LOW DELAY VIDEO TRANSCODING SERVICES ON DISTRIBUTED COMPUTING PLATFORM.

Raisul Bhuiyan

Master of Science Thesis

Supervisors

Dr. Julien Michot
Senior researcher.
Media Technologies,
Ericsson Research.
Stockholm, Sweden.

Dr. Sébastien Lafond
Embedded Systems Laboratory
Faculty of Science and Engineering
Åbo Akademi University
Åbo, Finland.

Dr. Benny Lövström
Senior Lecturer.
Department of Applied Signal Processing.
Blekinge Institute of Technology.
Karlskrona, Sweden.

March 2016.
ABSTRACT

The demand of digital video with higher resolution is increasing everyday and in a modern world the videos are consumed in all kinds of multimedia devices. The transmission of higher quality videos over the internet require higher bandwidth, which is not an acceptable option. So, it is necessary to compress the video to a compact file by removing redundancies and detail information.

The process of compressing a video file requires a lot of complex calculations, which is a time consuming process, specially for live telecasting or real-time video conferencing. In addition videos with higher quality such as higher number of Frame per Second (FPS) or higher resolution like HD and 4k video requires huge redundant data processing. Hence, this operation causes delays during the video playback. To minimize the time delay for the video coding, there are coding methods such as lossless and lossy coding which has been around for a long time. However, the idea to increase the number of processing unit like CPUs and memory for video coding software is an area that require further research to improve coding techniques.

Distributed system uses available resources in the network to achieve a common goal. It explores the available infrastructure so that the task can be done in parallel. Cloud computing is a great example of distributed system which has fully dedicated resources for such complex jobs.

This thesis deals with these areas in real-time to lower the video coding delay through investigating distributed resources as well as the parallelization in video coding standards such as AVC and HEVC. It has been carried out with a collaboration with Ericsson Research in Stockholm.

Keywords: Distributed Transcoding, Distributed Computing, Apache storm, Scheduling, Openstack, Cloud Computing.
ACKNOWLEDGEMENTS

First I would like to thank everyone who supported me during the thesis work. This project would not be possible without the help from Dr. Julien Michot. His continuous support and guidance throughout the research work inspired me to go forward. It was a great opportunity to work under his supervision. I am also grateful to Dr. Thomas Rosert for giving me the opportunity and for including me in his team.

I would also like to thank my supervisors at the universities Dr. Sébastien Lafond from Åbo Akademi University (ÅA) and Dr. Benny Lövström from Blekinge Institute of Technology (BTH) for their help to make this thesis possible.

Finally, I would like to thank my family for the unconditional moral supports for my academic pursuit.

Raisul Haque Masud Bhuiyan.

Stockholm, 2016-03-18.
CONTENTS

Abstract i

Contents iii

List of Figures v

Glossary vii

1 Introduction 1
   1.1 Background ........................................ 2
   1.2 Related Work .................................... 3
   1.3 Problem Definition .............................. 4
   1.4 Objective of the thesis .......................... 6
   1.5 Thesis structure ............................... 7

2 Video Transcoding 9
   2.1 Video Coding for Digital Video .................... 9
       2.1.1 Video Compression Steps and Types ............ 12
   2.2 Video Transcoding .............................. 14
   2.3 High Efficient Video Coding (HEVC) ............... 15
       2.3.1 Picture/Frame Partitioning .................. 16

3 Cloud Computing 21
   3.1 Cloud Computing ................................ 21
       3.1.1 Classification of Cloud computing .......... 22
   3.2 Openstack ........................................ 24
       3.2.1 Architecture of Openstack Nova ............. 24
   3.3 Configuring Openstack for Apache storm ............ 26
       3.3.1 Create Images .............................. 27
       3.3.2 Create Security Group and Network .......... 28
       3.3.3 Openstack Network .......................... 29
       3.3.4 Import Key Pair ............................. 31
       3.3.5 Creating Instances ......................... 31
4 Apache Storm
4.1 Apache storm .................................. 34
4.1.1 Apache Storm Architecture .................. 35
4.2 Component of Storm ............................ 37
4.2.1 Spout ...................................... 37
4.2.2 Bolt ...................................... 38
4.2.3 Parallelization in Storm ....................... 38
4.3 Configuration of Apache storm .................... 40
4.3.1 Configure Zookeeper .......................... 41
4.3.2 Configuring storm for Windows ................. 41
4.3.3 Configuring storm for Unix .................... 43
4.4 Scheduling .................................... 43

5 Topology Configuration .......................... 45
5.1 Current Topology ............................... 45
5.2 Multiple Camera ............................... 47
5.3 Scheduling ................................... 49
5.4 Maintaining a Topology in Storm Cluster ........... 49
5.4.1 Apache Maven ................................ 50
5.4.2 Run a Topology ............................ 51

6 Results .................................. 52
6.1 Data Collections ................................ 52
6.1.1 Topology Information ....................... 52
6.1.2 Topology Visualization ...................... 53
6.2 Results for Local computers ..................... 54
6.3 Cloud Computers .............................. 56
6.3.1 Measurements for 720p videos ................ 56
6.3.2 Measurements for 1080p (HD) videos .......... 57
6.3.3 Measurements for 4K videos .................. 58

7 Conclusion and Future Work ...................... 59
7.1 Conclusion .................................... 59
7.2 Future work .................................. 60

Bibliography ................................ 62
LIST OF FIGURES

1.1 One of the video transcoding application (On-demand video streaming.)[1] 2

2.1 RGB and YCrCb color components in a frame. .......................... 11
2.2 The simplest video coding over the network. ........................... 12
2.3 Sequence of a video frame in closed GOP. ............................ 13
2.4 Sequence of a video frame in open GOP. .............................. 13
2.5 A simplified implementation of bit-rate encoder [2, 3] ............... 15
2.6 A Scalable encoder for HEVC video coder proposed by Hahyun Lee et. al. [4] ................................ 16
2.7 Coding Tree Block of HEVC from 64x64 to 8x8 ..................... 17
2.8 Coding Tree structure for figure 2.7 ................................. 18
2.9 Different prediction unit sizes of Coding unit ....................... 18
2.10 CTBs are divided to correspondent Tilses. ......................... 19
2.11 Wavefront Parallel Processing of CTBs in parallel. ............... 20

3.1 Cloud infrastructure for Service oriented architecture (SOA). ...... 23
3.2 Architecture of Nova in Openstack [5]. ................................ 25
3.3 Overview of the videotr project in Openstack cloud. ............... 27
3.4 Uploading and storm image to the cloud ............................. 28
3.5 A part of the updated security group for instances. ................. 29
3.6 The network topology in the cloud with all instances. .......... 30
3.7 List of imported key pairs in Access and security. ............... 31
3.8 Instant launcher for Openstack. .................................. 32
3.9 Instant launcher for Openstack. .................................. 33
3.10 Instant launcher for Openstack. .................................. 33

4.1 Apache Storm Architecture. ........................................ 35
4.2 A storm topology Example. ..................................... 37
4.3 A storm topology Example. ..................................... 38
4.4 A sample code for showing different level of storm parallelism. .... 39
4.5 Different level of parallelism in Apache storm based on figure 4.4. .... 40
4.6 Zookeeper status and connected storm components with zookeeper cluster. .................................... 41
4.7 A part of configuration file for storm.yaml. ........................ 42
GLOSSARY

API  Application Programing Interface.

CPU  Central Processing Unit.

VCPU  Virtual Central Processing Unit.

UI  User Interface. An interface that reports back to the users about the present state of the software in real-time.

GPU  Graphical Processing Unit. A processing unit used for rendering graphics.

VM  Virtual Machines is being used on the cloud computing.

HTTP  Hyper Text Transfer Protocol.

GOP  Group Of Picture

DCT  Discrete Cosine Transform.

HEVC  High-Efficient Video Coding.

AVC  Advance Video Coding.

CTU  Coding Tree Unit.
IaaS  Infrastructure as a Service.

PaaS
   Platform as a Service.

SaaS
   Software as a Service.
1 Introduction

Nowadays videos are being recorded and played in all sorts of multimedia devices from smart phone to high definition television like 4k TV. The resolution of the recording devices and the displays are improving continuously, resulting in higher quality video data. High definition cameras increase the raw video size, hence, the displays need to deal with bigger files with lots of details. These huge amount of data with raw video file is challenging to store and transmit over the network, not to mention the devices with smaller memory and processing unit cannot store or process these big data. In order to lower transmission bits, playback delay and save memory, videos need to be converted into certain compressed digital formats. This procedure to compress video data from one format to another formats is known as video coding and it has been around for some time. Video coding algorithm has been developing over the time to produce lossy time efficient coding methods like entropy coding, Quantization Parameter (QP) coding and frame prediction using different coding standards such as H.260, H.261, H.262 (MPEG-2), H.263, AVC (H.264) and HEVC (H.265). Although the video standards make sure that videos are encoded and decoded with some specific criteria, the video playback devices such as smart phones, tablet, laptops and television might have different video player with different requirements. This issue could be solved by first storing the video with one specific coding standard and then transmit that video to the user demanded format.

However, this will not be the case for live video streaming such as sports event, concerts or video conferences. Video coding is a computationally heavy process and several methods has been proposed in order to increase its efficiency [6, 7]. Process like GOP encoding uses motion vector detection and DCT coefficients from the Bidirectional frames, decrease the coding time, but it is really complex when videos are on-demand. This problem can be solved by using a parallelized codecs which can take the advantage of increased resources like processing units and memory but it is
not economical or energy efficient since more memory system increases the prices of the electronic devices and higher processing unit requires cooling systems and more clock frequency resulting dynamic power consumption. Video coding for higher resolution like 4K videos is not possible in a single machine. This is where distributed transcoding become useful.

Figure 1.1: One of the video transcoding application (On-demand video streaming.)[1]

Figure 1.1 shows how live streaming videos are being processed, encoded and transmitted over the medium to the receivers. It also shows how different media can have different communication systems and devices which may require different coding standards. This is one of the examples of multiple application for video transcoding. Other applications are namely mass video storage in the cloud, videos stored in the electronic devices like computer storage, cell phone, network storage system and etc.

1.1 Background

Video coding requires encoding to compress the raw video file to such level where it is economical to store the video or to transmit it over the network. In some cases the coded video might be need to be decoded and re-encoded into a different format so that it fits the requirements of the playing device or display. This process of converting a compress video from one format into another is known as video transcoding [8]. This process is very important because we need to avoid data loss or lowering the quality during video coding. Coding time will add delays and sending the whole file will cause
jitter in the start of the video. To avoid these issues, the transcoding process is done using several powerful platform and software in the network and then transmitted over the internet to the playing device. This process of using multiple connected devices in the network is known as distributed computing. This distributed computing system could be using a couple of powerful CPUs in the same network or a cloud computing which will be discussed in chapter 3. Several works have been conducted on video coding in cloud and distributed system. However, the robustness of the cloud and network delay for communication between the systems play a big role. When we are using system with higher number of processing unit, we have lower delay which can provide a good control over the traffic but the disadvantages of such system is higher cost and power consumptions.

1.2 Related Work

Video coding standards H.265 or HEVC is the latest codec which has much higher complex computational requirements compare to its predecessor codec H.264 or AVC. HEVC aims to reduce half of the bitrate for AVC codec [9], hence it requires more processing unit as compared to any previous video codec. This is why parallel transcoding become very important research area. Parallel distributed computing for video transcoding has been trending since the beginning of video transcoding. A good number of research work have been conducted to explore parallelization that comes with the coding standard. The coding standard has improved parallelization with time. Furthermore there were several attempts to use multi-core or many core computers with parallelized transcoder. However, cloud computing has become more acceptable due to several advantageous terms such as cost, performance and higher level of parallelization [10, 11].

The authors of [10, 12, 13] has worked with video partitioning methods for HEVC encoding. The author proposes three stages for the video transcoding namely Partitioning stage, Transcoding stage and Combining stage. The video partitioner segmentize the uncompressed video file into multiple non-overlapping segment. This stage only recognise the start and end point of a video by using the video partitioning algorithm provided by the user. This stage also downsize the network transmission overhead by partitioning video and sending them to the transcoding stage where these segmented files will be mapped. The author of [10] described Uniform and GOP based video par-
tioning. These experiments with GOP based transcoding in cloud computing showed that it reduces transcoding time significantly. It is able to decrease HEVC transcoding time even with a greater computational complexity. This work was able to keep a corresponding bitrate with a higher number of I-frame. Although this work has very good similarities with the research that we are conducting in Ericsson research. Our research will also use independent GOP for distributed video transcoding. However, we will use different platform, transcoder and distributing software for scheduling.

Authors of [9, 13] focused on picture based and coding tree based parallelization. They explored also the Wavefront Parallel Processing (WPP) for Multi-core and Many-core processors. The authors were aiming to transcode different part of a frame in parallel without downgrading the quality and bitrate. These articles also focused on WPP coding implemented on multi- and many-core processors which they called Overlapped Wavefront (OWF). OWF processes several blocks and frame at the same time in different processing cores. The transcoder software that we are using is ‘C65’ and it is capable to use these existing parallelism.

In this paper [8] author worked with video transcoding with data level parallelism in a cloud computing platform. This paper shows very good improvements on transcoding with a dynamic scheduling to prevent overflow and underflow of the transcoded data. It has also shown improvements for resource allocation and deallocation in the cloud infrastructure. This work is largely similar to our research. However, the author worked with different coding standard (such as H.264), cloud platform as well as other parallelisms.

1.3 Problem Definition

Video coding has been around since the development of broadcast, interlacing and recording technique of predictive compression [14, 7]. Nowadays it has grown interest with modernization of the electronic devices and popularity of the on-demand videos like online gaming and sports event. Computer gaming has higher requirements when it comes to game precision, responsiveness and fairness. These requirements in high performance gaming architecture guarantees good playability [15, 16]. Such on-demand video coding has made it almost impossible for a single device to keep up with the network delays, processing delays and increased file size. Video transcoding is a complex system which requires several process to deliver a good quality video to
the end user with less distortion and minimized size in a required transmission speed. It becomes a complex task for the video service provider to maintain all these standards. On the other hand, it is not also possible to get big number of processing unit to keep encoding the video because some of the resources might not be available in any random time. A dedicated cloud service with software and hardware makes the service more costly. So one possible idea would be to use the devices that are already in the network for example a home network might contain several personal computer, laptops, tablets and phones [17]. If the network structure is capable to use the existing devices in the private network and create a simple cloud network, the processing unit could be increased significantly. This simple cloud system is easily scalable and a cost efficient distributed system as it uses existing systems.

There are several ways proposed to deal with small distributed systems. It received popularity due to the fact that it is possible to control very easily with efficient and optimized software to minimize arithmetic processes. It is easily scalable with minimal extra cost as the devices already exists in the network. However, this kind of process might require constant monitoring of the process and the reliability of the software will play a big role for a better outcome. It is also challenging for the following reasons:

- **Availability of the resources:** As the distributive system means the use of several available system it will require processing units to be connected to the host device continuously. Any kind of interruption on the communication might lead to data loss and make the process unstable. This kind of error will create more delays to the coding process which probably could take more time and produce unsatisfactory results than the serial implementation. So the transcoding process will require a dynamic resource management to deal with this kind of network or resource failure where the software should be able to reassign the failed tasks that were allocated for the failed device(s).

- **Supportability for request:** Input videos (raw videos) are segmented in a shorter length in order to avoid memory overflow in the devices. Moreover, the devices with smaller memory size will not be able to allocate the required memory for unsegmented video file. In addition, the host device sends the bitstream with a specific structure and an architecture of the decoding processing. We also need to consider that some of these devices will not be able to support the bitstream or fail to follow the instructions from the main device.
• **Scheduling:** We are also considering the scheduling process for the task distribution. The host devices should be aware about the available connected devices. Some of the connected devices might not be available as it has other tasks to complete. In addition, the scheduler should be able to know which device is responsible for what kind of tasks and what type of communication packet it should receive form the other devices. So, simple Scheduler takes part in load balancing and task distribution, which makes it complicated to create Scheduler for distributed systems.

• **Platforms:** The idea of distributive computing is fairly new and still developing. These are platforms that have not been adopted to this concept of sharing platforms in distributive computing. Moreover, the ideal distributive systems usually uses data to be stored and processed but this concept of processing bit-stream should be dealt in the new devices.

### 1.4 Objective of the thesis

The main objective of the thesis is to analyze the complexity of distributed computing such as Task distribution, Scheduling, Resource allocation and so on. These analysis will be with different architectures like x86 (32-bit) and x64 (64-bit) instruction set and operating systems like MS Windows and GNU/ Linux and compare the results. For this purpose, file or segment based video will be used to analyze the video encoding and transcoding process in C65 which is a video coding software developed in Ericsson research. C65 is capable to encode and transcode video files to H.265. This software can use all sorts of a parallelization and has the capability to use multi-threading and Graphical Processing Unit (GPU). The sub-goal of this thesis is to familiarize with the C65 and find the efficiency of this software in the distributed systems. The research will explore the GOP based video and amplify the data-level parallelism. The final work of this thesis should also be able to monitor the process in all stages from reading file to playing the bitstream using a dash player from videos saved by a HTTP server. The research work also requires to install and configure Apache Storm in both of the operating systems so that the host in a storm cluster can read input files directly from one or several cameras. Later, we will discuss how to configure storm and the topology design so that the storm cluster can run without any errors. Apache storm monitors each processes and report back to the user using a web UI and the work described in this
thesis aims at measuring the delays in each stages like reading video files, transcoding and dashing with different resource allocation. Finally, it compares both results in the MS Windows (for later part of the thesis we will refer as local computers) and GNU/Linux (this will be called as cloud VMs and Cluster) and discuss the results and possible improvements for both of these clusters.

1.5 Thesis structure

This thesis is divided into seven chapters. A short summary of each chapters are given here.

- Chapter 1 discusses about the background and shortcomings with the video coding techniques. It introduces distributed computing and how it is going to be useful for the video coding with huge redundant data and higher resolution. Chapter 1 discusses about the drawbacks that are coming with the distributed architecture and how we can deal with these few disadvantages so that it can improve the outcomes.

- Chapter 2 introduces the background of digital video compression. Chapter 2 discusses about different coding standards and their advantages and disadvantages. This chapter discusses about video transcoding and the necessity of faster transcoding software. It will explain transcoding standards such as H.264 and H.265 (HEVC). Chapter 2 gives brief description on HEVC codec.

- Chapter 3 discusses briefly about Openstack and how it has been used in Ericsson research’s cloud. It will also discuss how we should configure security group for each of the storm capable instances (Defined in section 3.2).

- Chapter 4 introduces Apache storm and its required set of pre-installed softwares in order to run a successful Apache Storm environment. It briefly introduces storm configuration for Windows, Linux and the usability, advantages and disadvantages of Apache storm. Chapter 4 discusses how storms operates and handles parallelization on task levels and thread levels. It will also give instruction how we can monitor, collect log files and use visualization and topology stats to measure the delays.
• Chapter 5 Apache storm requires a topology configuration which will explain the data flow from Spouts to Bolts (Both are defined in Chapter 4). This chapter analyzes the possibilities to vary the level of parallelization based on different requirements.

• Chapter 6 shows the measured data and which part of the software is exhibiting most parallelization. It also discusses about the introduced delays with different video resolution.

• Chapter 7 This chapter concludes on the work that has been presented with the corresponding results. It also shows how we can improve the future of distributed systems using clouds and networked computers.


2 VIDEO TRANSCODING

Previous chapter gave an overview to necessity of the video coding in video transmission and the coding standard. This chapter will give brief introduction and background for video coding. In the following sections, it will discuss about the basic video compression techniques and video transcoding and the most recent coding standards like High Efficient Video Coding (HEVC).

2.1 Video Coding for Digital Video

Video coding is a technique to store videos into discrete representation of the real world images sampled in temporal and spatial domain. These videos are stored as an image or frame and those frames in temporal domain are played with at least 25 frame per second (FPS). These frames can have different resolution as listed in Table 2.1. Video frames also contain color components according to the color space. Depending on the color space each frame has one or three pixels value. A single pixel has three color component and each color uses N-bit to store color intensity, where N could be 8, 16 and 24. Let's assume a color component has value from 0 to 255 (in binary 8-bits) and that requires 24-bits to store 3 color values for one pixel in a single frame. The most common color formats are RGB and YCbCr. RGB color describes the color value for Red, Green and blue values of a pixel in a frame and YCbCr are calculated using RGB value. The YCbCr color scale is developed to adjust with human eyes and it is not an absolute color space, it is a way of encoding RGB information. The visual perception is better with the YCbCr color format. In this format Y is the Luminance and calculated by weighted average, \((K_r, K_g, and K_b)\) by this equation

\[ Y = k_rR + k_gG + k_bB \]
Luminance $Y$ is the Gray scale representation of the original image. Rest of the information is calculated as follows

\[
C_r = R - Y \\
C_g = G - Y \\
C_b = B - Y
\]

We have discussed before that it is necessary to code a video in different resolution. The different resolution requires different subsampling in Luma and chroma value. $YC_bC_r$ has multiple subsampling like 4:4:4, 4:2:2, 4:2:1 and so on. The sampling mode 4:4:4 means that all three color component has the same sample rate and so it does not require any chroma sub sample. A sampling mode 4:2:2 means that both chroma components are sampled at half of luma sample rate. This sampling rate reduces the bandwidth of an uncompressed video.

Table 2.1: Frame resolution for different video format.

<table>
<thead>
<tr>
<th>Format</th>
<th>Frame Resolution</th>
<th>Pixels/frame</th>
<th>Frame size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIF</td>
<td>352 x 288</td>
<td>101376</td>
<td>2.4</td>
</tr>
<tr>
<td>4CIF</td>
<td>704 x 576</td>
<td>405504</td>
<td>9.7</td>
</tr>
<tr>
<td>720P</td>
<td>1280 x 780</td>
<td>921600</td>
<td>22.12</td>
</tr>
<tr>
<td>1080p</td>
<td>1920 x 1080</td>
<td>2073600</td>
<td>49.77</td>
</tr>
<tr>
<td>4k</td>
<td>4096 x 2304</td>
<td>9437184</td>
<td>226.5</td>
</tr>
</tbody>
</table>

The figure 2.1 shows the color component in a frame. It shows the RGB value of a frame and the correspondent YCrCb of the same frame.
Video transcoding is necessary for video broadcasting and requires a lot of bit stream to be transmitted over the network as explained in Chapter 1. Figure 2.1 shows the size of a single frame of a video segment. A video stream requires at least 25 frame/sec that means the network should be capable to transmit 553 MB/sec. A network with maximum bandwidth of 100 MB/second will require the video to be compressed by a factor of almost 6 times. A video could be downsize by removing unnecessary information and keeping only the relevant information. These two techniques are known as Redundancy reduction and Irrelevancy reduction. The redundancy reduces unnecessary information between frames and in a frame. Figure 2.2 shows block diagram of encoded video transmission over the internet.
In a Group of Pictures (GOP will be discussed in section 2.3.1) most of the frames have the same pixel values with a different position in the frame. This is called temporal redundancy. On the other hand a pixel in a location has almost the same values within one frame and this is known as spatial redundancy. Video compression uses these two redundancy reduction to achieve video compression. A video frame contains lots of unnecessary information that is not necessary for human mind as human eyes are only capable to receive a small information. This perception of human mind can be used to reduce information and compress a video.

2.1.1 Video Compression Steps and Types

Video compression uses the same preliminary techniques like temporal redundancy that we have discussed in the previous section. It uses the temporal redundancy that exists in the frames. In addition to the temporal redundancies, encoding process also uses the prediction method to detect the motion in the next frame based on the previous frame. The frames are divided into small blocks which are known as macro block (MB). Macro block is to find the nearest match to the previous frame. There are two different types of GOP depending on the frame sequence like closed GOP and Open GOP. GOP consists a set of frames with at least one Independent frame (I-frame), several Prediction frame (P frame) and/or Bidirectional frame (B-frame). The motion detection and motion prediction algorithms enables the encoder to avoid storing all the bits of a video frame. Figure 2.3 illustrate a video frame sequence in a closed GOP.
Figure 2.3 shows that a closed GOP is not dependent on any other frame from outside the GOP. This frame standard is known as closed GOP. When the frames in a GOP is dependent on a frame from the previous or the next GOP is called Open GOP. An open GOP contains one less P-frame than close GOP for same number of frames per GOP. Since a P-frame contains more bits than B-frames, so Open GOPs provide comparatively little bit better compression than closed GOPs for same structure and size. Figure 2.4 is showing the dependencies of frames between GOPs.

Figure 2.4 shows video segment of an open GOP. It shows that open GOP starts with a bidirectional frame b-frame. The last b-frames are dependent to the next I-frame and this characteristic makes harder to create open GOP.
2.2 Video Transcoding

The video transcoding software is capable to convert previously encoded videos from one video standard to another standard. In Ericsson Research developed several video transcoding software and 'C65' is newest transcoding software. This software can decode the coded video stream which was previously coded into one standard and then re-encoded into a different coding standard. During this transcoding process a video might change in several parameters like Bit-rate, Frame-rate format and resolution. Transcoding can also take place when a video stream requires to change the segment or GOP header in order to change the meta data of that input bitstream.

In the distributed architecture studied in this thesis, it uses transcoding in order to convert H.264 (AVC) video streams into H.265 (HEVC) stream. This process is slow compare to other transcoding for example MPEG to AVC [13]. A video transcoding can be simply built by an encoder followed by the decoder. First the decoder will decode the input video into bit stream and then the re-encoding will take place on the bit stream to produce the expected video standard. However it is necessary to mention that the whole re-encoding steps could be avoided to make the transcoding process faster. It is necessary to use the similarities between the input and output coding standard. Video transcoding is a long and complex process so if the output coding standard is going to have the same standard, we need to keep it similar and avoid re-encoding those similarities.
Figure 2.5: A simplified implementation of bit-rate encoder [2, 3].

Figure 2.5 shows the re-using of information that exist in the input video.

### 2.3 High Efficient Video Coding (HEVC)

High Efficient Video Coding (HEVC) or H.265 is the most recent video coding standard that has the comparable video quality with a bitrate reduce target up to 50% compare to previous standards such H.264. The goal of H.265 is to deliver larger and better resolution in moving pictures to phones with less bandwidth requirements than other video codec. HEVC introduced Coding Tree Unit (CTU) and Coding Unit (CU) instead of Macro block in order to increase thread level parallelism in the coding area which will be discussed in the later sub section 2.3.1. Figure 2.6 shows a scalable video encoder which was proposed by [4]. The paper propose Upsampling filter, Inter-layer sample prediction and Inter-layer Motion parameter prediction to achieve spatial and SNR scalability.
2.3.1 **Picture/Frame Partitioning**

The new video codec introduces Coding Tree Block (CTB) instead of Macroblocks in H.264 (AVC). HEVC also allows a block to be up to 64x64 pixels and each CTB can be split in a quad-tree structure like 32x32, 16x16 and 8x8 block as illustrated on Figure 2.7. For example, the Figure 2.7 shows that a 32x32 block could be divided into three 16x16 and four 8x8 block. These divided smaller blocks are knows as Coding Unit. [18, 19].

Coding Unit: Coding Tree Unit (CTU) is further divided into quad-tree in order to partition the CTU for further improvements on the calculation time and parallelization. The partitioning of CTU into CU is shown in Figure 2.7.

Prediction Modes: is the techniques used by HEVC in order to predict the motion of
the current frame compare to the previous frame. HEVC offers 34 intra prediction modes where the previous standard offered only 8. There are three different prediction models, like

- Intra-predicted coding unit (CU) uses the informations that are available in the current frame to get the information about the motion in the frame.
- Inter-predicted CU uses past and future frames to detect the motion vector of the pixels.
- Skipped CU uses the Inter-predicted CU method, however the difference here is that it uses the previous and next frames to detect the absence of motion so it can use available information from the previous and the next frames.

Figure 2.7: Coding Tree Block of HEVC from 64x64 to 8x8
Prediction Unit: A CU is divided using one of the eight partition mode that are shown in Figure 2.9. In the figure it is shown that the dimension of the prediction unit is defined by N and n. N is the half of Coding Unit and n is quarter of the CU. A 32x32 CU will have \( N = 16 \) and \( n = 8 \). A CU contains either one or two, intra- or inter- PU. It cannot have both inter or intra PU in one CU.

HEVC parallelization: HEVC introduced two approaches that are designed to en-
able parallel or multi threaded decoder. It uses two methods namely Tiles and Wavefront. Tiles: In this parallelization picture is divided into smaller rectangular groups of CTBs. Figure 2.10 shows a picture divided into nine tiles and the scanning order of CTBs. Motion vector detection and Intra-prediction is not performed on its boundaries and treat the block as a different picture. This way the tile do not need to wait for previous block to be done.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
</tr>
</tbody>
</table>

Figure 2.10: CTBs are divided to correspondent Tilses.

Wavefront: In Wavefront Parallel Processing (WPP) each CTB row uses its own thread. Wavefront parallelism reduces cross-thread prediction dependency. The Figure 2.11 shows how the WPP works with the number of available processors up to the number of CTB [20, 9]. However the disadvantage of WPP is all the thread cannot start at the same time which is shown in the figure below. This problem is solved by Overloaded Wavefront (OWF). In this way, when a processor completes computing a row it does not have to wait for all the processor to finish. This free processor can start with the next picture.
Figure 2.11: Wavefront Parallel Processing of CTBs in parallel.
3 Cloud Computing

This chapter provides necessary information about Cloud computing and Ericsson research cloud. The main target of this thesis was to use a distributed system in order to minimize the latency of complex calculations delay of the video transcoding. The distributed system used for this research was Ericsson Research cloud. Later sections of this chapter will show how should we configure Virtual Machines (VMs defined at section 3.1) in the cloud. This chapter will provide potential improvements for the cloud computing to make it user friendly and more controllable from the host side. Since Ericsson research cloud is using Openstack platform, a section will be dedicated to discuss the advantages and disadvantages of this platform. It will define the cloud computing service models and the usability and complexity.

3.1 Cloud Computing

In a cloud computing system a set of hardware, software, storage space and network system is established in manner so that a user can use them to create powerful system for processing complex tasks, store data and create software. Depending on the deployment models a cloud could be classified in four classes et. al. [21]

- Private Cloud: This kind of cloud provide services to single organization.
- Public cloud: It is provisioned for the open use by the general public.
- Community Cloud: This cloud infrastructure is used by a group of organization or people who share information or a common interest. A big use of this kind of cloud system is in Security development.
- Hybrid cloud: This cloud infrastructure is a mixture of two or more of the previous cloud system.
Public cloud is available to the people with an access key and private cloud is available for only specific people in the same network. It is available to the user through a web user interface. There are several definitions that explain cloud computing [22, 23]. A good explanation of cloud computing is given by Foster [23] which emphasize more on the economical scale and user-friendly perspective.

A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.

When we compare the definitions [23] with other definitions such as Armbrust et al. [24] discusses more about consumer-oriented perspective when they list three key aspects that are new in cloud computing [25]. This key aspects are:

1. Almost unlimited number of Instances ready to use for the users.
2. A strong community of cloud users and commitments.
3. Easy terms and conditions when it comes to uses for shorter period.

### 3.1.1 Classification of Cloud computing

At the introduction of this chapter we discussed about virtual machines or Virtualization of the software which means an operating system with all the required softwares and applications, are ready to use. A virtual machine allows sharing among multiple users. Cloud computing implies Service Oriented Architecture (SOA) includes three services models which is:

- **Infrastructure as a Service (IaaS):** This service provides virtual CPUs, memory, storage device and internal network depending on the user demand. Usually a service provider split, assign and dynamically resize the resources flexibility to build a system that is requested by the user. IaaS providers allows the user to access that virtual resources over the network. IaaS is much more suitable compared to other cloud services because it has individual data storage and servers to deploy and run video transcoding application. Another big advantage of IaaS is that the user can perform the load balancing and resource allocation for this system and it is also cheaper compared to PaaS. A good example of Iaas is Amazon Web Servers (AWS).
• **Software as a Service (SaaS):** provides services to the end-users or consumer which are developed and available for cloud service users. This kind of infrastructures do not allow users to access servers, platform or storage. In other word the clients cannot see the details implementation of the infrastructure compare to Iaas and Paas. These applications are accessible to the consumers through web UI or specific program interface. Web-email service or web-base shops are good example of this cloud infrastructures.

• **Platform as a Service (PaaS):** is an infrastructure where the service providers offer platform, Application Programing Interface (API) and resources to the client so that they can create and build application and services with their own requirements. The users do not have control over the PaaS infrastructures like servers, networks, storage or OS rather they have control over the running applications and configuration of the host-environment. A great advantage of this infrastructure is that it does not concern about the resource availability and it is much easier to scale up for maximum utilization of the hosted application. Google app engine uses PaaS infrastructure.

Figure 3.1 shows the hierarchical structure of SOA for cloud computing resources. The figure shows how each of this structure is different based on the resource visibility and service abstraction. For example IaaS has more resource visibility than Paas and Saas.

![Figure 3.1](image.png)

Figure 3.1: Cloud infrastructure for Service oriented architecture (SOA).
3.2 Openstack

Our distributed computing platform is the Ericsson research’s cloud which uses Openstack cloud software as a IaaS platform. Openstack is an open source cloud computing platform to build both private and public cloud. This Openstack project was initially started in the middle of 2010 with collaboration of Rackspace and NASA. These two organization contributed initial codes to provide cloud related software to the users. The Openstack comes with all the required functionality like virtual computing, data storage and network structure for IaaS. Openstack has been used by more than 150 companies like HP, Ericsson and so many cloud computing researchers for its wide flexibility on the cloud functionality and infrastructure. It has been developed by crowd sourcing and new versions are released in every six months. It is getting popularity and a lot of organizations are joining by agreeing on developer contract and they can contribute to the code. We are going to give a description for the Openstack architecture here:

3.2.1 Architecture of Openstack Nova

Openstack has produce a cloud computing middleware fabric controller. It supports several virtualization such as KVM, UML, XEN and HyperV using QEMU emulator [25, 5]. The cloud controller is the main controller of the Openstack Nova and this communicate and control other modules in different ways which is shown in the Figure 3.2.
The description and tasks for each block is given below.

- **Cloud Controller** is the main control unit in the architecture. It divides the user request and collect the result. It is responsible for the global and internal communication.

- **API Server** Openstack provides two sets of HTTP servers for APIs such as 'Amazon EC2API' and Openstack 'OSAPIs'. These APIs sends request to the cloud controller from the public clienest.

- **Admin Client and Authentication manager** is also connected to the cloud control. It is responsible to provide authentication and authorization for the services at Openstack. Authentication manager communicate with Admin client through
method call. There are several authentication methods and management in Openstack such as user-credentials and role based access control. User-credentials takes id, name, access key, secret key and administrative status and Role based access control to select the role of the user (admin, manager, developer or even user) in this project. It will provide different key for different user status.

- **Compute Controller** is responsible to provide necessary servers for the computational tasks. It could have multiple number and scheduler controls these servers.

- **Scheduler** is the most important controller after Cloud controller. It is responsible for the resource allocation, deallocation and scheduling the Computer controller servers. It communicates with controller using AMQP messages.

- **Object store** responsible for data storage and sending.

- **Volume controller** stores block-level storage for the compute servers.

- **Network Controller** manages the network for the Virtual Machines and it also communicate with external or public network.

### 3.3 Configuring Openstack for Apache storm

For our work we did not need to create or install Openstack for cloud computing. It was already done by Ericsson research team. We needed to configure Openstack so it is able to run Apache storm. Apache storm is an open source distributed realtime computation system which will be described in Chapter 4. Although there are a few Openstack clouds where storm is installed in Data Processing section. However, Ericsson has not introduced Storm to its research cloud. So this project needed to install apache storm which is explained in section 4.3.3 in Ubuntu and upload that image. Later sections describe how to create and build ISO images, instances. Figure 3.3 shows the available resources for our video transcoding project at Openstack.
3.3.1 Create Images

Uploading an ISO image to OpenStack is easy to do. Once all the steps are completed in section 4.3.3 then it is a simple step to upload that image to our project at Openstack. Figure 3.4 shows the required information in order to upload the image. It is possible to upload an image from HTTP url, .ZIP or .tar.gz. In our case we uploaded the *.vdi file from the virtual box folder location. The cloud image uploader requires information regarding the image that are being uploaded. In our case we used Ubuntu(x64) 15.10.
3.3.2 Create Security Group and Network

Once the image is uploaded we need to change the security group. There is a default security group which can be edited or we can create a new security group. For this project we used the default security group and added all the other ports that are required for the Storm. Figure 3.5 shows a small part of the security group with the port numbers that are added to the security group.
Figure 3.5 is showing a part of the allowed port numbers in the default security group. These port numbers are allowing the instances to communicate on specific ports. All the ports we opened in this security group has a specific task like port:22 for SSH communication and other ports are for Apache storm communications.

### 3.3.3 Openstack Network

Openstack also requires to create network and virtual router for the instances in the cloud. Virtual router uses subnet to communicate with the External network for example LAN. Figure 3.6 shows the internal network for the video transcoding project.
Figure 3.6: The network topology in the cloud with all instances.

Figure 3.6 is showing network topology and how each of the instances are connected to each other. The external communication will end on the net-ext04 and the virtual network will manage the internal communication in the cloud. The yellow bare is the subnet that we created for the storm instances.
3.3.4 Import Key Pair

To be able to communicate to the created instances we need to use SSH. In order to do that we need to create a key-pair in the local computer and import it from Openstack ‘Access and Security’. Once a key-pair is imported we can use it for the created instances. Figure 3.7 shows all the key-pairs that are imported from different host computers. We need to use one of the key to connect with the cloud instance from the host.

![Key Pairs in Access and Security](image)

Figure 3.7: List of imported key pairs in Access and security.

3.3.5 Creating Instances

It is possible to create instances with default security groups and settings. However for our project it is necessary to adjust the setting and add extra ports so that all the instances can reserve storm ports that are configured in the configuration file. As we have prepared all the storm related ports and security options in Openstack UI, we can now create instances. Instance creator window has five steps and we need to set options for the first three and the others can use the default settings. Section 3.8 shows the instance launcher for Openstack.

- Details: First tab is Details which takes information for the instance that are about to be created. It requires zone name, instance name, size of the instance and number of instance to be created. Once the size of the instance are created with Flavor, all the properties like VCPU, Disk and memory will be automatically selected. Section 3.8 shows the the required information for the overview to launch an instance.
• Access and Security: is responsible to set up the required Key-pair and the security group for the instances. Here you need to select the key pair that was imported in section 3.3.4. As we have only one security group we do not have to consider the security group. Section 3.9 show the settings we selected for our instances.
• Network: tab gives options to select the network and the router that were created for the instances. It is really important to select the Subnet that were created. Section 3.10 is to choose the network where the instance will be connected.
4 Apache Storm

This chapter is about the background of Apache storm and how it is developed. It also discusses about the parallelization techniques and how parallelization works in storm. This chapter explains how we can configure Apache storm for different operating system. In our project we use both Windows-7 and Ubuntu.15.10 operating system. We use Windows-7 for local computer measurement and Ubuntu will be used for cloud computing.

4.1 Apache storm

Apache storm is an open source distributed real-time computation system which was developed using Java and Coljure based framework. Coljure is a Lisp based functional programming language. The advantage of this language is that, it can run on Java Virtual Machine (JVM). The beginning of Storm was at Backtype by Natahn Marz and later it was acquired by Twitter in 2011. Since then storm has been used by Twitter to process large amount of data from the users in real-time. Storm gets more popularity compared to Hadoop and other batch-processing framework because of its real-time capability and robustness. Apache storm has been used by several big corporation like Twitter, Yahoo, Yelp and Spotify [26, 27].

The advantage of using storm is that it has feature like fault tolerance, horizontal scalability, robustness which guarantees data processing and the support of several programming language. Apache storm has an option to re-balance the work load among workers as soon as a new worker joins the cluster. This option is very useful when it comes to small distributed system like home network cloud as discussed in section 1.3. Fault tolerance and guaranteed data processing comes into action when one of the worker fails to process the data then the host server distributes the task among the available workers. Apache storm cluster runs jar files and the workers in storm supports multiple programming language like C++, Java, Python, etc. The most attractive
feature of Apache storm is the higher performance in real-time. One benchmark has shown it can transfer approximately one million tuples per second [27, 28]. In the later section we will discuss about parallelization techniques for Apache storm.

### 4.1.1 Apache Storm Architecture

The architecture of Storm is very simple and it consists of two clusters, one for Storm cluster and another is Zookeeper cluster. Storm cluster consists Nimbus, supervisor and UI daemons. It works with master and slave architecture where master node runs the Nimbus and UI daemons and slave nodes run supervisors. Zookeeper cluster lists all Zookeeper servers running in the network. Figure 4.1 shows the architecture and the components of a Storm cluster.

![Apache Storm Architecture](image.png)

The storm cluster allows only one master node which runs the compulsory Nimbus daemon and an optional webUI. The webUI is responsible for providing web based User Interface (webUI). This UI provides cluster information like number of available workers, zookeeper, settings for the masters and workers. It also shows the information for the running topologies. Each slave node runs supervisor daemon and optional
logviewer. The supervisor daemon run the listed workers and an optional logviewer. The supervisor register to the zookeeper. Nimbus communicate with the zookeeper to get the list of worker.

The main three part for a storm cluster are described below [28].

1. **Nimbus**
   Nimbus is a component of the master node in the architecture of the storm. This Nimbus daemons runs using the configuration files (discussed in section 4.3) which includes the host name and ip address for zookeeper servers. Nimbus make sure all the zookeeper servers are up and running. Nimbus also collects data form the supervisors through zookeeper servers. When a topology jar file is uploaded, Nimbus daemon receive that jar file and divide the task according to the scheduler. Storm Nimbus store its work in Zookeeper unlike Hadoop. This stateless Nimbus avoid the one point failure if the Nimbus daemon fails while a job is running.

2. **Supervisor**
   The Supervisors daemons are the one who list all the workers from the configuration file. Supervisor daemons run a Java Virtual Machine (JVM) for each worker process. It reports all the worker states to the zookeeper node and keep a log file for the running task for each worker. A supervisor is the slave in a storm cluster who executes the task distributed by the nimbus daemon. Each slave node runs one supervisor daemons and each supervisor can run one or more worker per computer. The default number of worker per supervisor is four.

3. **Zookeeper**
   Zookeepers are the third major component of the storm cluster. This zookeeper cluster run an odd number \((2n + 1)\) of zookeeper node. Zookeeper node is the one who manages the communication, task management, task re-balance between storm nimbus and supervisors. It is also responsible to store the states of Nimbus and supervisors so that when one of these Nimbus or Supervisor fails, zookeeper can restore them to the previous state as if nothing happened. This is why the states of the nimbus and supervisors are continuously stored in zookeeper. Zookeeper also reports to the UI for all the supervisors health, so that a user can check present state for each of the worker.
4.2 Component of Storm

The components of storm are important part of the storm parallelization. It shows how the tasks should be mapped and how it will be connected for the workers. To do a real-time computation on storm we need to create topologies where a user mapped the connection between two major components *Spout and Bolts* and the part where these communication is built is called topology. In other word topology is the one who connect the workflow from input to output. It is also responsible to configure parallelism described in 4.2.3 and setup the required number of workers. Figure 4.2 shows the data flow between spout and bolts.

![Figure 4.2: A storm topology Example.](image)

4.2.1 Spout

The spout in a storm topology is a source of the storm topology. It is responsible to read or take input data from an external source like database, distributed file system and does some operation if it is necessary. A spout converts the input data into storm stream for Bolts which is discussed in the next section. The tuples from spouts can be divided in two groups such as reliable and unreliable spouts. The reliable tuple has the ability to reply for a failed tuple and an unreliable tuple discard its tuples as soon as it emits to the bolt. Figure 4.2 has two spouts which can read from two different sources. The most popular method of implementing a spout is `nextTuple` which emits the next
tuple to the storm stream for the connected bolts. Figure 4.3 shows that nextTuple emits the new sentence that come to the spout.

```java
public class RandomSentenceSpout extends BaseRichSpout {
    SpoutOutputCollector _collector;
    Random _rand;

    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        _collector = collector;
        _rand = new Random();
    }

    @Override
    public void nextTuple() {
        Util.sleep(100);
        String[] sentences = new String[]{"the cow jumped over the moon", "an apple a day keeps the doctor away", "four score and seven years ago", "snow white and the seven dwarfs", "i am at two with nature");
        String sentence = sentences[_rand.nextInt(sentences.length)];
        _collector.emit(new Values(sentence));
    }
}
```

Figure 4.3: A storm topology Example.

### 4.2.2 Bolt

The actual process of parallelization and job processing is done by using storm Bolt. The Bolt can take its input from either one or multiple Spout or Bolt. Similarly it can produce multiple output stream for other bolts. It has the capability to aggregate storm streams, filter, join and execute other functions [26]. Figure 4.2 has three green bolts and one yellow bolt. The green bolts take input tuples from spouts and process and yellow bolt aggregate all the storm stream from green bolt. The Bolt has two major part, one is called execute which process the data received from previous spout or bolt and the second one is called prepare which is responsible to declare output field for the next tuple.

### 4.2.3 Parallelization in Storm

Apache storm has the biggest advantage when it comes to parallelization and distributing its tasks. The architecture of Apache Storm offers multiple levels of parallelism.
Storm identifies three main entities that are used to run a topology in a storm cluster. The greatest advantage a storm cluster provide is that, it is possible to re adjust the combined parallelism of a running topology with a command *storm rebalance*. Figure 4.4 below shows the setting of different level of parallelism in storm.

![Figure 4.4](image)

Figure 4.4: A sample code for showing different level of storm parallelism.

1. **Supervisor/Worker** is the external parallelization for a storm cluster. The number of supervisor depends on the total number of node that has supervisor daemon running. In storm, it is possible to add as many supervisors and workers to the cluster. However, the number of required supervisor and worker will be decided by the topology builder. It is even possible to add or remove worker during runtime which is really important for video transcoding to avoid over flow or under flow of the data. Each supervisor gets multiple workers (usually four workers) from the configuration file. Workers run JVM which is capable to run Executors. From figure 4.4 it is visible that we set number of workers to 2 and figure 4.5 shows the two workers and the tasks.

2. **Executor** Executors run one or multiple threads within a JVM process. The number of executers depends on the parallelism hints when we define bolts. For example the green bolt has two executors and the yellow bolt requires 6 executors.

3. **Task** The number task is how many task will be spawn per executors or thread.
Each executor will run one task serially. For example, we have not define the number of task for the yellow bolt so it will run six task in both of the workers. The green bolt has 2 executor but the number of task is 4. This will make sure that each executor runs 2 task per executor.

Figure 4.5: Different level of parallelism in Apache storm based on figure 4.4.

4.3 Configuration of Apache storm

Our distributed system uses apache storm for the distribution of the tasks. So we needed to configure an apache storm cluster with multi node where the cluster will work with both operating systems MS Windows and Unix. In a single node installation all the Configuring apache storm has different steps with different operating system, however, the pre-installed software requirements are the same. To install storm we need to make sure that all the nodes have installed Java JDK1.7.045 and Python 2.7.
or an above version. Once these development kits are installed we also need to keep a running Zookeeper server for single node or storm cluster. Storm relies on zookeeper cluster to perform the coordination between Nimbus and the Supervisor [29].

4.3.1 Configure Zookeeper

Zookeeper cluster is widely useful with other distribution like Hadoop. It is possible to install by using Cloudera CDH4. However, we are going to configure the zookeeper manually. In order to do that, we downloaded zookeeper-3.3.6 and changed the configuration file (zoo.cfg). Once the zoo.cfg is configured we need to make sure that zookeeper runs in the background during the system startup. In windows system zookeeper will keep a command prompt to show the server status. But in the Ubuntu we need to check the zookeeper status by checking on the port. Figure 4.6 shows the number of messages and connected supervisors with the Zookeeper clusters.

![Zookeeper status](image)

Figure 4.6: Zookeeper status and connected storm components with zookeeper cluster.

4.3.2 Configuring storm for Windows

Configuring a storm on Windows operating system was a complicated task with the previous versions, but as of storm-0.9.* Apache storm is easier to configure for windows systems. First we need to download storm and configure storm.yaml file in the configuration folder named `conf`. The storm.yaml folder contains settings for storm
nimbus, supervisors and IP addresses for Nimbus host and zookeeper servers. Apache storm has a ‘defaults.yaml’ file which has all default settings for the storm cluster. The main functionality of the ‘storm.yaml’ file is, it overwrites the ‘defaults.yaml’ so that it can support all our topology requirements. Figure 4.7 shows a part of a storm.yaml file. The next step is to add environmental variable for Windows. After completing all the steps, we need to run Nimbus and UI in the host and the Supervisors daemons to the workers. Once the daemons are running successfully without any error we can get the cluster info and setting by using webUI.

```yaml
### These MUST be filled in for a storm configuration
storm.zookeeper.servers:
  - "147.214.99.215" # Nimbus (Development computer)
  - "147.214.99.215" # Nimbus host: 147.214.99.215
storm.zookeeper.port: 2181
nimbus.host: "147.214.99.215"
nimbus.childopts: "-Xmx1024m"
nimbus.thrift.max_buffer_size: 20480000
nimbus.thrift.port: 6627
# Time out for SUPERVISOR at NIMBUS
nimbus.supervisor.timeoutsecs: 180

supervisor.slots.ports:
  - 6700
  - 6701
  - 6702
  - 6703

storm.local.dir: "C:\\tmp\\storm_working_dir"
storm.scheduler: "DemoScheduler"

supervisor.schedulers.meta:
  name: "dash-supervisor"
  worker.childopts: "-Xmx768m"
# Default is IMB - 1048576
worker.heap.memory.mb: 1024
# worker.heap.memory mb: 768
storm.messaging.transport: "backtype.storm.messaging.netty.Context"
```

Figure 4.7: A part of configuration file for storm.yaml.

Figure 4.7 shows the storm.yaml file which contains information for Zookeeper, Nimbus host and settings, supervisor settings and worker ports. Zookeeper servers contains the ip addresses of all the zookeeper running for the nimbus. The Nimbus setting requires the ip address of the Nimbus host and other settings could be left with the default value. The supervisor and worker settings set up the worker ports and require memory for each of the workers.
4.3.3 Configuring storm for Unix

Installing storm in the Unix based operating system requires all the steps we followed in 4.3.2 with an extra application named ZeroMQ. ZeroMQ is a library that extends the standard socket interfaces with features traditionally provided by specialized messaging middleware products. Storm relies on ZeroMQ primarily for task-to-task communication in running storm topologies [30].

4.4 Scheduling

Apache storm has a default scheduler which usually distribute the tasks evenly among the available workers. From the discussion in section 4.1.1 we mentioned that Storm Nimbus takes the submitted topology by the user. Nimbus usually distribute the tasks for the workers as long as it is free and a user does not have control over that task distribution. In some cases it is necessary to make sure that a particular topology tasks are located at a specific Supervisor. This is why I need to implement a pluggable scheduler for Apache Storm. The Storm versions with higher than Apache Storm-0.8.0 allows user
to create user defined pluggable scheduler.

The pluggable scheduler is created by user and written in Java. After building the java project, it will create a jar file which will be stored in storm/bin folder of the host computer. The name of the scheduler will be edited on storm.yaml file as shown in figure 4.7. Each of the supervisor will also have a different name. When a nimbus is up and running it will get the list of available supervisors, then a topology with specific spout or bolt name will be distributed to a specific supervisor.

In our project we used a pluggable scheduler that we will show in 5.3. Pluggable scheduler is necessary when we need to make sure that a component of a topology will be distribute to a specific supervisor. In our project we needed to make sure that a specific supervisor is responsible to provide the file and collect the output files. This is why a pluggable scheduler was necessary in our project.
5 Topology Configuration

This chapter explains about the topology that we designed for video transcoding in a distributed system. It will discuss about the spouts and bolts that are created for the topology and the algorithm for each component of the topology. It will discuss the background of Apache maven and how to create jar files to run on a storm cluster.

5.1 Current Topology

The initial topology that we designed consists of multiple components like Spout and Bolts. Initial topology contained four components such as one spout, two obligatory bolts like transcode bolt and dash bolt and one optional bolt (MP4Muxing bolt).

The topology has three main parts namely Parsing arguments, creating the components and topology settings for cluster. Figure 5.1 shows the storm data flow for the initial designed topology. The description for each of the component is given below.

![Data flow for the initial topology.](image)

1. In the previous chapter 4.2 we discussed that Spout is responsible to collect data
from a location. The spout in `EncodeVideoTopology.java` is responsible to do more than just reading the video segments. After reading the input data spout converts them into tuple to put them in the storm stream. The Spots is capable to take already segmented video. However, for real-time video transcoding, there is another spout which will take the raw video file and segment them. In every component we need to map tuples. Mapping decides what tuples will emit for the next component. Figure 5.2 shows the task execution steps for Spout in topology.

![Data flow for the initial topology.](image)

Figure 5.2: Data flow for the initial topology.

2. Transcode Bolt is the first bolt in the topology that receive the videos from the
spout and it will transcode them into required video coding standard. This step is long and the most time consuming process. That is why it is better to use more available workers for this component.

![Diagram](image)

**Figure 5.3: Data flow of the Topology with number of representation=2.**

3. **MP4MuxBolt** is an optional bolt in the topology. It depends on the user whether muxing is necessary for the transcoded segments. If muxing is true then transcoded bitstream will go to the dash bolt through MP4MuxBolt or it will go directly to dash bolt. Muxing bolt uses FFMPEG to convert the transcoded video such as H.265 bitstream to MP4 files. If it is not necessary to play the video through Dash player then it is recommended to skip this step.

4. **Dash Bolt** is the component which collects all the video from transcoder or from MP4MuxBolt depending on the muxing option. If muxing is true then MP4Mux takes the input from the Transcode bolt Dash which is responsible to collect all the format or representation from the input bolt

### 5.2 Multiple Camera

The tasks for our project was to update our topology into a dynamic topology which will work with multiple cameras. The dynamic topology will have a location where video segments from all cameras will be stored. The topology should be able to read
segments and transcode those segments. The topology will increase the number of components proportionally to the number of cameras. Here we have experimented with three cameras with 0 or 2 number of additional representation. Figure 5.4 shows a topology for three camera and two representations. The representations are responsible to transcode the video into different resolutions. This topology has created three Spouts for each camera and three transcode bolts for one regular representation and two representations mean two different resolution.

Figure 5.4: Topology for multiple camera and two representation.
5.3 Scheduling

Section 4.4 discusses about how pluggable scheduler functions in Apache storms. That section gave settings for scheduler in \textit{storm.yaml} and how it should be configured in storm nimbus. This part will show how we created a pluggable scheduler for our topology. The scheduler is designed in a way so that all Spouts and the dash bolt are under one worker namely \textit{special-supervisor}. This special supervisor will load store all segments form cameras so that Spouts can read the video files. Secondly, the dash supervisor is the one who receive the transcoded files after muxing and dashing. Dash supervisor collects all the representation and then store them to the HTTP server. When the dash player plays those segments we need to know the location of the dasher. This is why we use pluggable scheduler to keep the nimbus and dasher under a specific supervisor.

```java
public class DemoScheduler implements IScheduler {
    private static final String SITE = "site";
    static final String TOPOLOGY_NAME = "c65-demo-production";
    static final String[] SPOUT_BOLT_NAMES = {"spout0","spout1", "dash"};
    static final String[] SPOUT_SUPERVISOR_NAMES = {"special-supervisor",
        "special-supervisor","special-supervisor"};
}
```

Figure 5.5: Mapping Spouts and Bolts to a specific supervisor.

5.4 Maintaining a Topology in Storm Cluster

The topology that we configured in the previous section does not only define the data flow from the source to destination, it also include configuration for the cluster like the number of workers, number of ackers, spout spending, worker timeout etc. Topology uses \textit{Conf} class to configure topology. Figure 5.6 shows the configuration part of a topology.

```java
Config conf = new Config();
conf.setNumWorkers(20);
conf.setMaxSpoutPending(5000);
StormSubmitter.submitTopology("mytopology", conf, topology);
```

Figure 5.6: Mapping Spouts and Bolts to a specific supervisor.
The important settings of the topology configuration are explained below:

- Number of workers are set in the topology which let the cluster know the number of workers this topology will require to run in a storm environment.

- Topology acker executors is a part of the reliability model for storm architecture. The number of executors will monitor the tuples that are being processed and completed.

- Topology maximum spout spending setup, the number of maximum spout tuples that can be pending in a single spout task at a single time [31].

- Topology message time out settings, set up the time in seconds to wait for a tuple to be processed before it is counted as a failed tuple.

5.4.1 Apache Maven

Apache maven is an accumulator which is originally created to simplify the building process of the project. Apache Maven collects the project information and creates a jar file to share across several project. Figure 5.7 shows the dependency of configured topology. When we compile the pom file it will download all the dependency and convert them into a jar file. This jar file will be used in the storm cluster. Storm cluster is able to extract the jar files and run the topology without knowing about the programming language.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive>
      <manifest>
        <mainClass>com.path.to.main.Class</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

Figure 5.7: Apache Maven pom file.
5.4.2 Run a Topology

Once all the jar files are created using Apache maven we can upload the jar files to the cluster through storm submitter. Storm submitter will require three arguments like topology name, configuration for cluster and topology as shown in figure 5.6. After completing all the configuration of the jar files we can run the program by using the command

\[\text{storm jar \ast.jar arg1 arg2 ..... argN}\]

After running this command, if the cluster can configure the requirements of the topology then webUI will show topology name in the webUI. From the webUI it is possible to see the topology and and the data flow between the components and the information like the number of messages between the components that is shown in figure 5.8.

![Topology summary with the information of a submitted topology.](image)

Figure 5.8: Topology summary with the information of a submitted topology.
6 Results

This chapter will present the results for the implementations that has been done through the previous chapters. It will show how we collected the results from the Apache storm in the local computer and in the cloud instances. Outside of the collecting results, this chapter will provide measurements for the network delay, jitter and discuss about the maximum parallelism. It will show the improvement of the cloud computing over the local storm data.

6.1 Data Collections

Apache storm creates log files for each daemons to make troubleshooting for the user. We can use these log files to store the output result and the required processing time in each worker. However this is a simple task when we have a local host or single node storm. This method of collecting data becomes more complicated when we have a cluster with a large number of workers.

6.1.1 Topology Information

In this situation webUI daemons in Apache storm becomes very useful. The webUI shown in figure 4.8 shows the summary for Cluster, Topology, Supervisors and Settings as explained in 4.3. The Topology summary shows the running topologies in the cluster and it is possible to open each topologies to check the components status and time to process and execute.
The previous Figure 6.1 has one spout and five bolts. The spouts show the number of emitted tuples and messages. The bolts show same thing as well as other information like Executed latency, Process Latency and Capacity. Process latency shows the average time needed for a bolt to wait for the tuples to receive. The bolt does not start processing the tuples until it receives minimum number of tuples from the sources spout or bolt. It displays capacity information which shows maximum usage of the workers for the last 10 minutes. When the capacity is more than one in topology details it means the topology requires more workers.

6.1.2 Topology Visualization

Another useful feature of apache storm is to use Topology Visualization. Topology visualization shows how each of the components are connected to each other with processing latency and capacity. Figure 6.2 shows the communication between a spout to dasher bolt. Each bolt shows the processing latency and the capacity in the communication link for the bolt. Higher capacity means the parallelization has reached maximum point. If Capacity is close to one the bolt will change color to yellow and when it is more than one bolts become red.
Figure 6.2: Topology visualization of one camera with three representation.

6.2 Results for Local computers

Our previous topology was built for MS Windows system so we configured Apache storm in Windows. Windows or the local cluster was configured with a two Windows-7 systems with 8 workers. We ran 720p video segments with H.264 standard. The Table 6.1 shows the execution time in Windows cluster.
Table 6.1: Video transcoding with multiple representation and camera in Windows.

<table>
<thead>
<tr>
<th>Representation = 1</th>
<th>Execution (sec)</th>
<th>Representation = 3</th>
<th>Execution (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2</td>
<td>1</td>
<td>2.1</td>
</tr>
<tr>
<td>1</td>
<td>8.6</td>
<td>12</td>
<td>18.29</td>
</tr>
<tr>
<td>2</td>
<td>9.2</td>
<td>13.29</td>
<td>21.4</td>
</tr>
<tr>
<td>3</td>
<td>7.7</td>
<td>15.11</td>
<td>12.5</td>
</tr>
</tbody>
</table>

The figure below shows the latency and capacity of a video topology in Windows cluster.

![Topology visualization of one camera with one representation.]

Figure 6.3: Topology visualization of one camera with one representation.

From Table 6.1 it is visible that when we are running the topology with one camera the execution time is close to one second. However, the time increases dramatically with the higher number of representation. Higher number of representation is causing the topology to run slow specially when it comes to read video segments with multiple cameras.
6.3 Cloud Computers

Ericsson research cloud uses Nova architecture for the cloud computing. A user can choose different size for the flavor or sizes for the virtual machines (VM). The simulation conducted mostly with the m1.medium and m1.large virtual machines. The difference between medium and large VMs are mainly in the VCPUs, RAM and storage disk. For medium VMs, Openstack allocates 2 VCPUs, 4GB RAM and 40GB data storage where large VMs can allocate 4 VCPUs together with 8 GB RAM and 80GB disk memory.

6.3.1 Measurements for 720p videos

The storm cluster in the cloud is run in Unix operating system and we were able to create six instances with 4 workers on each of them, with a total number of 24 workers.

Table 6.2: Video transcoding latency with multiple representation and camera in cloud for 720p.

<table>
<thead>
<tr>
<th>Representation = 1</th>
<th>Representation = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Camera</td>
<td></td>
</tr>
<tr>
<td>1.4 sec</td>
<td>1.3 1.4 1.5</td>
</tr>
<tr>
<td>1.8</td>
<td>3.6 4 4.7</td>
</tr>
<tr>
<td>1.7</td>
<td>4.4 5.2 3.6</td>
</tr>
<tr>
<td>1.9</td>
<td>4.7 5.6 5.8</td>
</tr>
</tbody>
</table>

The visualization in figure 6.4 shows process latency for three cameras. In this visualization it is visible that spout is taking a longer time to read input video segments. This phenomenon becomes critical when we are using HD and 4K videos.
6.3.2 Measurements for 1080p (HD) videos

Table 6.3 shows the execution latency for HD videos with the same cluster. It has much higher latency in the spouts because of the higher resolution in the videos.

Table 6.3: Video transcoding latency with multiple representation and camera in cloud for 1080p.

<table>
<thead>
<tr>
<th>Representation (sec)</th>
<th>Execution (sec)</th>
<th>Execution (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation = 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Single Camera</td>
<td>6.9</td>
<td>3.5</td>
</tr>
<tr>
<td>Multiple Camera</td>
<td>1</td>
<td>15.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>27.3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>24.35</td>
</tr>
</tbody>
</table>

Figure 6.5 changes the transcoding bolt color according to the value of capacity explained in 6.1.2. Here the capacity is more than 1 thus the bolt changes color to Red. If the bolt color does not change, the topology must rebalanced with the higher number of workers.
6.3.3 Measurements for 4K videos

Measurement of 4K videos with the same settings are complicated because of the video file size. Previously we used two seconds segments, however for 4K segments we need to use one second segments. The results are shown in table 6.4

Table 6.4: Video transcoding latency with multiple representation and camera in cloud for 1080p.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Execution (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Representation = 1</th>
<th>Execution (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Camera</td>
<td>9.2 sec</td>
</tr>
<tr>
<td>Multiple Camera</td>
<td>1  8.21</td>
</tr>
<tr>
<td></td>
<td>2  10.73</td>
</tr>
<tr>
<td></td>
<td>3  20.99</td>
</tr>
<tr>
<td>Representation = 3</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>18.29</td>
</tr>
<tr>
<td></td>
<td>30.11</td>
</tr>
<tr>
<td></td>
<td>18.29</td>
</tr>
<tr>
<td></td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>26.7</td>
</tr>
</tbody>
</table>
7 CONCLUSION AND FUTURE WORK

This last chapter will present the conclusion of the thesis work. It includes the goal of this thesis and what we have achieved throughout the project. A section in the later part of this chapter will be dedicated for the future work explaining what kind of improvements we can bring to this project for even better results.

7.1 Conclusion

The main goal of this work was to investigate distributed systems in order to lower video coding delays. Familiarize with the configuration and parallelization of Apache storm and use real-time data processing software to lower delays in the transcoding. To lower the video transcoding delay we used GOP based segmented video files. The project tested with three different video resolutions to find better transcoding time for each of them. The results we discussed in chapter 6 shows that the apache storm provides the best results for 720p segments. The cluster showed biggest improvements for this video files compared to the other resolutions.

A successful apache storm configuration for different operating systems. It was also necessary to build a pluggable scheduler. This thesis explored different level of parallelism offered by the Apache storm. This project work also includes testing the storm reliability and robustness of a storm cluster with multiple topologies running at the same time. Nimbus was capable to identify supervisors through Pluggable scheduler and distribute the respective task to specific supervisors.

The configuration for the distributed platform like Ericsson research cloud to run Apache storms. It required to configure VMs and run in Openstack. It was necessary to enable resource allocation and deallocation for the VMs through exploring IaaS. The infrastructure was capable to increase resources with the demand of topology.
7.2 Future work

The future work for this project is to bring changes and improving the cluster to get better results in the transcoding. The thesis work explore further possibilities to improve delays in the video coding. Few of the possibilities are explained here.

Topology Improvements  The improvements in the topology side will be

- Spout: The results in section 6.3 shows that spout is taking a longer time to read the files. Which is causing timeout failure in the transcode bolt. Transcode bolt waits for a specific time for the next tuple to arrive and if the tuple from the spout fails to reach within that time the transcode bolt stops the communication and that causes failed spout messages. We need to increase the heap size so that the spout can read the video files. Increase in the number of ‘Parallelism hints’ and this increases the number of executor. With the higher number of executers the storm will be able to run the spout with multiple tasks serially in the spout.

- Real camera: The topology is capable to encode raw video file from file. The spout is also changed in a manner so that the spout waits for the next segment to arrive.

- Bitrate and codecs: The topology is also capable to run with different bitrate, however, the muxing and dashing requires improvements to run different codecs.

Improvements on the C65  The improvements on the c65 is necessary when it comes to use Unix environment. When we are running the transcoder in local Windows computer, it is capable to use both multithreading option and GPU acceleration. However it is not the case when we are using the cloud VMs. Enabling multi threading for the unix would be one area we can explore.

Storage: The storage in Ericsson reseach cloud in not fast enough for the VMs. So reading the video files and writing the MP4 files from the dasher takes a longer time than we expect. Faster data storage will be an option to read/write the file in cloud.

Improvements on the cloud  Installing storm and creating instances is really a time consuming process. Although it is possible to create instances from the snapshot from
a running instance. Still it will take much time to configure all the setting for Storm and Zookeeper services. Openstack offers Apache storm for data processing and this could remove all the complication to configure each of the instances individually.

We created a couple of large instances to get a higher number of VCPUs. Large instances requires 8 GB RAM and half of this memory goes to waste as we cannot use them. The opportunity to create a medium instance with higher number of VCPUs will make each worker much faster.
BIBLIOGRAPHY


