create SQS message for each segment
7. Send a response to client
8. Delete temporary folder of the process

When the request passed to upload function, it will receive two arguments: req and res. The multiparty form will parse req parameter to get the files and fields data from the request. The file will be saved in a temporary folder and can be deleted after being processed. The most important step here is splitting the video file into smaller segments. We can use the ffmpeg-fluent module for NodeJS to create the command [appendix 1]. The command to split a video by FFmpeg:

```
ffmpeg -i input.mp4 -y -acodec copy -vcodec copy -f segment -
segment_time 60 -segment_list_size 0 -segment_list output/seg-
ment.ffcat output/segment%d.mp4
```

Listing 17: Splitting video file into segments

In the given command in listing 17, the input's video and audio codec are copied to output segments. Segment time is 60 seconds for each part of the video.
The segment.ffcat is segmented list file, which contains all the segments name in order. segment.ffcat file content will look like this:

```
ffconcat version 1.0
file segment0.mp4
file segment1.mp4
file segment2.mp4
```

Segment list size equal 0 to make sure the list file will contain all the segments. All the segments file output will be written to output folder, and have the format for example segment1.mp4, segment2.mp4.

After uploaded all split files to a temp folder in amazon S3. The message for queue can be created in this JSON format:

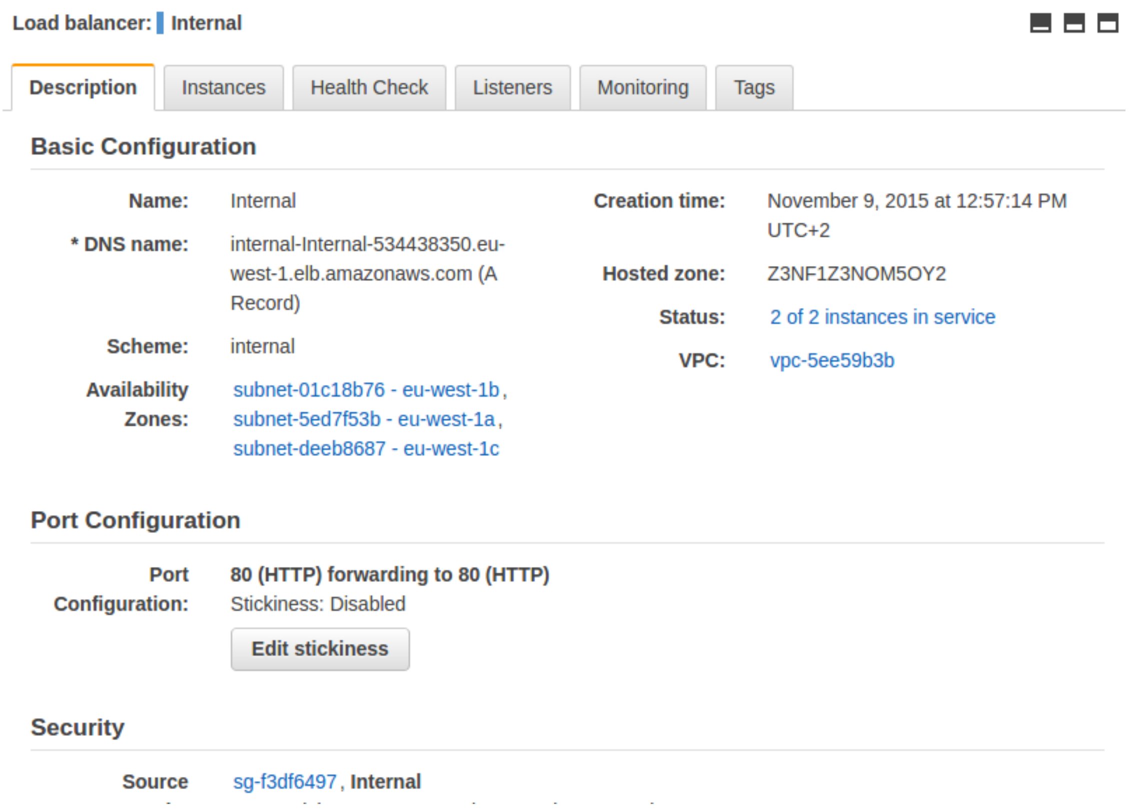
```
{
   type : 'transcoding',
   id : videoId,
   key : s3_url_to_segment_file,
   file : filename
}
```

Listing 18: SQS Message format

In the JSON object of listing 18, keyword type is to tell slave node the requirement of processing, in this example is transcoding. Keyword id is the id of video saved in the database, to identify later. The key is the amazon S3 link to the uploaded segment file. Now the list of the message will be saved on SQS and waiting for slave node receive.

Setup Load Balancer and Auto Scaling Group

The load balancer is a bridge between the master node and slave nodes. To setup a load balancer, go to the EC2 service, select load balancer section and create a new one in classic type. It is important to mark the “internal load balancer” option because of the communication between master and slave nodes only inside the network. All the requests are coming from outside of network should not have the permission to send to slave nodes. With the configuration in figure 22, it helps to secure the slave nodes from being attacked.



Basic Configuration

Name:	Internal	Creation time:	November 9, 2015 at 12:57:14 PM UTC+2
* DNS name:	internal-Internal-534438350.eu-west-1.elb.amazonaws.com (A Record)	Hosted zone:	Z3NF1Z3NOM5OY2
Scheme:	internal	Status:	2 of 2 instances in service
Availability Zones:	subnet-01c18b76 - eu-west-1b, subnet-5ed7f53b - eu-west-1a, subnet-deeb8687 - eu-west-1c	VPC:	vpc-5ee59b3b

Port Configuration

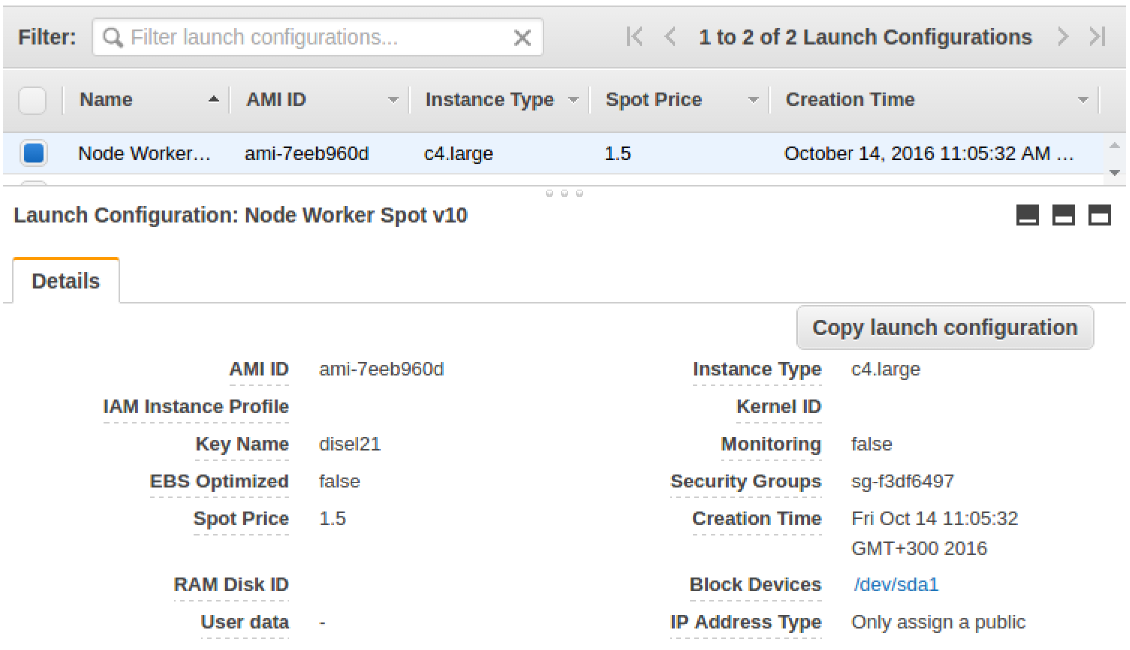
Port Configuration:	80 (HTTP) forwarding to 80 (HTTP) Stickiness: Disabled

Security

Source	sg-f3df6497 , Internal

Figure 22: Amazon Load Balancer configuration

After setup load balancer, there is still no slave node attached to it yet. To have a slave node launched automatically, we need to set up auto scaling group. In the same place with a load balancer, there is auto scaling section. Before creating a new auto scaling group, there is Launch Configuration needed to be set up first. Setting up Launch Configuration can be understood as having an existing image file for the slave node, preselect the type of EC2 with a necessary configuration such as security, storage. The configuration is shown in figure 23.



Filter: Filter launch configurations...

1 to 2 of 2 Launch Configurations

	Name	AMI ID	Instance Type	Spot Price	Creation Time
■	Node Worker…	ami-7eeb960d	c4.large	1.5	October 14, 2016 11:05:32 AM …

Launch Configuration: Node Worker Spot v10

Details

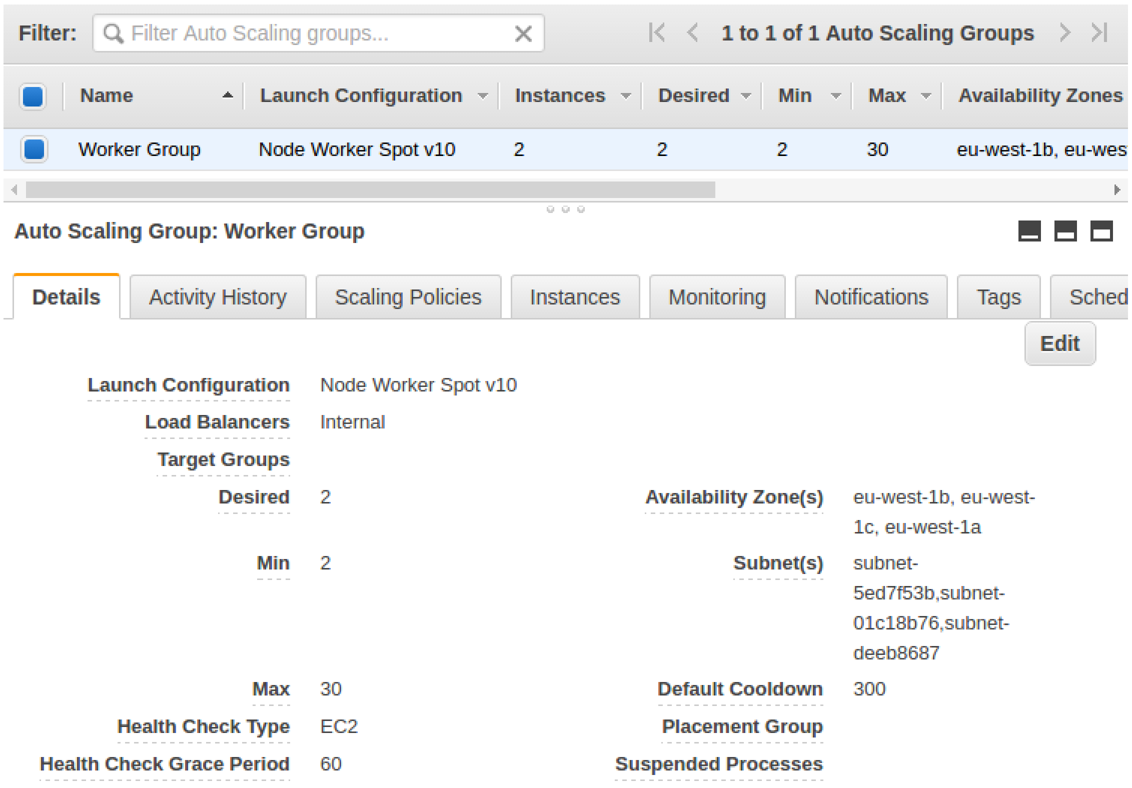
Copy launch configuration

AMI ID	ami-7eeb960d	Instance Type	c4.large
IAM Instance Profile		Kernel ID	
Key Name	disel21	Monitoring	false
EBS Optimized	false	Security Groups	sg-f3df6497
Spot Price	1.5	Creation Time	Fri Oct 14 11:05:32 GMT+300 2016
RAM Disk ID		Block Devices	/dev/sda1
User data	-	IP Address Type	Only assign a public

Figure 23: Amazon Launch Configuration for slave nodes

Because slave node will have responsibility for processing, which consumes a lot of computing power, C4 High-CPU large type for EC2 instance with 2 cores CPU, 8 units of computing, 3.75 GB of RAM would be an option.

After having launch configuration, we can set up auto scaling group by creating a new group named workers. This auto scaling group will use launch configuration to run a new instance and the newly launched one will be attached to the load balancer. Max, min and desired number of the instance can be adjusted depending on the demand. Figure 22 shows the example of a configuration for auto scaling group.


Filter: Filter Auto Scaling groups... 1 to 1 of 1 Auto Scaling Groups

	Name	Launch Configuration	Instances	Desired	Min	Max	Availability Zones
■	Worker Group	Node Worker Spot v10	2	2	2	30	eu-west-1b, eu-wes

Auto Scaling Group: Worker Group

Details | Activity History | Scaling Policies | Instances | Monitoring | Notifications | Tags | Sched

Edit

Launch Configuration	Node Worker Spot v10		
Load Balancers	Internal		
Target Groups			
Desired	2	Availability Zone(s)	eu-west-1b, eu-west-1c, eu-west-1a
Min	2	Subnet(s)	subnet-5ed7f53b,subnet-01c18b76,subnet-deeb8687
Max	30	Default Cooldown	300
Health Check Type	EC2	Placement Group	
Health Check Grace Period	60	Suspended Processes	

Figure 24: Amazon Auto Scaling configuration

The next thing needs to be considered is scaling policies. There are many options for setting up a proper policy. In this case, when mostly CPU consumption will be high, the policies based on CPU percentage are suitable:

- Increase Group Size policy: When CPU >= 80% for 300 seconds
- Decrease Group Size policy: When CPU <= 10 for 9 consecutive periods of 300 seconds

When a new instance is launched, the load balancer will automatically register it to the network.

Slave node workflows

Slave node is an EC2 instance which is launched with a configured image. The image for slave node is also Linux, with NodeJS, FFmpeg installed. Like the master node, we set up an HTTP server by the express framework for NodeJS. Below are the workflows for slave node:

1. Slave node listens to new message
2. Get the new message, SQS hide that message from others

3. Download video segments from S3
4. Process the segment based on the task
5. Upload the processed file to S3
6. Send notification to master through load balancer
7. Delete SQS message

For long polling SQS, we create a new SQS instance from AWS sdk. For example:

```
var sqs = new AWS.SQS();
var longPollingSQS = function() {
    sqs.receiveMessage({
            QueueUrl: SQS_url,
            MaxNumberOfMessages: 1
        }, function(err, result) {
            // if err or there is no more messages in queue
            if (err || !result.Messages) {
                //console.log('No message in queue: ' + err);
                longPollingSQS(); // continue polling
            } else {
                // handle the result.Messages
            }
        }
    })
}
```

Listing 19: Listening for SQS message

In the listing 19, the longPollingSQS is a recursion function which calls itself to looking for a new message in the queue. If there is a new message, it will call the function to handle which is processing function. For example, assume we already have the process-Message function. In another statement, we can call the function as in the listing 20:

```
processMessage(message.Body, function(err) {
    if (err) {
        console.log(err); // error can be send back to master
node
    }
    // delete the message weather it is success or not
    sqs.deleteMessage({
            QueueUrl: SQS_url,
            ReceiptHandle: message.receiptHandle
        }

    })
})
```

Listing 20: Processing message function calling

In a message, it has body properties, which contains the content of the message. After passing the message to another process and receiving an anonymous callback function, we can delete the message by receiptHandle properties of the message.

There are many types of a processing an event. For example, transcoding, trimming, and filter effect. According to the criteria of research, we will give the most typical example for transcoding video file.
To convert a video file from different format and codec to mp4 file with H264 video codec and AAC audio codec, this command is used in FFmpeg:

```
ffmpeg -i input.avi -y -acodec libfdk_aac -b:a 64k -vcodec
libx264 -preset fast -b:v 500k -filter:v
"scale=iw*min(640/iw\,360/ih):ih*min(640/iw\,360/ih),
pad=640:360:(640-iw*min(640/iw\,360/ih))/2:(360-
ih*min(640/iw\,360/ih))/2" 360p.mp4
```

Listing 21: Transcoding video to 360p resolution.

The command in the listing 21 converts and scales video to 640x360 resolution with 64k audio bitrate and 500k for video bitrate. The “preset” tells the compressing efficiency of algorithms. If we set it to slow instead of fast, the quality of video might be better but the time for processing will increase quite a lot. Similarly, we have the command in listing 22 for 1280x720 resolution with 128k audio bitrate and 1000k video bitrate.

```
ffmpeg -i input.avi -y -acodec libfdk_aac -b:a 128k -vcodec
libx264 -preset ultrafast -b:v 1000k -filter:v
"scale=iw*min(1280/iw\,720/ih):ih*min(1280/iw\,720/ih),
pad=1280:720:(1280-iw*min(1280/iw\,720/ih))/2:(720-
ih*min(1280/iw\,720/ih))/2" 720p.mp4
```

Listing 22: Transcoding video to 720p resolution.

With the filter option, FFmpeg auto calculates and scales video to 16:9 ratio. Padding might be added to the video if the original width per height is not 16:9.
Although the command looks complicated, fortunately, there is one library called fluent-FFmpeg for NodeJS to generate the command from JavaScript, which is much cleaner and easier to use [appendix 2].

When processing segment has been done, slave node will send a message to master through the load balancer to notice about the status. If the process is successful, the master node will update the database for segmentDone as a success. If the process fails with an error message, the master node will mark the status field in the database as fail. Master node checks to see if segmentDone equal to segmentNum in the database, it will send a request through the load balancer to anyone of slave nodes. This request asks slave node merge all processed file into one, then upload the file back to S3. This process can be described by the following workflows:

1. Master node receive the notification from slave node
2. Update the number of segment done to database
3. Check the number of segmentDone and total
4. If the segmentDone equal to total, send request to slave node
5. One slave node will download all segments from S3
6. Slave node merges segments into one file
7. Upload the file to S3
8. Slave node sends a message to master node inform the work done
9. Master node update status in database
10. Master node send a message to client by web socket

Once everything is complete, the master node will notify users via WebSocket.

WebSocket

Every time the user opens the application there is a WebSocket connection between server and client and this socket will be saved into a place on server memory called room. The name of the room is user id. When we have the room name, the master node can emit an event to the room, which will broadcast to all user devices whose sockets are inside the room.

```
var io = require('socket.io').listen(server, {
    transports: ['websocket', 'polling']
});
io.on('connection', function(socket) {
          socket.join('user_' + userId); // join the room which
has user id
})
```
Listing 23: Socket.io connect

Listing 23 shows the source code to get connection up between client and server in NodeJS. When emitting an event to client, listing 24 shows how to handle that:

```
io.to('user_' + userId).emit(event, data); // send the message to all
devices of the user
```

Listing 24: Emit event to client

For a video processing event, data would have all necessary information about trans-coded video like filename, URL, thumbnail, duration. Finally, when all the processing has been done, the client receives the message data from the socket and updates the view or shows some notification.

5.5 Result

The outcome of this study is EdVisto product. The product has video editor running on the web platform. The editor is illustrated in figure 25.

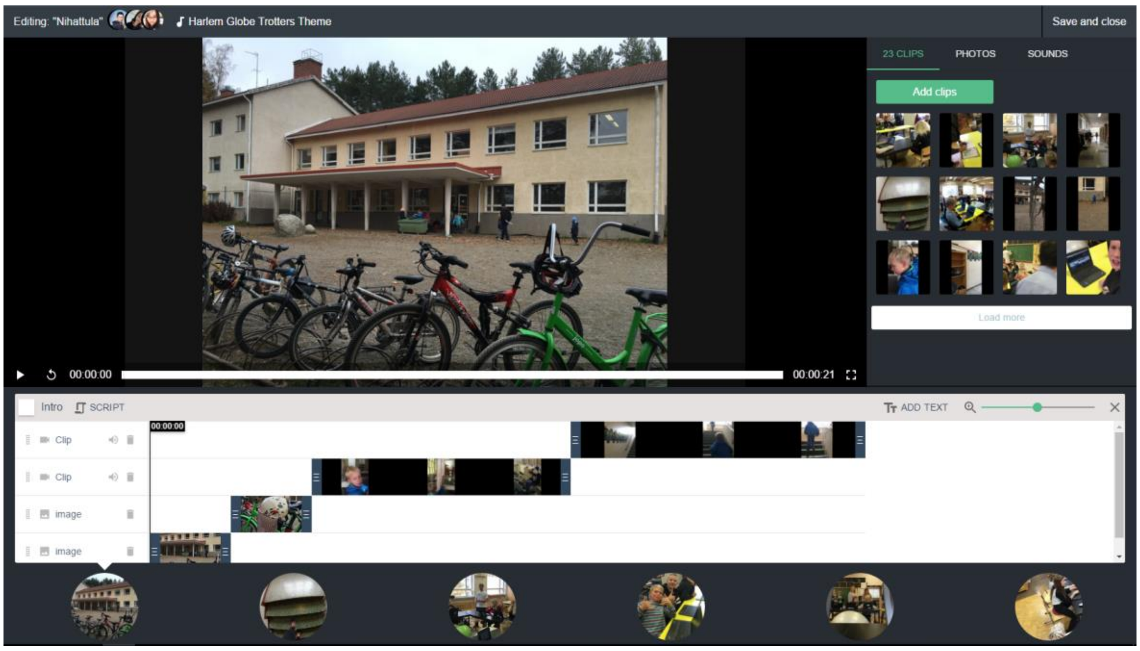


Figure 25: Screenshot of the product

This editor allows the client to upload videos to the website and edit them directly in the web environment. Some features are trim video, merge video, changing the time and many others. Not only supporting video, EdVisto also supports other media type like audio, photo, and text. In summary, the outcome has archived the goal and given the answers for the research questions.

6 Discussion

6.1 Performance benchmark

In this benchmark, we will compare the transcoding performance of different video files between one single machine and a distributed system with 5 slave nodes running.
All the machine is the EC2 instance with duo core CPU, 3.75 GB Ram, 8 GB SSD. The video sample is downloaded from DivX official website [37].

- Video 1: size 56.7mb, duration 1.48 mins
- Video 2: size 859mb, duration 14.58 mins
- Video 3: size 16.6mb, duration 10.02 mins
- Video 4: size 52.5mb, duration 5.01 mins
- Video 5: size 383.9mb, duration 3.4 mins

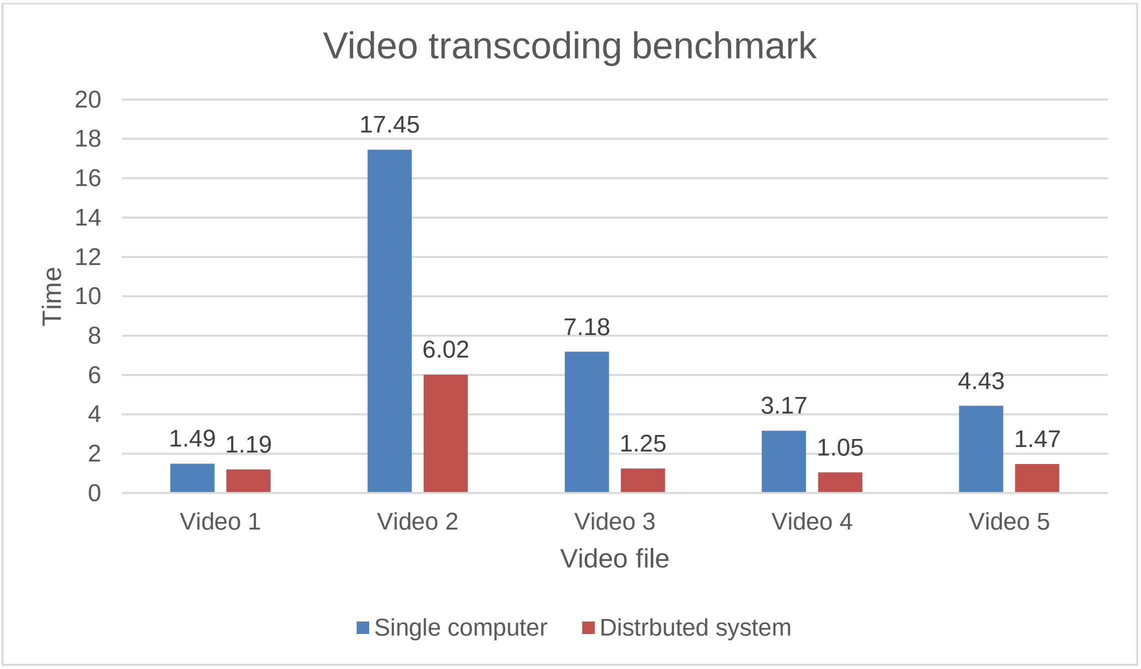


Figure 26: Video transcoding benchmark

Based on the comparison in figure 26, we can see a huge difference of processing time between a distributed system to a single machine. Obviously, the processing time is much faster in the distributed way.

However, in some cases the video has a short duration, and then the difference is negligible. When the file has a short duration, it also means less segment or maybe only one segment. Therefore, the process is still only being taken by a slave node. On the other hand, because of the latency of networking, the download and upload files also accounted for a period in the whole process.

6.2 Cost analyse

To make it easier to evaluate the cost, we can compare it with an Amazon services for media transcoding called Amazon Elastic Transcoder. With this service, users can transcode video in the form of premium services. Prices for the processing of a video file in the EU region are as follows:

- Standard Definition – SD (Resolution of less than 720p) $0.017 per minute
- High Definition – HD (Resolution of 720p or above) $0.034 per minute. [38.]

For example, to process 100 mins of video to SD and HD output will cost:
100 x ($0.017 + $0.034) = $5.1

If EdVisto is running one master node in a T2 medium instance which costs $0.056 per hour and five slave nodes in C4 large type which costs $0.119 per hour. The total cost for one hour of using is:
$0.056 + 5 x $0.119 = $0.651

To process 100 mins of video, it takes around 20 mins to 50 mins which are only around $0.2 to $0.6. Moreover, by launching a C4 instance in spot strategy, EdVisto is saving from 80% to 90% of total cost now. Obviously, it is a huge benefit when implementing a distributed system for video processing.

7 Conclusion

Distributed system has a lot of challenges in the design and operation phase. However, it cannot be denied that they also bring many advantages. In the past, to build a distributed system is very expensive when we must invest in hardware devices as well as hire a team of professionals for management and maintenance. However, since the fast growth of cloud computing, anyone can build a distributed system. By hiring virtual servers from the cloud service providers, businesses no longer should worry about the maintenance cost as well as capital cost of server. Also, a distributed system can share numerous open source tools and tools that save a lot of time for development.

In video processing, there are very strict requirements such as faster processing speed, auto scaling and being durable. There have been many solutions for the distributed processing of the video. However, EdVisto needs a complete solution which can be applied immediately in production.

Firstly, this system has solved the problem of optimizing the processing of a video. Rather than processing each file in turn, the splitting files and parallel processing reduce a lot of time. Second, the architecture can serve many concurrent users, as well as ensure that the system will operate smoothly by using a queue system. Finally, due to the ability to scale dynamically based on demand, it solves two problems at once. Those are the overload and cost issues. However, there are some limitations that can be improved in the future such as optimizing the performance, reducing the latency of connection and supporting more resolutions and mime types.

In conclusion, the main research problems have been solved by applying distributed system on cloud environment. EdVisto has achieved some success with this approach. Currently, this solution has been adopted in production and it has met the original requirements. Not only does it apply to video processing, it can also be applied to various types of media files, as well as handling common tasks with a good performance.

References

1 White paper: Cisco VNI Forecast and Methodology, 2015-2020 [online]. Cisco official website.
URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html. Accessed 19 August 2016.

2 George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. Distributed system concepts and design (5th edition). Pearson; 2011. p1. Accessed 19 August 2016.

3 Fred B. Schneider. On Concurrent Programming. Springer; 1997. Accessed 2 September 2016.

4 V. K. Garg, Neeraj Mittal. Time and State in Distributed Systems, Wiley Encyclopedia on Parallel and Distributed Computing; 2008 p1. Accessed 5 September 2016.

5 Luiz André Barroso, Urs Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers; 2009. p80-84. Accessed 15 September 2016.

6 HDFS Architecture Guide [online]. Apache official website.
URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Accessed 21 September 2016.

7 Hard Drive Reliability Review for 2015 [online]. Blackblaze official website.
URL: https://www.backblaze.com/blog/hard-drive-reliability-q4-2015/. Accessed 21 September 2016.

8 Krishna Nadiminti, Marcos Dias de Assunção, Rajkumar Buyya. Distributed Systems and Recent Innovations: Challenges and Benefits. Infonet Vol. 16 Issue 3; 2006. Accessed 30 September 2016.

9 About NodeJS [online]. NodeJS official website.
URL: https://nodejs.org/en/about/. Accessed 8 October 2016.

10 Kai Lei, Yining Ma, Zhi Tan. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js. Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on; 2014: p665. Accessed 8 October 2016.

11 Applications using NodeJS [online]. NodeJS official github page.
URL: https://github.com/nodejs/node/wiki/Projects,-Applications,-and-Companies-Using-Node. Accessed 8 October 2016.

12 Hirotaka Nakajima, Masao Isshiki, Yoshiyasu Takefuji. WebSocket Proxy System for Mobile Devices. Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on; 2013. Accessed 20 October 2016.

13 About WebSocket protocol [online]. WebSocket official website.
URL: http://www.websocket.org/aboutwebsocket.html. Accessed 20 October 2016.

14 About socket.IO module [online]. Socket.IO official website.
URL: http://socket.io/. Accessed 20 October 2016.

15 Peter Mell, Timothy Grance. The NIST Definition of Cloud Computing. NIST Special Publication 800-145. National Institute of Standards and Technology; 2011. Accessed 22 October 2016.

16 Qusay F. Hassan. Demystifying Cloud Computing. The Journal of Defense Software Engineering (CrossTalk) 16-21; 2011. Accessed 22 October 2016.

17 Amazon Global Infrastructure [online]. Amazon official website.
URL: https://aws.amazon.com/about-aws/global-infrastructure/. Accessed 24 October 2016.

18 About Amazon web service [online]. Amazon official website.
URL: https://aws.amazon.com/about-aws/. Accessed 24 October 2016.

19 Amazon Elastic Compute Cloud [online]. Amazon official website.
URL: https://aws.amazon.com/ec2/details/. Accessed 24 October 2016.

20 Amazon Simple Storage Service [online]. Amazon official website.
URL: https://aws.amazon.com/s3/details/. Accessed 24 October 2016.

21 Amazon Elastic Load Balancing [online]. Amazon official website.
URL: https://aws.amazon.com/elasticloadbalancing/. Accessed 24 October 2016.

22 Amazon Classic Load Balancer [online]. Amazon official website.
URL: https://aws.amazon.com/elasticloadbalancing/classicloadbalancer/. Accessed 24 October 2016.

23 Amazon Application Load Balancer [online]. Amazon official website.
URL: https://aws.amazon.com/elasticloadbalancing/applicationloadbalancer/. Accessed 24 October 2016.

24 Amazon Auto Scaling [online]. Amazon official website.
URL: https://aws.amazon.com/autoscaling/details/. Accessed 24 October 2016.

25 Amazon Simple Queue Service [online]. Amazon official website.
URL: https://aws.amazon.com/sqs/details/. Accessed 24 October 2016.

26 About x264 codec [online]. VideoLan official website.
URL: http://www.videolan.org/developers/x264.html. Accessed 25 October 2016.

27 About Xvid codec [online]. Xvid official website.
URL: https://labs.xvid.com/project/. Accessed 25 October 2016.

28 FFmpeg [online]. FFmpeg official website.
URL: https://www.ffmpeg.org. Accessed 25 October 2016.

29 About Divx codec [online]. Divx official website.
URL: http://www.divx.com/. Accessed 25 October 2016.

30 Codec and format [online]. Zencode official website.
URL: https://app.zencoder.com/docs/faq/codecs-and-formats. Accessed 25 October 2016.

31 About ffmpeg open source [online]. FFmpeg official website.
URL: https://www.ffmpeg.org/about.html. Accessed 25 October 2016.

32 FFmpeg stream mapping [online]. FFmpeg official website.
URL: https://trac.ffmpeg.org/wiki/Map. Accessed 25 October 2016.

33 FFmpeg concatenate file [online]. FFmpeg official website.
URL: https://trac.ffmpeg.org/wiki/Concatenate. Accessed 25 October 2016.

34 Browser and mobile compatibility [online]. Mozilla official website.
URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats. Accessed 27 October 2016.

35 AAC and MP3 comparison [online]. Winxdvd official website.
URL: https://www.winxdvd.com/resource/aac-vs-mp3.htm. Accessed 27 October 2016.

36 Jie Zhu, Rangding Wang, Juan Li. The filterbank in MP3 and AAC encoders: A comparative analysis. Electronics, Communications and Control (ICECC), 2011 International Conference on; 2011. Accessed 27 October 2016.

37 Sample videos [online]. Divx official website.
URL: http://www.divx.com/en/devices/profiles/video. Accessed 28 October 2016.

38 Amazon Elastic Transcoder [online]. Amazon official website.
URL: https://aws.amazon.com/elastictranscoder/pricing/. Accessed 28 October 2016.

39 Oliver. Icons for the figures [online]. Flat icon creator official website.
URL: http://www.flaticon.com/authors/madebyoliver. Accessed 10 October 2016.

Splitting video file by ffmpeg-fluent

```
/**
 * Split video file into multiple
 * @param inputFile
 * @param outputFile
 * @param outputPath
 * @param callback
 */
var splitVideo = function (inputFile, outputFile, outputPath
,callback) {
    fs.ensureDir(outputPath, function (err) {
        if (err)
            callback(err);
        else {
            ffmpeg(inputFile)
                .audioCodec('copy')
                .videoCodec('copy')
                .outputOptions([
                    '-f segment',
                    '-segment_time 60',   // split to 60 secs
length
                    '-segment_list_size 0',
                    '-segment_list ' + outputPath +'seg-
ment.ffcat'
                ])
                .output(outputFile)
                .on('start', function(cmd) {
                    console.log('Started ' + cmd);
                })
                .on('error', function(err) {
                    console.log('An error splitting file: ' +
err);
                    callback(err);
                })
                .on('end', function() {
                    // return the array of files
                    fs.readdir(outputPath,function(err, files){
                        if (err)
                            callback(err);
                        else
                            callback(null, files);
                    });
                })
                .run();  // execute;
        }
    });

};
```

Converting video file by ffmpeg-fluent

```
/**
 * Convert video to h264 video codec and aac audio codec
 * Export to 360p and 720p resolution
 * @param outputPath
 * @param inputFile
 * @param outputFile
 * @param callback
 */
var convert = function (outputPath, inputFile, outputFile,
callback) {
    async.parallel([
        // convert to 360p
        function (next) {
            ffmpeg(outputPath + inputFile)
                .audioCodec('libfdk_aac')
                .videoCodec('libx264')
                .outputOptions([
                    '-preset fast'
                ])
                .output(outputPath + '360/' + outputFile)
                .videoBitrate('400k')
                .audioBitrate('64k')
                .size('640x360').autopad()

                .on('start', function (cmd) {
                    console.log('Start converting ' + cmd);
                })
                .on('error', function (err) {
                    console.log('An error occurred: ' + err);
                    next('PROCESS_FAIL');
                })
                .on('end', function () {
                    console.log('Finished converting');
                    next(null);
            })
            .run();  // save the file;
    },
    // convert to 720p
    function (next) {
        ffmpeg(outputPath + inputFile)
            .audioCodec('libfdk_aac')
            .videoCodec('libx264')
            .outputOptions([
                '-preset veryfast'
            ])
            .output(outputPath + '720/' + outputFile)
            .videoBitrate('1200k')
            .audioBitrate('128k')
            .size('1280x720').autopad()

            .on('start', function (cmd) {
                console.log('Start converting ' + cmd);
            })
            .on('error', function (err) {
                console.log('An error occurred: ' + err);
                next('PROCESS_FAIL');
            })
            .on('end', function () {
                console.log('Finished converting');
                next(null);
            })
            .run();  // save the file;
    }
], function (err) {
    if(err)
        callback('PROCESS_FAIL');
    else
        callback(null);
});
```