Scalable Distributed Video Transcoding Architecture

Tewodros Deneke

Master of Science Thesis

Supervisor: Prof. Johan Lilius
Advisor: Dr. Sébastien Lafond
Embedded Systems Laboratory
Department of Information Technologies
Åbo Akademi University

September 2011

ABSTRACT

Nowadays video is being produced and consumed in more component representation formats, more device types and over variety of networks than ever. Transcoding is a process of translating or converting one coded signal representation to another. However most of the time transcoding becomes computationally intensive process. Due to this, one might need to utilize some sort of distributed computing approach to efficiently exploit the extra computational resources available among multiple machines, multi-core CPUs, and distributed computing resources in a given facility, home or a dedicated cloud computing infrastructure. This distributed transcoding approach will then reduce rendering and start-up time of a video for on demand transcoding. Towards this goal, this thesis presents a distributed transcoding approach on how a set of heterogeneous processing and network capabilities can be utilized in an effective manner for the purpose of video transcoding.

Keywords: Cloud Computing, Distributed Computing, Transcoding, Scheduling

ABBREVIATIONS

ARM Advanced RISC Machine.

ASIC Application-Specific Integrated Circuit.

AWS Amazon Web Services.

CPU Central processing unit.

CIF Common Intermediate Format.

DCT Discrete Cosine Transform.

DSP Digital Signal Processing.

EC2 Amazon's Elastic Compute Cloud.

HD High Definition.

HPC High Performance Computing.

HTTP Hyper Text Transfer Protocol.

MIPS Millions of Instructions per Second.

MPEG Motion Picture Experts Group.

MPI Message Passing Interface.

NAS Network Attached Storage.

QOS Quality of service.

RGB Red Green Blue.

RISC Reduced Instruction Set Computer.

SDTA Scalable Distributed Transcoder Architecture.

TC Transcoding Capability.

TEKES Teknologian ja innovaatioiden kehittamiskesku.

VLSI Very Large Scale Implementation.

GLOSSARY

Bitstream Bitstream is a time series or sequence of bits which represents a stream of data. In video compression this sequence is emitted by an encoder and is dependent on the codec used for compression.

Cloud Cloud is a name to imply an entity that delivers computing resources as a service rather than a product.

Chrominance Chrominance is the information that represents colour of an image with out the brightness.

Cluster Cluster is a set of networked computers used to deal with a big computing task through collaboration or divide and conquer.

Codec Codec is a software(computer program) that is able to do the task of both encoding and decoding.

Decoder Decoder is a software, algorithm or a device that is used retrieve back the original information representation or row data from a previously encoded data.

Encoder Encoder is a software, algorithm or a device that is used to convert information from one representation to another. It is also used to get a good and compressed representation of a row data.

Granularity Granularity describes the extent a system is broken down into smaller parts.

Luminance Luminance is the information that represents the brightness in an image.

LIST OF FIGURES

CONTENTS

CHAPTER

ONE

INTRODUCTION

Video content is being produced, transported and consumed in more ways and devices than ever. Meanwhile a seamless interaction between video content producing, transporting and consuming devices is required. The difference in device, network and video representation types results in the necessary requirements for a unified mechanism for video content adoption. One such mechanism is called transcoding. Video transcoding is a process of converting one signal representation of a given video to another. Currently transcoding is being utilized for such purposes as: bit-rate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for insuring high QOS [23][26].

Transcoding is a computationally heavy process and several methods has been proposed in order to increase its efficiency[18] [28]. Among them many attempts have been made to decrease its computational complexity by reusing information like DCT coefficients and the motion vectors extracted from the original coded data instead of fully re-encoding the video content. On the other hand to realize multiple transcoding and speed up, studies has been done to integrate multiple processors to fully decode and re-encode incoming

video [21]. However there has not been much work done in the analysis of how a distributed transcoding architecture could be utilized in view of on demand transcoding where quality of users experience such as start up time and jitter free video experience matter a lot. In addition, the advances in the utilization and economics of the distributed computing has widened such possibilities as elastic computing in the cloud which require a different type of distributed transcoding architecture [14][29].

In this thesis we will try to propose a possible scalable distributed architecture a transcoder designed with the cloud in mind and evaluate its implementation performance for on-demand video transcoding.

1.1 Background

The CloudSoftware Program (2010-2013) is a TEKES financed and Tivit Oy coordinated program. Its aim is to improve the competitive position of the Finnish software intensive industry in the global market. As part of this research project, the Embedded Systems Laboratory at the Department of Information Technologies at Åbo Akademi, has been evaluating the possibility of building a more energy efficient cloud computing infrastructure using low power nodes. Along side this endeavour suitable cloud software services such as scalable distributed video transcoding over the cloud are being developed [9]. This project is also in-part sponsored by Amazon cloud [8].

1.2 Overview of the problem

As mentioned in the previous sections Transcoding is a computationally heavy task. Especially, small devices such as NAS boxes, routers and mobile phones are not capable of handling it alone for on demand streaming requirements with acceptable user experience. The following table shows the transcoding capability measures of a given set of processors that are commonly available.

Processor		Transcoding capacity	
Name	Specification	4cif - hd720	4cif - hd1080
Intel core 2 Duo E6550	2.33GHz 4GB RAM 4094Kb cache Desktop	26 fps	14 fps
Intel core 2 Duo E6570	2.10Ghz 4GB RAM 2048Kb cache Laptop	16 fps	10 fps
ARM cortex	250MHz 512MB Mobile	6 fps	2 fps

Table 1.1: Video resolution transcoding capability measures of different processors (devices)

Table 1.1 shows the transcoding capability of different processors that are commonly found in home multimedia devices. It shows transcoding speed achieved by the different processors in frame per second for a given sample video [7]. Transcoding is done to change the samples video resolution from 4CIF to HD720 and HD1080 respectively. The table clearly shows how transcoding is a computationally intensive process even for a relatively high end processor working on a single stream. A given home media server might need to process several streams which makes transcoding even more demanding.

On the other hand the high availability and scalability of distributed and cloud computing infrastructures seem to provide the perfect missing requirements such as the lack of computational power and availability at all times [14][8][1].

Interestingly, the availability of distributed computing infrastructures could even be seen in current conventional homes. For example, an average home is equipped with a network of devices with one or more CPU units in them. Some of these devises include desktop computers, game consoles, networking devices and the like. In addition the increase in the availability of private and commercial cloud infrastructures insures the possibility of on demand computational scalability with seemingly infinite computational power and low cost [14].

So far several transcoding approaches and optimizations has been proposed in order to enhance transcoding process. A distributed video transcoding approach is one of the promising ones due to the fact that it is scalable and works with the current available infrastructures making it inexpensive as well. However, the task of distributed video processing is not trivial and is by far one of the challenging tasks due to:

- *Strict stream structures.* All video standards specify a given type of bit-stream structure and its decoding process. This bit-stream structure is arranged in such a way that allows temporal and spatial redundancies are reduced through forward and backward referencing of video data along with difference vectors used in calculating complete video data rather than keeping all the original video. While this makes the compression of video possible and efficient it poses a challenge when it comes to load partitioning and balancing for a distributed video processing.

- *Load balancing.* In any distributed system load distribution is a challenging task. Without proper load balancing a distributed implementation of a transcoder might perform badly or even worse than its strictly sequential counterpart. The main challenge of load distribution from the perspective of transcoding comes directly from the video stream structure which contains variable data in terms of processing time dependent only on the type of data contained.

- *scheduling.* Scheduling of video stream processing on a heterogeneous computing platform with certain time constraints is a rather non trivial task due to the architectural and capacity differences in devices involved and a possible dynamic computational load and capacity changes over time.

- *non-existence of supporting platforms.* Distributed video stream processing is a relatively new and active research area. as such their are not many well developed generic platforms that abstract at least part of the necessary components in building such systems. Classical distributed systems are only used to dealing with stored data rather than a stream.

For a proper solution towards a distributed video transcoder all the above challenges should be properly investigated and tackled.

1.3 Objective of this thesis

The main objective of this thesis is to investigate existing distributed computing infrastructures and how they are being utilized from the point of view of the video stream processing and propose a way or an architecture on how to utilize such infrastructures in an efficient manner. The investigation will begin with studying and finding the right system components that are required. In addition to this a minimal version of such system needs to be implemented on a selected platform for the purpose of further experimentation and performance analysis of the components and the proposed system as a whole.

1.4 Thesis structure

To effectively communicate the work done and contribution of this thesis in the area of transcoding, this thesis document is structured as follows.

- *Chapter 1* briefly explains the problems that are the basis of this thesis work and also explains the challenges that need to be addressed. In addition a short introduction of the research area is given in a more abstract way so that all kinds of readers are introduced to the work done and its relevance.
- *Chapter 2* and *Chapter 3* explains the basic principles related to the thesis related fields. Theories and principles that currently exist are explained. This are then used as as basic building blocks of the thesis work.
- *Chapter 4* presents the details of the methodology used to develop the distributed transcoder. Details of the actual implementation and platforms used are discussed. In addition the general architecture of the

system is explained along with the detailed component description that make up the entire system.

- *Chapter 5* presents tests and experimental results related to the work done.
- *Chapter 6* concludes the document through mentioning the most important findings of the work and a recommendation on further future work.

CHAPTER TWO

VIDEO COMPRESSION AND TRANSCODING

In this chapter we shall start with a brief overview of video compression. This is important because understanding the fundamental concepts associated with it is necessary when it comes to proper partitioning and scheduling of a video stream on a distributed platform for transcoding. It is also useful for understanding the underlying concepts that make up a video transcoder [12][25][13]. Following that, a summary of video transcoding and its associated theories will be presented in hope of giving an overview of what already exists.

2.1 Video Compression

Row video contains a large amount of data. On the other hand communication and storage capabilities are limited and thus expensive. For example a given HD video signal might have $720 * 1280 pixels/frame$, and a playback speed of $40 frames/sec$. This produces an information flow of:

$$\frac{720 * 1280 pixels}{frame} \times \frac{40 frames}{sec} \times \frac{3 colors}{pixel} \times \frac{8 bits}{color} = 884.74 Mb/s$$

and for a channel with a bandwidth $50Mb/sec$ we require the video to be compressed by a factor of about 18. The way this is achieved is through video compression. Video compression is done through reduction of redundancy and irrelevancy [12][15][13].

- *Redundancy reduction.* Consecutive frames in a video signal are highly correlated and exhibit temporal redundancy. They typically contain the same content with the only difference in placement (movement) of objects that makes up the content. Redundancy also exists inside a single frame due to the fact that neighbouring pixels exhibit some sort of correlation and this type of redundancy is called spatial redundancy. Video compression utilizes this two sources of redundancies.
- *Irrelevancy reduction.* All the information in a video signal is not perceptually important. The human brain perceives the world in a certain way and this fact can be utilized to reduce some irrelevant information that is unimportant [12][15][13][10].

2.1.1 Generic Compression System

In all its generality compression can be abstracted to a simple set of operations as can be seen in the following figure.

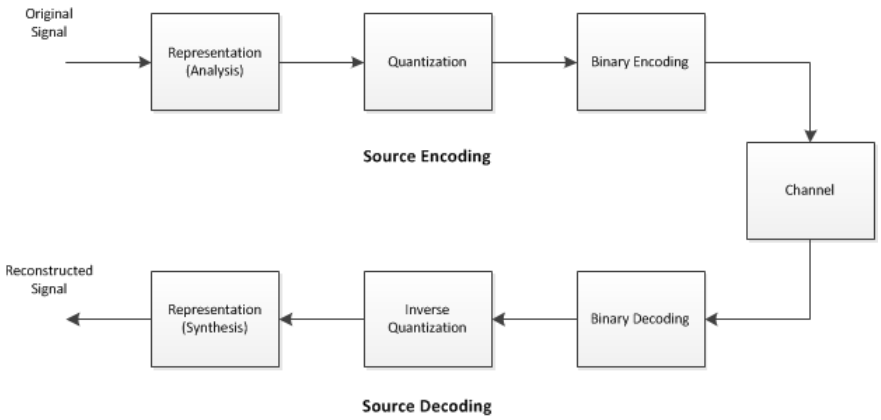

Figure 2.1: A Generic compression system

Figure 2.1 illustrates the general idea of how compression is made in a given data. As one can easily deduce first the original signal is taken in and is digitally represented in a such a way that the most important information is made to be concentrated. Then only the most important part of this representation is quantized, encoded and transmitted or stored.

Image Compression

Frames (images) are building blocks of a video. Video is simply a playback of a set of consecutive images at a certain rate (mostly at about 25 images/sec or more). During compression, the spatial redundancy that exists among neighbouring pixels is utilized through different techniques. This techniques include such concepts as colour conversion and transformation of information representation. In an image compression color pace conversion is made from RGB to luminance/chrominance. This fact is useful to utilize the weakness of the human perception and discard some irrelevant information. Then the image is partitioned in to smaller blocks. In smaller blocks, pixels are likely to correlate more which allows room for some compression. Finally, DCT of each small block is computed. This DCT transformation represents the original signal with basic components in such a way that these components are ordered in terms of there relevance to the whole signal. This allows one to only take the first limited number of components to be quantized and binary coded in to a compressed bit stream [13][10].

Video Compression

Video compression is an extension of image compression which also utilizes the temporal redundancy that exists among consecutive frames in addition to image level compression. Instead of only encoding individual frames independently as discussed in the previous section one can first predict a frame based on the previous frame and code only the difference in this prediction. Therefore to minimize the error and be able to get a better compression requires an efficient way of prediction.

In current video encoding systems predictions are made through compensating for relative motion between two video frames. This process is known as *motion-compensated prediction* and is practised through dividing the current frame into blocks, finding the best matching block from the previous (and/or future) frame and calculating an associated *motion vector*. This fact then enables us to predict current frames using previous frames and a set of associated motion vectors and avoids the storage and transmission of all the bits from the original video. How ever such strong dependency also makes partitioning and scheduling of distributed video processing more difficult. The following figure shows the prediction dependencies that exist in a sequence of video frames [25][13].

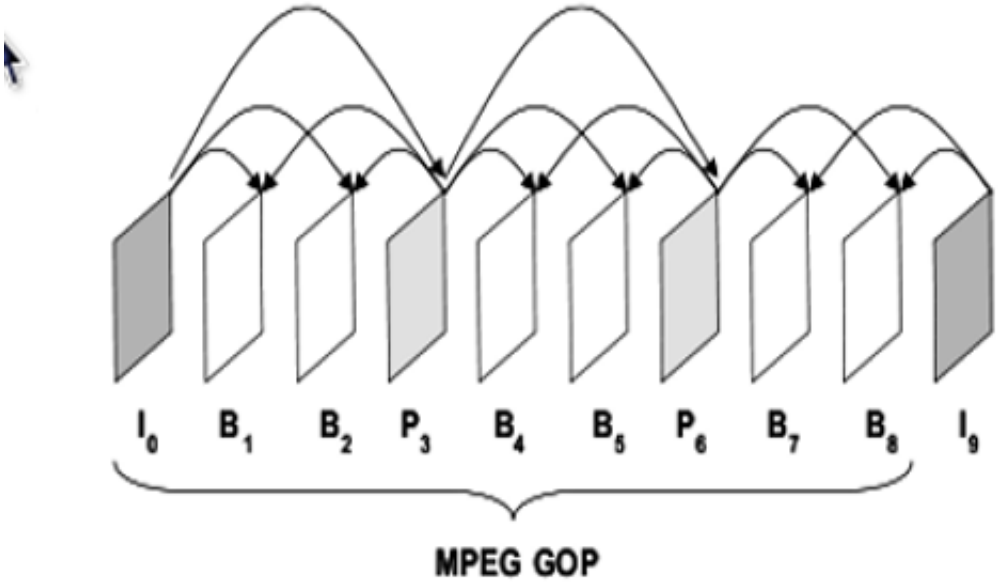

Figure 2.2: Example of prediction dependencies among sequence of video frames [13].

Figure 2.2 illustrates the dependencies created among sequence of video frames that enables video compression. As can be noted from the figure MPEG has three types of frames namely I, B and P. *I frames*, or intra-coded frames are frames which are coded independently of any other frames. These frames can be decoded by themselves with out requiring any additional information from neighbouring frames. They are used to give a random accesses (seek point) to the video stream and as a key frame from which other types of frames can start to be calculated and predicted. *P frames*, or predictively coded are frames which are coded based on previously coded frame. these means a reference to a previously coded frame is required to decode these frames *B frames*, or bi-directionally predicted frames which are coded using both previous and future coded frames [25][13].

In the following section we shall see how this video sequence structure affects the process of distributed video transcoding.

2.1.2 Video bit-stream structure

Different standards have been set to specify the bit-stream structure of compressed video content in order to make sure the proper interoperability of video producing and consuming devices. This bit stream is what is transmitted and stored among different devices. The specific structure of this bit-stream is different for different compression techniques but has a general structure that follows the fundamental concepts that make compression possible. The following figure shows the MPEG bit-stream structure in an abstract way.

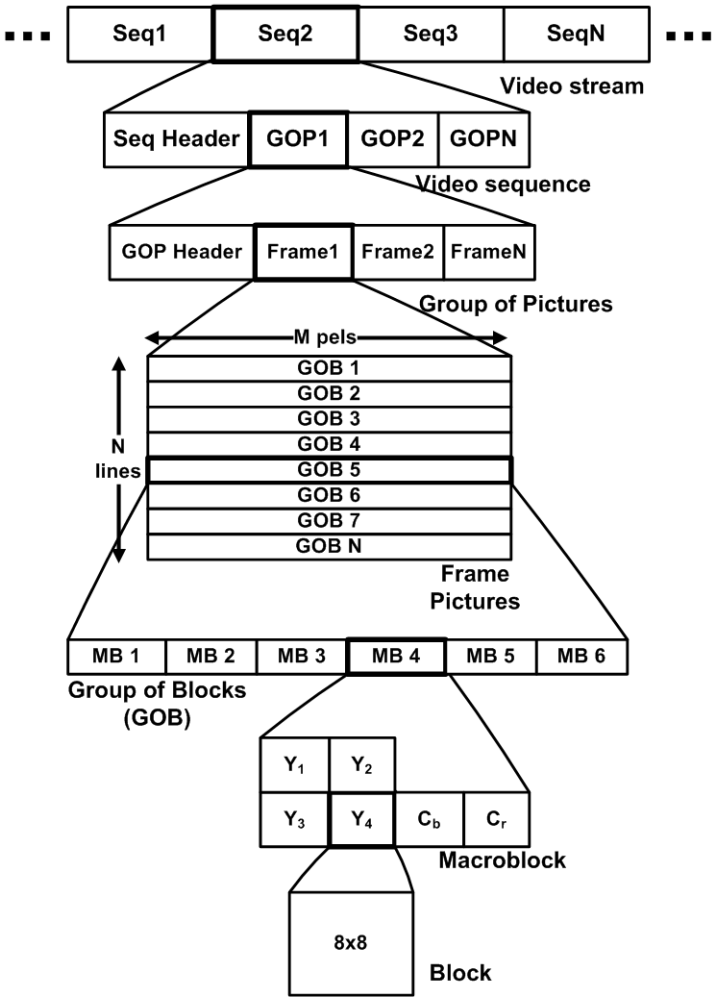

Figure 2.3: MPEG codecs video in a hierarchy of layers

Figure 2.3 shows the bit-stream structure that result from video compression algorithms. From the perspective of distributed transcoding we are more interested in this structure for the main reason that we require a suitable way to split the bit-stream among different processing units. Choosing the right layer for this propose is essential as different granularity results in different performance in terms of system latency, total transcoding time and implementation complexity.

Utilization of layers with smaller granularity such as macro-block results in higher communication and synchronization costs as well as higher implementation complexity. This is due to tighter data dependency down the hierarchy as seen in figure 2.3. Granularity such as video sequence (see figure 2.3) will result in higher latency of the system affecting the streaming behaviour of the video being processed. This is because it takes more time to process a video sequence than a GOP or a frame.

The distributed transcoder implementation in this work utilizes the group of pictures as atomic units of distribution. However the main reasons for this choice are mainly implementation complexity and the side effect of fine granularity in terms of communication over heads [26] [23].

2.2 Video Transcoding

On a more abstract definition video transcoder is a piece of software that decodes a given signal representation of a video and re-encodes it to another. The following figure shows the general task of the transcoder.

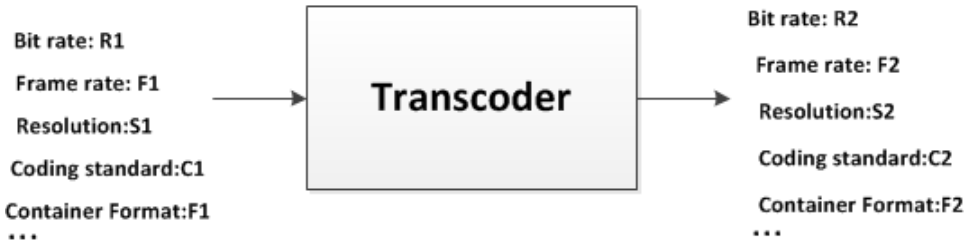

Figure 2.4: A Generic compression system

Figure 2.4 illustrates the main idea behind transcoding and the common video transcoding operations. The task of a video transcoder could range from a simple task of converting the container (i.e. the header that contains meta-data about the streams that are found in a video file) format of the video to re encoding the entire stream with a different codec (standard). In the following section we shall consider the transcoding that requires encoding to a different codec standard.

A straightforward realization of a transcoders to cascade a decoder and an encode such that the decoder decodes the compressed input video and the encoder re encodes the decoded video into the target format. However the whole process of decoding and re-encoding are not entirely necessary. In fact complete re-encoding is disadvantageous due to the fact that media encoding needs extensive computation. In addition consecutive encoding always results in the loss of more quality. Therefore some of the steps that make up decoding and re-encoding are to be left out through reuse of encoding decisions that are already made during the last encoding. This approach is what is called transcoding and is highly optimized and efficient than re-compression. The following figure shows a cascaded implementation of a video transcoder end its simplified architecture that utilizes the reuse of information contained in the original incoming bit-stream.

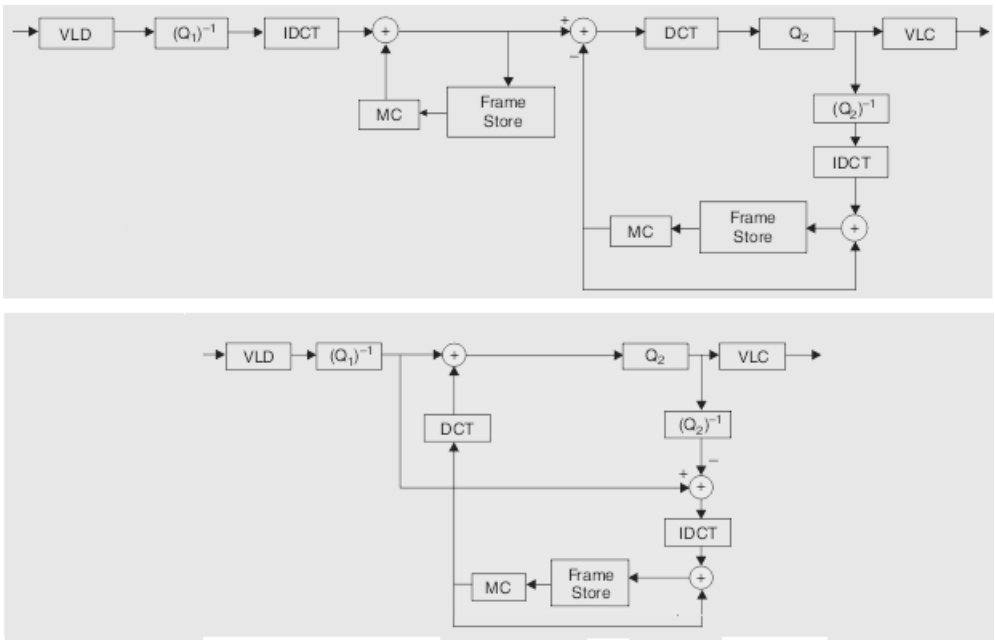

Figure 2.5: A straight forward and simplified implementation of a bit-rate transcoder [23]

Figure 2.5 illustrates some of the optimizations that already exist in a given transcoder. Reuse of information that is found in the originally coded stream such as Motion vector differentiates transcoding from a straight forward re-compression.

Video transcoding is currently an active area of research. A lot of such optimizations are being done on the optimization of such systems as described in the Figure2.5. Other approaches that are being used include such ideas as ASIC chip design, rate adoption transcoding and distributed cluster based transcoding. Each of these approaches has there own advantages and disadvantages [2][21]. For example a specifically designed ASIC chip is fast but is non flexible as it is not possible to add new functionality and component representations (codecs and formats) once the system is manufactured. On the other hand rate adoption transcoding has the disadvantage of degraded video quality when the bandwidth and the processing power in a given transcoding environment becomes low but has a rather easier implementation than a distributed transcoder which has a potential for delivering high quality service using existing infrastructure with the expense of higher system complexity.

CHAPTER

THREE

DISTRIBUTED VIDEO TRANSCODING

Distributed computing offers scalable load management and higher availability. At the same time transcoding requires scalability and availability which makes the two an ideal combination into solving the problem of on demand video content delivery through different channels and to different devices.

The aim of using distributed computing resources for the purpose of transcoding comes from the fact that new physical or virtual machines can be attached to a system in order to handle the extra computational requirement needed to fulfil video availability constraints. Figure 3.1 shows the idea behind the need for distributed transcoding. Notice how data parallelism improves the availability and throughput of the system.

Figure 3.1 illustrates the general idea behind the utilization of several compute nodes for improving the total throughput of the system. As long as the bandwidth is available and the computation to communication ratio is high such stream processing approach will result in some sort of performance gain depending on the efficiency of the scheduling mechanism.

Distributed computing usually requires universal access to high-grade com-

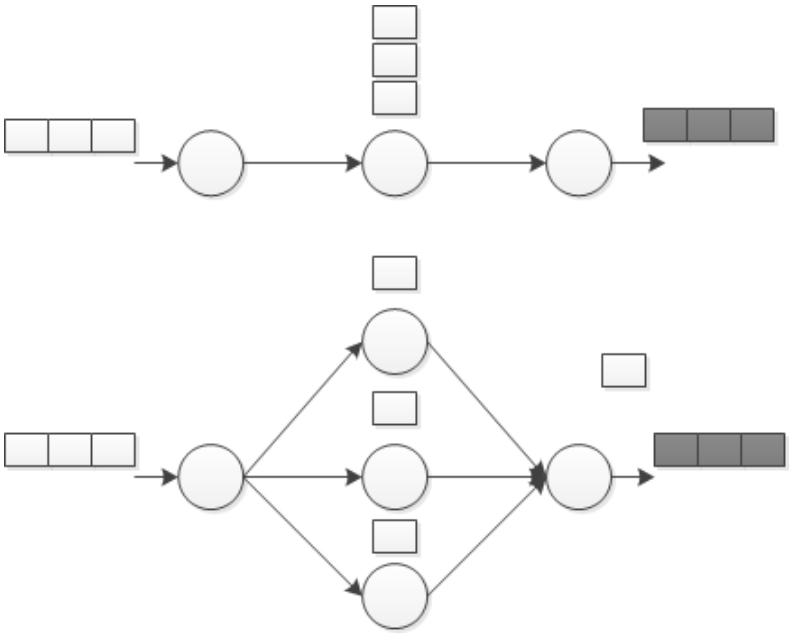

Figure 3.1: Distributed stream processing

putation facilities. Currently this is becoming a reality with the advent of commercial cloud infrastructures such as Amazon cloud (web services). At the same time average users still suffer from lack of processing power for demanding applications. Often, computation complexity and bandwidth necessity make video encoding or transcoding a difficult task. Thus, parallel and distributed architectures for video encoding have been the subject of research for already some years. Most successful attempts have unfortunately remained in the domain of those with high performance computers connected by high-speed networks. Architecture to distribute and encode or transcode video on the Internet (cloud) where the system is heterogeneous would benefit users. Though such a system would not be strictly real-time for some time to come, it would still enable users to encode and share high quality videos with resources that are available everywhere.

As opposed to the classic distributed systems, in distributed transcoding systems it is required that the system should be able to deal with a stream of unbounded data and in a certain time constraints. Data stream processing is a way to support real-time processing of continuous data streams. Typical scenarios include multimedia processing and sensor data analysis from

large sensor networks. These applications need low latencies for real-time responses. Usually stream data rates vary in time. How to insure some real-time stream processing is one of the key technical challenges in the area of distributed stream processing systems. One of the simple solutions is to add new physical nodes to guarantee that the overall system performance is adequate to handle the largest possible burst of data. However, various problems often prevent the use of this solution, for example even if we can solve the problems and provide sufficient computational resources to handle the highest possible data rate, most of those resources may be wasted if the normal data rate is/becomes low. Fortunately the emergence of cloud computing as an extension of distributed computing gives us the chance for an utility type of computing and essentially solves both problems of over provisioning and under provisioning [29].

Cloud computing is a new paradigm in distributed computing in which computing is offered as a utility by third parties whereby the user is billed only as per their consumption in a service oriented manner. In the next section we shall see what the cloud means and how it has a role in distributed computing and particularly to video transcoding.

3.1 Cloud Computing

Cloud computing is a way to use dynamically scalable and virtual resources as a service through an on demand request over the internet. The main characteristics of the cloud include high scalability and high availability. Scalability means that the cloud infrastructure can be expanded to very large scale with several resources. Availability means that the services are available even when quite a number of nodes fail or at any particular time. In addition to such features the cloud brings a new business model to the IT industry where users pay for only the time they require the service (resource). This means that small start-up or private users can gain access to a high quality IT infrastructure with a small cost. The following figure summarizes the idea behind cloud computing.

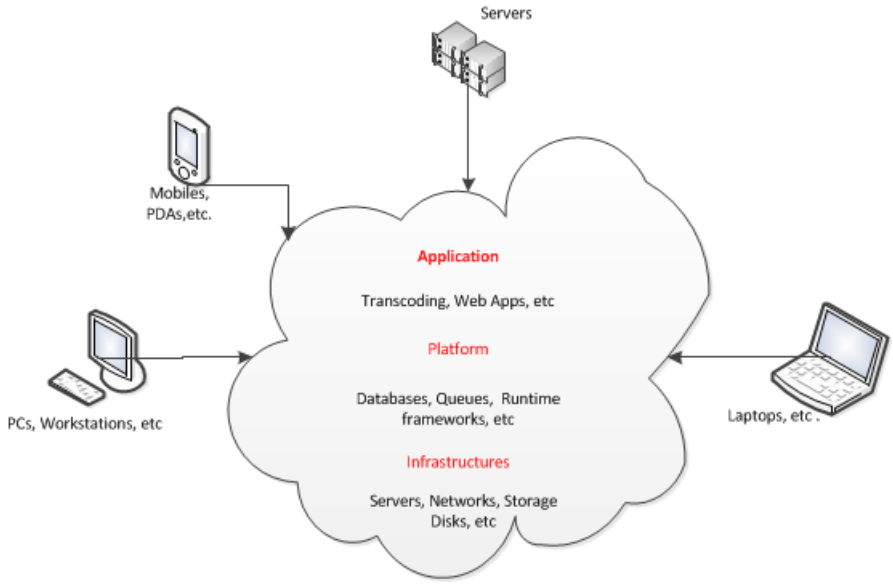

Figure 3.2: Over view of cloud computing

Figure 3.2 shows the basic components that make up the cloud. These components often called layers which include the application, the platform, and infrastructure. The cloud could therefore be defined in short as an abstract entity that delivers different applications, platforms and infrastructures as a service.

To our distributed transcoding system, the main components of the cloud are virtual machines which can be spawned on a given computer hardware dynamically. The use of virtual machines instead of the physical machines enables efficient utilization of resources in an automatic manner.

The growth of the cloud computing paradigm requires the IT industry to have a different architecture for the applications that run on it. In this section we try to illustrate a general architecture of a distributed transcoder that is made in consideration with the existence of the cloud. The following figure shows this architecture.

Figure 3.3 illustrates an architecture of a distributed transcoding application which utilizes distributed resources that are found locally together with virtual and scalable resources from the cloud. Note that in the figure, the diamonds represent compute nodes and the circles represent the tasks. The main

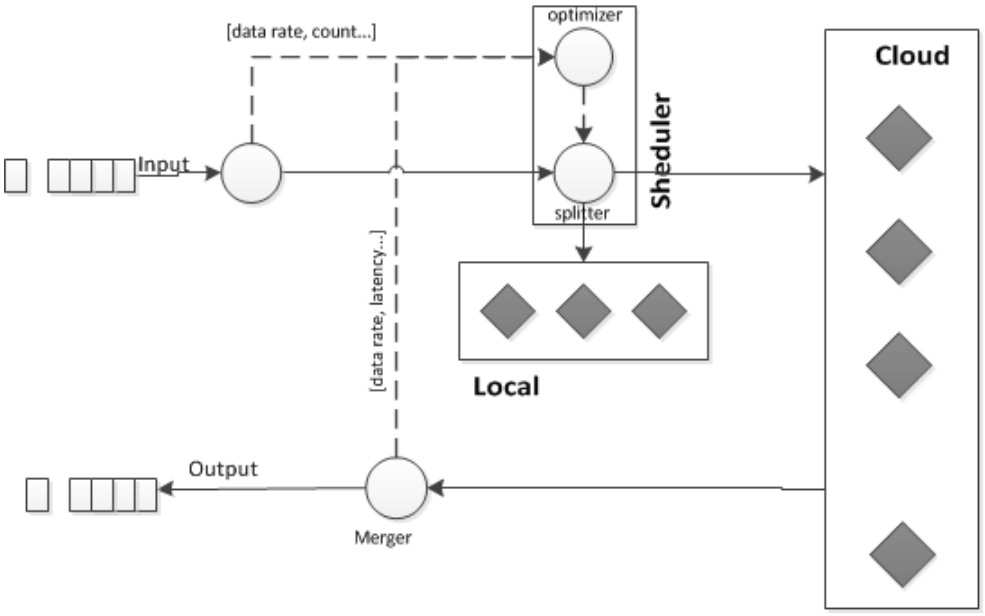

Figure 3.3: A possible architecture for a distributed transcoder

components that make up such a system are listed and discussed as follows.

- *Splitter* is a component that splits the stream in such a way that each processing unit in the system is well balanced. A good splitter should take additional input metric besides the input stream. In such a case important data from the output stream and the input stream could be collected by an optimizer component and analysed in to information for decision making by the splitter. Among the decisions that the splitter needs to take include, size of work to be sent to each node in the system and the amount of machine instances to add or reduce.

- *Optimizer* is a component that collects data from the stream (both input and output) and tries to optimize some parameter such as economic cost of the system, latency, start-up time, etc. A given optimizer could use several approaches in order to calculate and decide on the required resources in the future according to optimization parameter. In video transcoding it is important to note that video streams have very dynamic characteristics with respect to processing time of different parts of the stream. Transcoding process is also dependent on the type of transcoding task we are dealing with requiring different functional blocks with

different computational demand to be executed. For example, file format transcoding requires much fewer non expensive function calls in comparison to a codec transcoding (see Appendix A). Approaches that could be used for these include selecting a set of features and learning patterns in such streams [19][27][17]. Correct prediction of these processing times is needed for qualified QoS [20][24] and scalable transcoding system.

Scalability is a term used to express a systems ability to handle a growing amount of work through graceful addition of resources. This particular property is important to our distributed transcoder in order to maintain a required set of constraints paused by a user or total load requirements of the system. For example, some video transcoding requests might need more computational power than others or a user might want to start playback of a video on a relatively short time. With a scalable system such user constraints can be met through utilization of the right amount of resources for individual transcoding requests. A scalable transcoder also enables to meet different load conditions.

The concept of scalability is related to the need of matching capacity of the transcoding system with the total load (such as transcoding requests). With a scalable transcoder, it possible to handle more transcoding requests with addition of virtual or physical machines from a distributed pool of resources. A scalable system need also be able to deal with under utilization. This is important in many ways such as energy consumption of media server.

When dealing with video content delivery some kind of quality of service has to be maintained by the system. These include such things as video start up time, delay and jitter free playback. Users of video content usually has higher expectation and expect a relatively higher quality of service and are intolerant to such things as excessive frame drop-out, start up time and jitter which should be avoided as much as possible. Here such things as admission control and elastic computing play a major role [20][24][16][29]. Admission control is an important concept in real time systems where a system only admits those jobs that has some guarantee to complete in time [16] and elastic computing is an approach

of utilization of resources as a utility.

- *Merger* Is a component that merges processed (transcoded) streams from several compute nodes in a system.

The following sections will further discuss some of the main tasks that are required by the system components listed above.

3.2 Video Stream Segmentation

Distributed video transcoding is a data parallel application where several instances of a program do similar task only on a given portion of the data. For this reason a certain way is required to split the video data into several parts. As we have seen from section 2.1.2 a video stream is defined in several layers. At each of these layers there is a potential for partitioning video content. However partitioning at lower layers such as macro block results in heavy synchronization issues as dependencies on such layers is heavy. Similarly spiting at higher layers like sequence layer has the side effect of increased latency in the system and affects the streaming behaviour inherent to video content streaming.

Our approach for video splitting utilizes GOP as an atomic unit of distribution. As can be seen from figure 2.2 a GOP is a structure that contains one key frame (I frame) and a set of predicted and bi-directionally predicted frames. Video segmentation at GOP is easy as there is no or minimal dependency between consecutive GOPs in a given video stream. This fact makes GOPs suitable to distributed and transcoded video on different compute nodes with less synchronization overhead and software complexity. However, care must be taken to avoid the use of large GOPs which could contain several frames (i.e. more than 25 frames, an optimal number for human perception). The main reason for this is that slower compute nodes could take long time before processing large GOPs and at the same time it is not possible to distribute this load among several nodes using our approach.

In general video segmentation is needed in order to balance the load of stream processing among several compute nodes with different capabilities. The following section will discuss more on this issue.

3.3 Load Balancing and Video Segment Scheduling

One of the most important concepts in distributed computing is load balancing, where compute nodes in the system are given relatively equal task per unit time. However different compute nodes that are found in a distributed environment are equipped with different computational capability and have different communication bandwidth. In addition video content is expressed in terms a sequence of constraints. This constraints has to be maintained besides the distribution of equally balanced tasks.

In this thesis two load balancing approaches will be looked at and used, one static and another dynamic.

3.3.1 Static Load Balancing

In case of static load balancing, the distribution mechanism has to have a prior knowledge of each compute node capabilities and segment the video in such away that each node gets to finish its job as the same time as the rest and be prepared to receive its next task in-line. The following figure shows the the basic idea of how a video stream could be segmented and distributed among different compute nodes.

Figure 3.4 illustrates one way of video stream segmentation and load distribution. The input video stream is read and segmented in to a set of parallel streams that are to be processed by different compute nodes in a given distributed video transcoding system. The main idea behind such a distribution approach is, different compute nodes get different number of GOPs to process according to their capability. Note that the capabilities of each compute node need to be known prior to the stream segmentation possibly

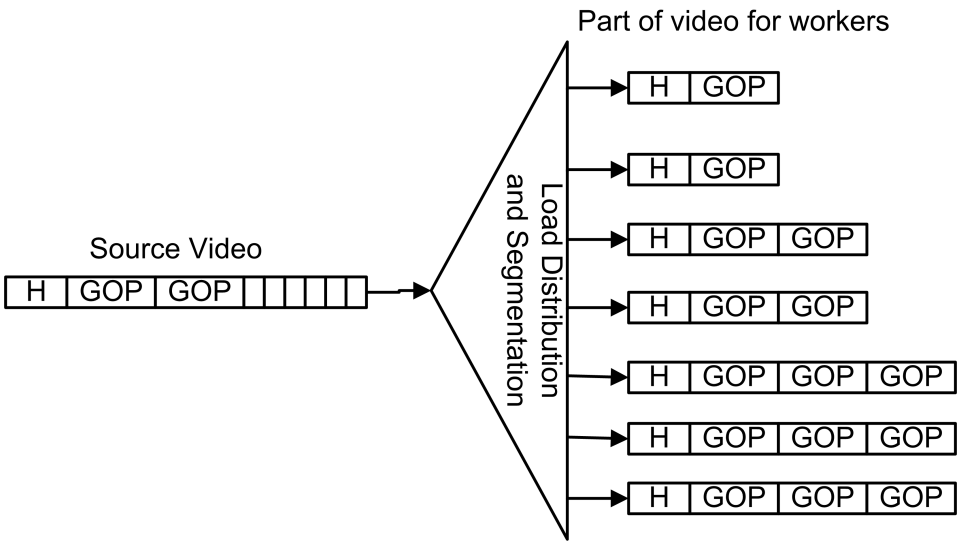

Figure 3.4: Stream segmentation and load balancing

through profiling or at runtime through pooling. This approach is easy to implement as we only need to have a table that holds the estimated capability of nodes. How ever this is not often efficient as resources could be shared with other tasks other than our application.

3.3.2 Dynamic Load Balancing

In this case we control the load distribution through runtime monitoring. This could be done using different methods however, we shall look at three such approaches as follows.

- **Acknowledgement based approaches**

  A simpler implementation of dynamic load distribution would be where, all the nodes are given equal size yet relatively small task and the ones that are done will be given the next job in line. In this approach the the granularity of the data to be processed at a time is made to be fixed and we do not need to predict the capability of the nodes in advance however we need to keep track of the progress of each node possibly via acknowledge signals. An algorithm for such approach is listed in section 4.2.3.

- **Control theory based approaches**

  Such concepts as adaptive prediction (adaptive filters) could be used for predicting decoding times before hand [27][17]. The summary of such methods could be stated as follows.

  - Generic metrics is chosen (such as encoding parameters) and given as an input to an adaptive predictor (filter).
  - At each input metric the filter produces a *prediction*.
  - Current input is then, compared with past predictions to get a correction (error) value .
  - Final prediction will then be sum of correction and prediction is the output of the algorithm.

  Following this approach requires a clever choice of input metrics by an expert and example algorithms can be found int [27][17].

- **Machine Learning based approaches**

  Dynamic load balancing could also use more complicated machine learning approaches to decide at run time what the optimal load distribution should be. In this approach a set of features (metrics) that are crucial in predicting processing times of a given video stream are selected. Feature selection could be done through analysis and profiling of sample transcoder and constructing a generic model to select features for. Once a set of features are found, model parameters could be calculated through a learning phase on a specified platform. The model and the calculated parameters are then used to predict processing times for future data [20][19].

  This approach would give better performance if we are able to construct a good, simple and generic model. However as a transcoder is a collection of codes and performs in a very dynamic manner (see Appendix A) it would be a challenging task and with the addition of new codecs in the future construction of such models should be done carefully.

3.3.3 Segment scheduling

According to the type (method) used for stream segmentation and load balancing the scheduling of tasks differ. For example for the static load balancing approach explained in the previous section a simple Round ribbon type of schedule would suffice. This is because the capability of the nodes is known in advance and the segmentation are made to take equivalent processing time and nodes are expected to finish their assigned task equally. On the other hand often dynamic load distribution approaches proposed in the load balancing section requires a scheduler that keeps track of the compute nodes and assigns the task to the fastest currently free nodes. We shall later come to the implementation details of these two approaches.

CHAPTER

FOUR

DISTRIBUTED VIDEO TRANSCODER IMPLEMENTATION

Implementation of the distributed transcoder utilizes the master-slave approach. One compute node among several compute nodes is used as a master node and the rest of the nodes are used as worker nodes. In a real life case this master node could be a media server which gets several requests on stored media. The following picture shows the general logical architecture of the distributed transcoder.

Figure 4.1 illustrates the abstract logical architecture of the distributed transcoder. The master node acts as the *transcoding manager* with the responsibility of segmenting the video stream, load balancing, scheduling and parsing individually transcoded parts into the desired output stream. On the other hand the individual worker nodes act as a *transcoding capability* with the only responsibility of receiving and transcoding their assigned GOPs in to the desired output video representation (format and codec). Both the *transcoding manager* and *Transcoding capability* implement different components that enable their functionality and will be explained in the coming sections.

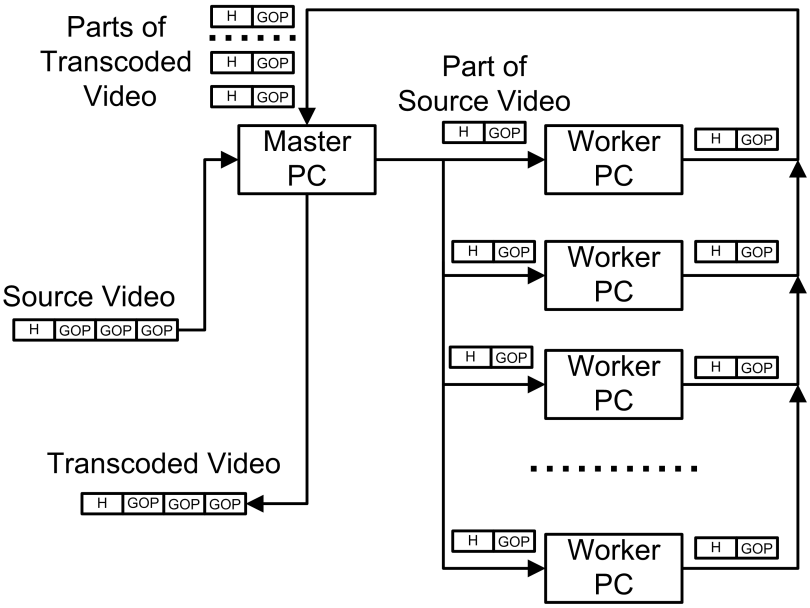

Figure 4.1: Logical architecture of the distributed transcoder

In principle the worker nodes could be located anywhere nearby or further away in to a private cloud or a commercial cloud. The following activity diagram shows the particular responsibilities of a worker and master node.

Figure 4.2 shows the roles played by a master process and the worker processes. The main task of the master node is to control the overall activity of the system while workers only run the any one or multiple processes could be scheduled to run on a given virtual or physical machine. This process mapping among different compute nodes (physical machines) is done through an MPI runtime environment and will be discussed more in the coming section.

In our particular case the master node also called *transcoding manager* is part of a media server holding the required content to be transcoded and the worker nodes are low power private cloud nodes with the intention of showing the possibility of utilizing low power and easily available compute nodes for collaborative transcoding. The whole system is also deployed and tested in the Amazon commercial cloud. This experiment as shall be seen in the coming sections is intended to test the scalability of the system [22][6].

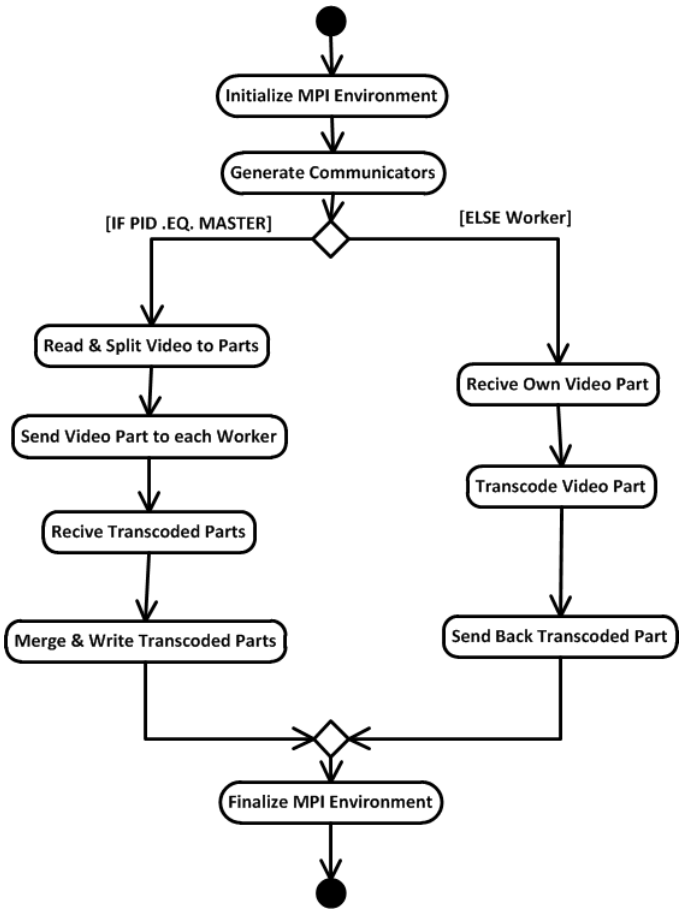

Figure 4.2: Activity diagram of the distributed transcoder

4.1 MPI Framework

In order to build and test the architecture of the distributed transcoder in a specific amount of time we have utilized the MPI framework. MPI or message passing interface is a standard that specifies implementation of a message passing model of parallel computing. Message passing is a computing model which is mostly used in a network of compute nodes with distributed memory. This model of computation mainly consists of a number of processes working on a subdivided task while communicating and synchronizing through sending and receiving messages in order to attain a common goal or solve a large problem. The simplicity of this model is what makes it useful and important. The following figure shows the basics behind

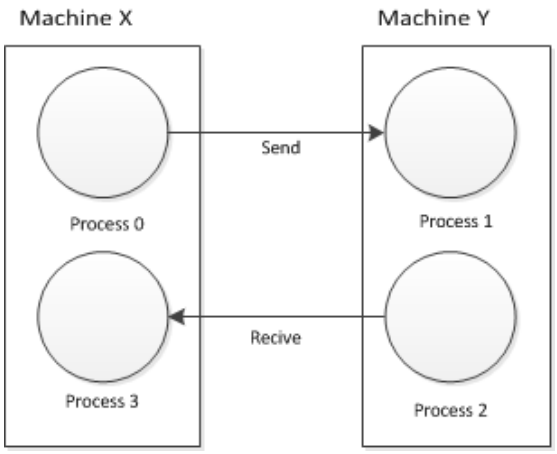

Figure 4.3: Basics of message passing

message passing model of computation [3][5].

Figure 4.3 illustrates the basic idea associated with the message passing model of computation. As can be seen the main components of the model include characteristics:

- A set of tasks mostly also called processes that use their own memory. These tasks (processes) can be located in the same machine or across a network on an arbitrary number of machines. In the case of our distributed transcoder, there are only two types of processes. One is the root process that plays the role of the master and a set of others that act as transcoder processes.
- Computation is performed by each process while collaborating and communicating with other processes through sending and receiving messages. This messages usually contain results of some operation by other processes. In the case of the distributed transcoder the messages contain the video data that need to be processed and other synchronization messages used to track the order of the stream.

4.1.1 Important Features of MPI

Some of the features that are inherent to MPI standard that made it useful for our purpose of implementing the distributed transcoder include:-

- *Portability:* MPI is made to be portable across a wide range of computer architectures and OS platforms. This is achieved through several MPI libraries written for different architectures. This libraries are written in such a way that abstracts the difference in underlying platforms. This fact is important for our distributed transcoder in terms of enabling the utilization of any available platform in a distributed pool of resources.

- *Interleaving:* As a message passing model of computation, MPI is designed to allow overlap of communication and computation. This fact allows programs to take advantage of intelligent communication agents, and to avoid communication latencies. This for example is achieved through the use of non-blocking communication calls which run along side with some computation.

- *Efficiency:* Since the MPI standard [3] only specifies the the logical operation and functionality of its system components (such as send and receive functions), it allowed efficient and optimized implementation of its runtime and libraries in the different platforms. This fact is important for our distributed transcoder in terms of performance.

- *Scalability:* A good example that shows the scalability features of MPI could be collective communication. In MPI the concept collective communication allows processes that are involved in an application to broadcast messages to all or a group of processes rather than one to one communication. Such features enable MPI to be scalable.

- *QoS:* MPI guarantees correct transmission of messages. This means user does not need to check if a message is received correctly or not and cope with such failures. Such features let application programmers to concentrate on the main task rather than unnecessary details.

- *Made for HPC than Batch processing:* In high performance computing the priority is also given to latency of an application than in batch processing models. This is also true because MPI allows explicit expression of the parallelism by the programmer. Distributed computing models such as map-reduce are made to perform batch processing applications on large data and are more concerned on reducing over all performance in terms of completing a large task. This makes such systems hard to program for constraints such as latency and response time with out utilizing overly excessive resources.

- *Faster development and testing:* Message passing has been around for some time and has evolved very much. This fact enables it to have a lot of well developed implementations and documentation that makes the development process much easier than a new technology.

- *Future additions:* In the future the MPI implementations are stated to have a dynamic way of process management. This means we are able to attach a new process to an existing pool as we get more and more physical processors . This fact will enable one to be able to utilize the current elastic model of distributed computing such as the cloud.

These previously listed facts has made us to choose MPI for testing and analysing the distributed transcoding architecture in-time and with relatively acceptable effort while maintaining the general concepts and functionality. In the following sub sections we will present the most important aspects of the Message passing model, Communication and computation.

4.1.2 Communication in MPI

As shown in figure 4.3 communication is the one of the most important part of the message passing model of computation. In MPI the communication is made interleaved with computation which also constitutes the other main part of the model.

According to the MPI standard documentation [3][5], the communication techniques also called "modes" in MPI are of four main types which include Standard, Buffered mode, Synchronous mode and ready mode.

- *Standard mode:* In this mode MPI decides weather messages are buffered or not. This means small messages will be held in buffer before they are sent but large messages are sent when the receiver is ready and will block. Therefore, this mode affects the interleaving property between communication and computation for a large data. This mode will not be used in our implementation for this reason.

- *Buffered mode:* In this mode of communication a buffer supplied by the application during communication. The use of such buffer makes this communication mode to have a local completion semantics such that the sender process does not need to wait for its message to be received in order to proceed with its job ahead. However the application has the responsibility of assigning the proper amount of memory to ensure an error free communication.

- *Synchronous Mode:* As opposed to the previous mode this mode of communication has non-local completion semantics. In this mode a send operation completes only after a corresponding receive has started to receive the message. Therefore synchronization is the main concern here and communications in this mode has a blocking nature.

- *Ready Mode:* In this mode the application must make sure that there is a receiver ready at the time of sending data. This means the sender process trusts that the receiver is waiting and it assumes the receiver has already called the receiving procedure. If no receiver is found for a message error occurs.

- *Non Blocking communication:* Besides the previous four main communication modes MPI also offers some additional safe communication modes. Among these non blocking communication is important for our implementation. In this communication mode the communication routines return immediately regardless of weather the message has been

transfer or not. This fact makes it possible to overlap communication with computation. Here a sending process informs the system that it wants to send a message to a receiver process. Then the send routine sends the message and continues with the next statement. However it needs to confirm with the receiver via a wait statement when it needs to send the next message. This is so because the message buffer would not be overwritten by the new one.

The idea behind different modes of communication in MPI is to give more options to the programmer in writing an application. Besides communication, computation is another important thing in MPI and the following section deals with that.

4.1.3 Computation in MPI

In MPI computation is encapsulated in terms of processes. Processes are given a particular id and are assigned piece of code to execute. The code could be shared among several processes according to the architecture of the program. The number of possessors is supplied during the execution of the program as an argument to the MPI execution environment and is fixed afterwards. However new and emerging implementation have included dynamic process management systems which enable attachment of process at run time. This fact goes well with the elastic resource provisioning in the cloud and could as well be utilized in the future.

The MPI Process management is responsible for scheduling and mapping processes into available processors. This process mapping can as well be controlled from an external file that specify the mapping of processes on to physical machines. The file can also specify how many processes are created on each physical machine. In our implementation this fact will be used to insure that the machine that executes the transcoding manager process (task) also executes the transcoding tasks.

In the following sections we will look at the specific implementation details of the system components.

4.2 Transcoding Manager

The task of the Transcoding Manager is to monitor the whole process in the system. This includes splitting the stream, scheduling tasks on multiple CPU units and parsing the processed stream from the individual compute nodes. In our MPI implementation the transcoding manager acts as a master in the proposed master-slave architecture. The following sections describe the most important program components the make up the transcoding manager. Namely these are the stream splitter and the stream parser. These can also be seen from figure 4.2.

4.2.1 Stream Segmenter

As has been discussed in 2.1.2 the Bitstream structure of a video content is organized in a such a way that the spatial and temporal redundancies could be exploited during video compression and coding. Therefore the task of splitting a given stream is not trivial both in terms of finding the right place in the stream for proper segmentation and proper distribution of load among compute nodes. The splitter component reads the stream and splits the stream into a sequence of GOPs which are scheduled among different transcoding nodes to optimize a given constraint such as the total transcoding time or latency. The following figure shows the the process of video segmentation in more detail. Figure 4.4 shows the process of video

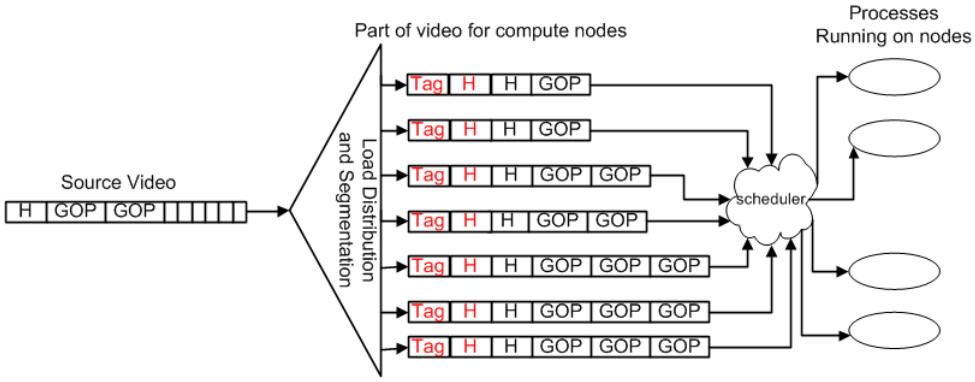

Figure 4.4: Video segmentation

segmentation and distribution among several transcoding process that reside in different physical machines. Two headers are used associated with each segment. The second header (marked in red) contains the information needed to deal with the video bit stream and is a copy of the original video input stream header. The second header is the used by the MPI frame work and contains such things as video part size and time stamps as in the following structure.

```
typedef struct buf{
...
char *splitbuf;
int splitsize;
double sendtime;
double recvtime;
...
}spbuf;
```

Finally a message tag is appended to each split and is used By the MPI runtime as an identifier for each video segment.

Once the video segmentation is done the scheduler takes the task of assigning each segment to a waiting transcoder process. The scheduler can range from a simple round robin scheduler to a more complex one which optimizes some parameters such as system latency.

4.2.2 Stream Parser

The stream parser component of the distributed transcoding architecture is responsible for properly merging the already processed parts from the different nodes. This component may receive processed parts in different order than the normal playback order, however along with the splitter it will keep track of the were about of each unprocessed segments using a message tag and the additional headers (see figure 4.4).

In general the stream parser utilizes one of the headers and the tag as shown in 4.4 in order to re construct the processed stream in to a single output stream.

4.2.3 Scheduler and Load Estimator

Two possible scheduling methods have been tested in this experiment. Dynamic scheduling and static scheduling

Static Scheduling

In this case the scheduler needs to know the available computational resource from each compute node and need to assign different number of GOPs to the compute nodes according to their capability. This means profile of each node containing their estimated relative transcoding capacity need to be known in advance and be stored to be used by the application.

Dynamic Scheduling

In this approach there is no need to know the capabilities of compute nodes in advance. Here an atomic unit (e.g. N GOPs) is sent out to nodes for processing. Whenever nodes finished processing their task they acknowledge the scheduler (master node). The scheduler then assigns the next segment to this node thereby balancing the overall load according to the capabilities. The algorithm for scheduling and load balancing is stated in the following pseudo code listing.

```
Decide on atomic unit to use(N GOPs)
while(!end of stream){
   read stream to buffer
   split the stream in buffer to atomic units(N gops)
   while(atomic units are available) {

      /* this way nodes can work on one unit while waiting
```

```
        acknowledging and receiving another unit for next time:
        interleaving communication with computation */

        send two atomic units to all nodes
        wait for acknowledgement from all nodes
        send one atomic unit in response to every ack.

    }
}
```

The pseudo code listed above shows the algorithm used to dynamically assign tasks to each process running a transcoder. Notice that the scheduling algorithm tries to load balance the processing through constantly monitoring the progress of each node. For this algorithm to work effectively the atomic unit used should be decided. Analysis of how this choice affects some of the system properties such as latency, total transcoding time and load balancing among nodes shall be found on the experiments section.

4.3 Transcoding Capability

Transcoding capability (TC) is a term we shall use to express the physical machine that is available to us to run a transcoding job (process). In the implementation each process will be assigned to a transcoding capability via MPI's process management system. Transcoding capabilities could be of different types and vary from low power arm based processors to high performance multi-core processors.

4.3.1 Transcoder

The term transcoder is used to express the program that shall be used to perform the actual transcoding. Here we used Ffmpeg [6], an open source cross-platform solution(tool) to record, convert and stream audio and video.

From our implementation point of view the transcoder is a black box and is an abstract entity however the basic components (libraries) are discussed in the following paragraphs. In the next section we shall also look at the basic internal workings of the transcoder implementation and the slight modification made in-order to make it work in a distributed environment.

Ffmpeg utilizes three notable libraries. According to Ffmpeg documentations [6] the libraries include:-

- Libavformat: an audio/video codec library, which contains a set of containers (video file format), multiplexers and demultiplexers. This library already supports parsing and constraction several video formats. It also contains different IO utility functions used to read streams from different resources. The following figure shows the basic idea behind the need and structure of video containers and where the libavformat library plays a role.

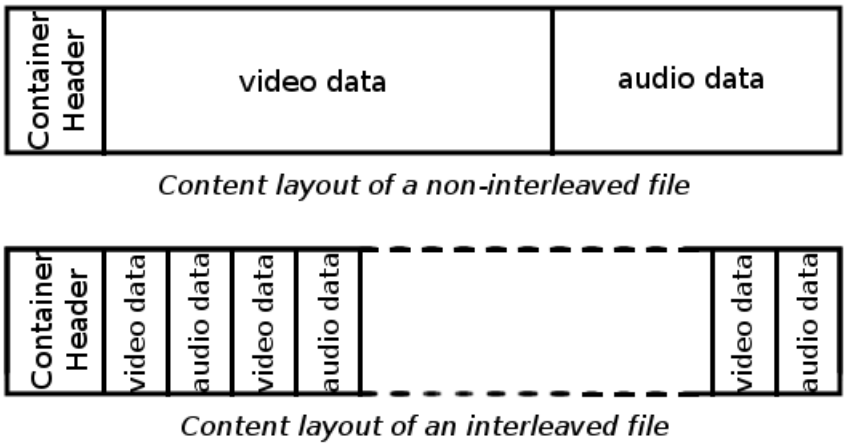

Figure 4.5: Video container in a video stream or file

Figure 4.5 Illustrates how video container is used to encapsulate both video and audio streams while providing synchronization and streaming information. Video containers, therefore contain all the meta data needed decode and stream video content. Libavformat is thus, a library that is able to construct and parse such containers to wrap around video and audio streams.

- Libavcodec: an audio/video codec library, which contains a set of audio/video encoders and decoders. This library already contains implementation of several known codecs. Codecs are the components of the transcoder that are responsible for encoding and decoding the video and audio bit streams as discussed in chapter 2.
- Libavfilter: an audio/video filter library. Which contains a collection of filters that could be applied on a given audio/video stream. Such things as cropping and watermarking are possible through this library. This library is also responsible for components used for interpolation and other such frame processing tasks that are needed for frame resizing while resolution transcoding.

The distributed transcoder instantiates several Ffmpeg processes and assign each of these to a physical machine (transcoding capability). Each of these processes will then communicate with the master process that has the task of partitioning the video and assigning it to them according to a given scheduling algorithm such as listing 4.2.3.

The following figure shows the general sequence of operations of the distributed transcoder, especially the communication among transcoding instances and the master node.

Figure 4.6 especially illustrates how the communication and computation are overlapped. It also shows how several transcoding processes operate simultaneously through send and receive messages.

4.4 Ffmpeg modifications

Some modification have been made to make Ffmpeg work with the MPI framework and the architecture proposed. The following figure shows the activity diagram of the original Ffmpeg transcoder along with the the changes applied to the transcoder.

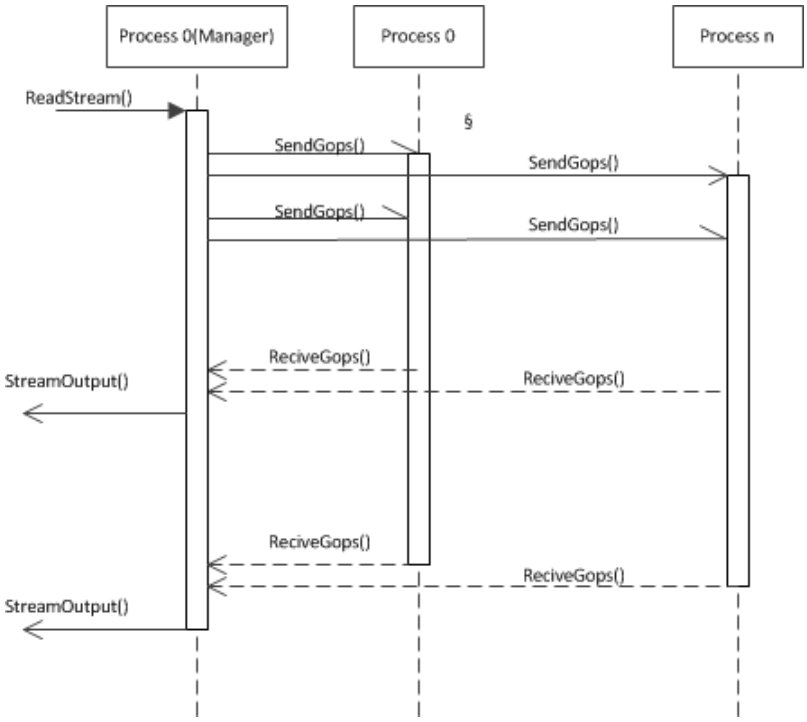

Figure 4.6: Sequence diagram depicting the interaction among different components of the distributed transcoder

First part of figure 4.7 illustrates the main operations that the transcoder implementation (Ffmpeg) goes through when dealing with audio and video streams. The main idea is to parse the format header for the stream meta-data and find an encoder from the *libavcodec* library to deal with the encoding. The second part of the the figure illustrates the modification made in-order to utilize the original transcoder in the context of the distributed one. Since our distributed transcoder uses GOP as an atomic unit of data distribution among multiple nodes, each transcoding instance (process) needs to wait until there is no transcoding tasks (GOPs) are left to be done.

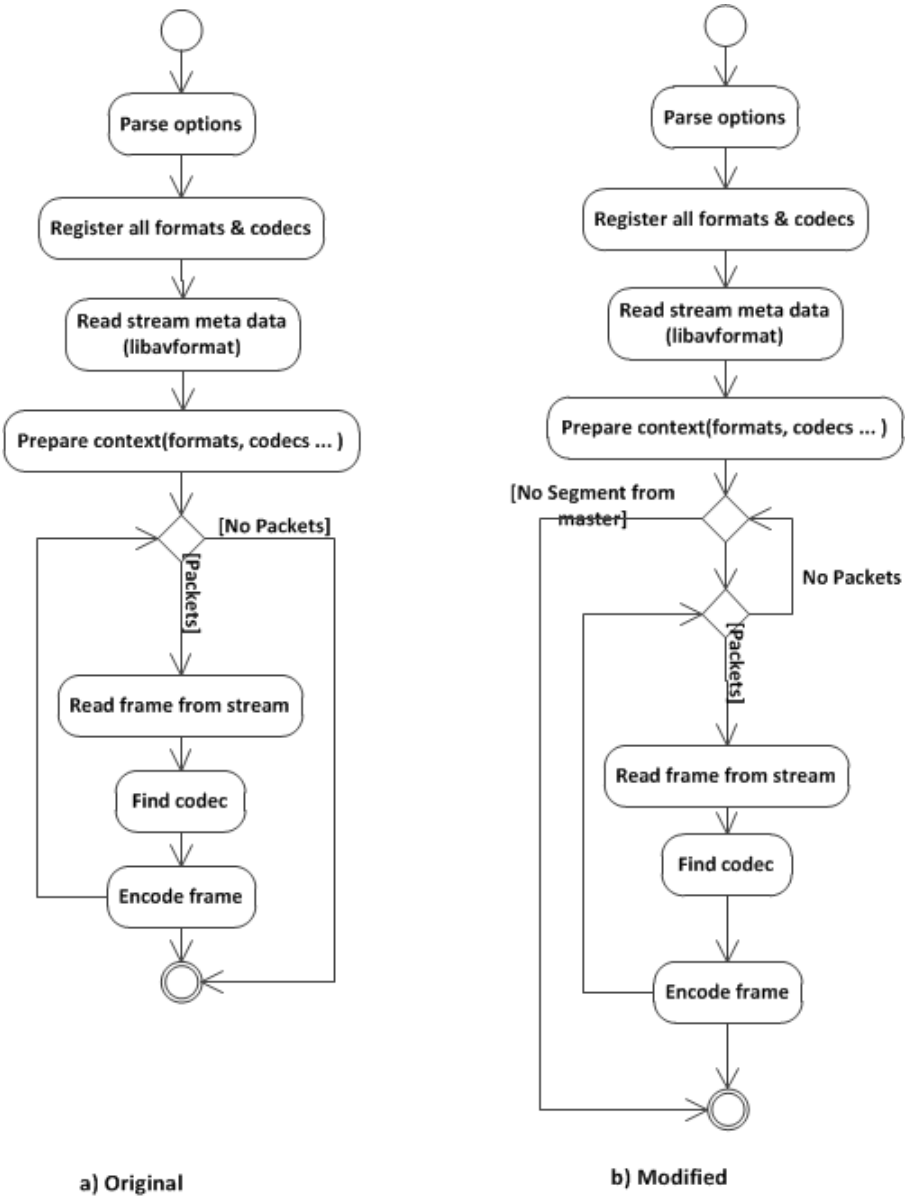

Figure 4.7: General operation of Ffmpeg and slight modifications made.

CHAPTER

FIVE

EXPERIMENTS AND RESULTS

5.1 Experimental Set-up

The experiments that are made in these section utilize two types of platforms. The first platform is the Amazon cloud together with the StarCluster and the second one is a private cloud consisting of 6 low power ARM based nodes along with with a dual core PC. The following table shows the main properties of the test platforms.

Platform	Description
Platform-1	Private cloud.
	ARM based physical machines.
	Six ARM Cortex A8 nodes (1 core, 512MB).
	One PC (core 2 duo, 2.26Ghz, 4GB,).
	Used for heterogeneous system test.
Platform-2	Public cloud.
	Virtual machines.
	Small instance (1 core, 1.7GB).
	One master node, 1-10 compute nodes.
	Used for system scalability test.

Table 5.1 lists the test platforms used in all the experiments in this chapter. Note that the Amazon cloud based cluster is used to test the scalability of the system in relation to the number of compute nodes utilized. And, on the other hand the private cloud is used in order to test the performance and functionality of the system in a heterogeneous environment.

5.2 StarCluster

StarCluster [4] is an open source cluster-computing toolkit for Amazon's Elastic Compute Cloud (EC2). StarCluster has been designed to automate and simplify the process of building, configuring, and managing clusters of virtual machines on Amazon's EC2 cloud. StarCluster allows anyone to easily create a cluster computing environment in the cloud suited for distributed and parallel computing applications and systems with a set of shell commands. With StarCluster compute nodes could be added (attached) to an existing cluster with a simple command. This fact enables us to test the scalability of our transcoder and its performance in the cloud.

5.3 Test Video

Tests are performed on a sample video of 10.95 seconds playtime containing 14315 frames [7]. The relative transcoding speed (in fps) of this video file for different transcoding capabilities are listed in table 1.1. The tests in the following section are made while transcoding from 4CIF to HD720 rsolution and a bit rate of 200kbits/sec.

The following sections shall explain some of the results in detail for our distributed transcoder implementation.

5.4 Total Transcoding Time

The Total transcoding time is the time it takes for the distributed transcoder to finish transcoding a bounded stream or a video file. The performance of a distributed transcoder could be measured with this value if the requirement on the latency of the system is not so much important. This means the user is less interested in when the transcoded video stream can be started to be viewed. In this case larger chunks can be distributed to each compute nodes reducing the communication and synchronization overhead and in turn resulting in a reduced total transcoding time. However, often users require a lower latency and are much eager to see the transcoded stream becomes available in which case the smaller video chunks should be used in an interleaved manner such that the streaming behaviour of the stream becomes unaffected.

The following figure shows the total transcoding time spent in utilizing different number of processing units in the cloud (platform 2, see table 5.1).

Figure 5.1 illustrates how the number of additional processing units affect the performance of the system in terms of total transcoding time. The type of the processors used are similar (m1.small machine instances in AWS cloud, see 5.1). As it can be noticed the scalability of the transcoder is not linear and tends to saturate when the number of processors involved increase. However this is

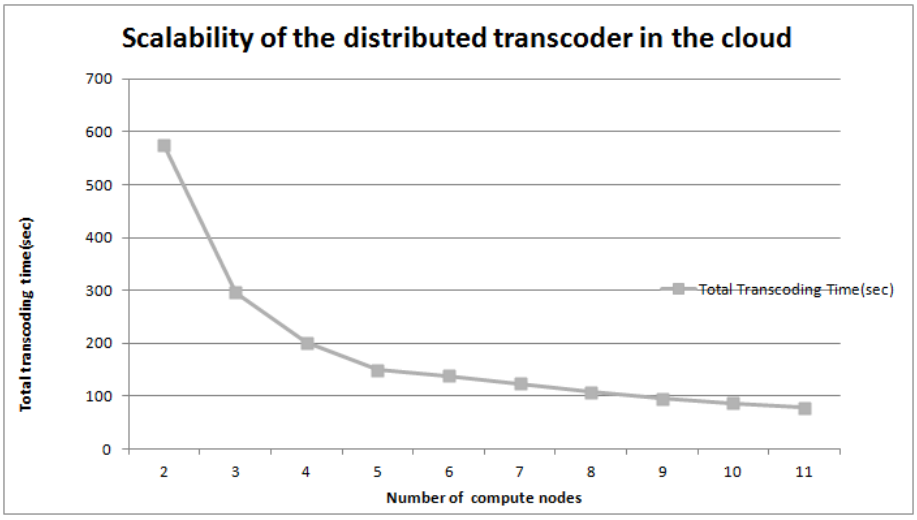

Figure 5.1: Total transcoding time while utilizing different number of cloud virtual machines.

expected as transcoded video streams requires increased synchronization and communication and at the same time less data to process relative to individual compute nodes per transcoding capabilities. this fact can easily result in

The following figure shows the load balance achieved while utilizing 11 processors (1 master and 10 transcoder nodes). The scheduling method used is round robin where the master node sends a specific number of GOPs while splitting the stream.

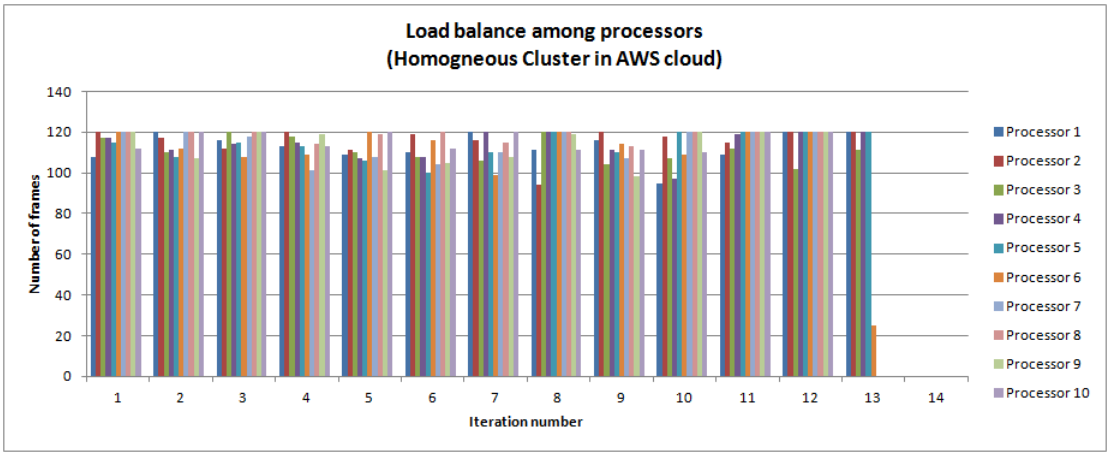

Figure 5.2: Load balance in terms of number of frames processed among homogeneous nodes in a cloud cluster.

Figure 5.2 illustrates the load distribution in the system utilizing 11 processors including a master node and 10 compute nodes that does the actual transcoding. Notice the overall pattern on the total load distribution as being almost equal for all the processors over all the iterations. By number of iteration we mean that we load each of the ten processors 12 times resulting in 120 chunks being processed out of the total 126 partitions. On the last iteration the remaining 6 chunks are assigned to the six processors depicting a round robin scheduling. The load distribution is made in terms of group of pictures as it has shown a better performance than either frame based or size (in MB) based portioning [11]. Therefore each of the ten bars in figure 5.2 show the same number of GOPs but with slightly different number of frames in each. Figure 5.3 shows the load distribution in the system as viewed in terms of the processing time rather than the number of frames as presented in the figure 5.2.

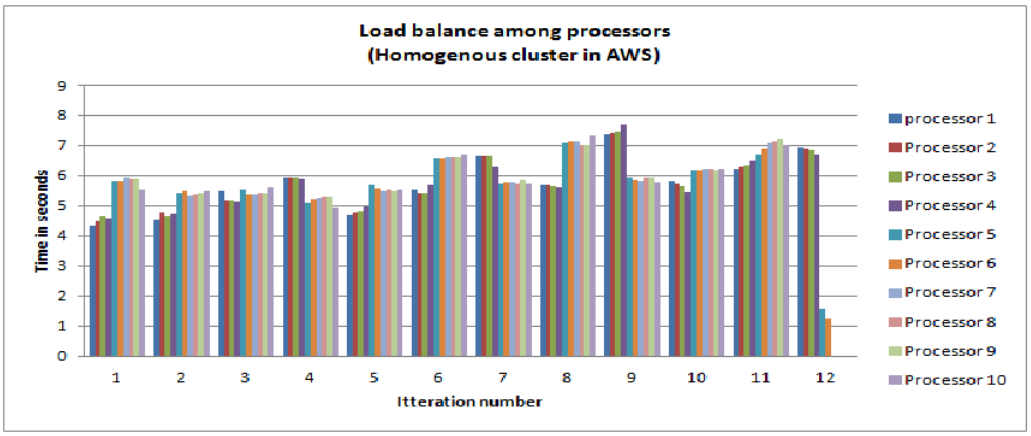

Figure 5.3: Load balance in terms of processing time of frames processed among homogeneous nodes in a cloud cluster.

Figure 5.3 illustrates the load balance of the distributed transcoder implementation on a homogeneous Cloud cluster nodes in terms of processing time. It can be noted from the figure that the load distribution looks good but improvements can be made with the help of a good transcoding time estimator.

Figure 5.4 also illustrates the load balance on a test performed under a private cloud (see platform 1, table 5.1). Here the heterogeneity of the nodes is clear and is also exhibited by the load balance achieved by the distributed

transcoder.

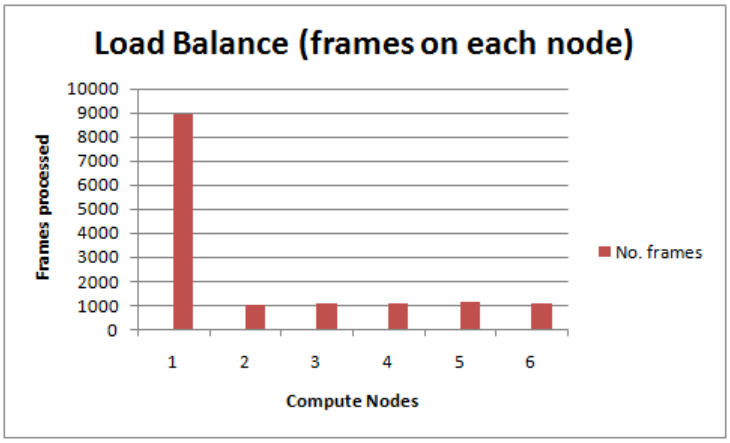

Figure 5.4: Load balance in heterogeneous environment.

Figure 5.4 illustrates the results of static scheduling depending on a pre calculated capability estimation of the different nodes found in the private cloud. Pre calculation of transcoding capabilities was done on the different platforms and could be seen from table 1.1. Accordingly static variables are used corresponding to these capacity estimates to be used by the scheduler in order to load balance the input stream.

Figure 5.4 shows how the five similar nodes (ARM based processors) processed approximately equal but considerably less number of frames than the first compute node which is of better performance platform (see Table 5.1).

5.5 Start-up time

We define start-up time as the time it takes to complete a set of chunks that would try to produce a possible jitter free playback for an on demand transcoding.users often demand for the shortest possible start-up time. The concept of start-up time is also directly related to latency of the system. latency of the system is measured as the time it takes for a chunk to be processed and becomes available to the user. As the latency of the system decreases the start up time will also gets smaller as well. one way of decreasing the latency of

the system is to use smaller chunks. As the size of each chunk gets decreased the amount of time required to process these chunks on each processor gets decreased.

The following figure shows the start up time in relation to the number of chunks used for an on demand video transcoding for the sample video [7]. It also shows the total transcoding time achieved while using the different chunks used.

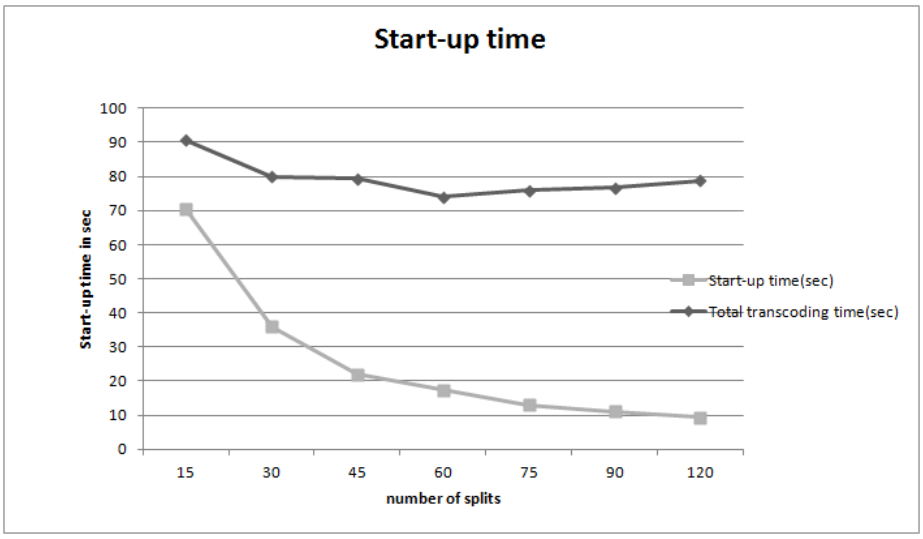

Figure 5.5: Startup time for diffrent split sizes.

Figure 5.5 shows the video transcoding through utilizing 11 compute nodes (10 worker nodes and a master) and different number of chunks ranging from 15 to 120. Notice that the performance of the system increases in terms of start-up time as the chunk size gets reduced. Interestingly, the total transcoding time also gets slightly decreased, possibly due to smaller chunks resulting in an increased utilization of the compute nodes in the system. The fact that processing smaller chunks resulted in a lesser start up time comes from the reduced latency of the system for small chunks.this fact will be looked at in section 5.7.

5.6 Scalability

Scalability of the distributed transcoder can be seen through looking at how the system performs with respect to an increase in the number of processors. For this experiment the number of chunks used are kept constant for the reason that communication and synchronization costs remain the same while utilizing additional processors (see figure 5.1).

5.7 Quality of services

In video on demand services, the concept of quality of service is important. Quality of service is usually used to measure how well a user is satisfied with a given service and can be measured through different ways and might have different meanings to different users of a given services. In this section we try to explain and measure some of them.

Among others, quality of a video content delivery system could be measured in terms of latency of the system. Figure 5.6 shows the response time of our transcoder utilizing 10 CPU units in Amazon cloud where the video stream is divided into smaller units and scheduled among the processors in a round robin manner. From section 1.2 we could remember that different individual transcoding capabilities could not handle that processing requirement needed for on demand streaming of a sample video. This means the response time of the system at times becomes smaller than the minimum latency requirement producing a jitter at several points in the stream.

Figure 5.6 illustrates the availability of the system measured in terms of frames. It shows the number frames that become available for playback at specified times while utilizing 10 transcoding nodes along with a desired frame availability required for a playback speed of 30 frames/sec and availability while utilizing a single ARM Cortex 8 Processor (see more on table 1.1). Note that for such configuration of the system a short start up and jitter free playback has been achieved.

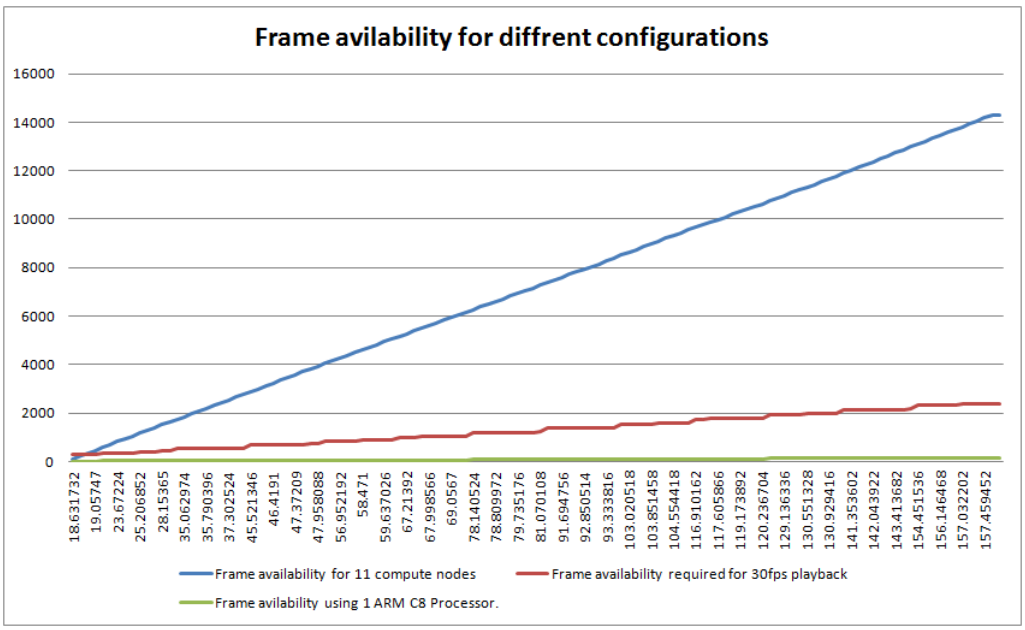

Figure 5.6: Availability and playback time of a given video file.

Another measure for quality of service that we have look at is the signal to noise ratio of the output video for various bit rate transcoding jobs. Note that the performance and scalability tests that are made are done in the previous sections are of resolution transcoding with a fixed bit rate at 200kbs. The following figure shows results obtained for a bit rate reduction transcoding of the sample video [7].

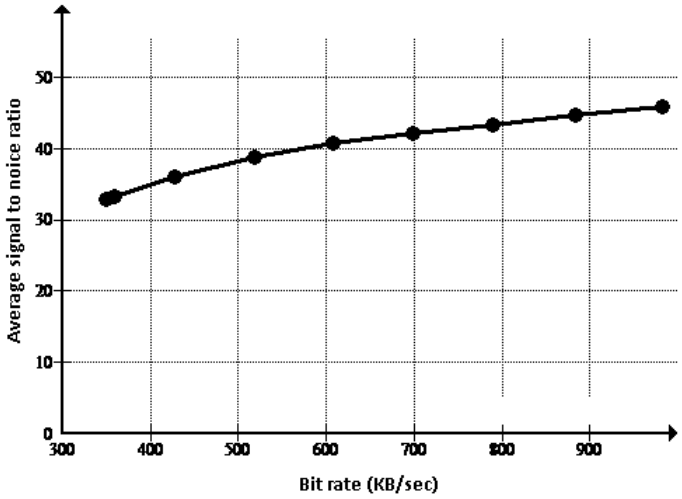

Figure 5.7: Signal to noise ratio of a sample transcoded output.

Figure 5.7 illustrates the quality of the transcoded output. The Average Peak Signal to Noise Ratio is often used to measure the quality of video and it is calculated as in the following equation.

$$PSNR = 10 \times log_{10}(\frac{MaxErr^2 \times W \times H}{\sum_{i=1,j=1}^{W,H}(x_{ij} - y_{ij})^2})$$

The x_{ij} and y_{ij} shows the pixel values of source image and compressed image. The W and H indicate the width and height of the image. From the results we can fairly conclude that the quality of utilizing the system for bit rate transcoding is also promising.

This thesis describes the design for scalable and distributed video transcoding architecture. this transcoder is based on a the MPI framework and FFMPEG, an open source implementation of a transcoder. unlike the original transcoder this implementation is modified to utilize the available distributed computational resources in order to provide a higher performance and quality of service.

The advantage of this approach is that it gives higher availability and scalability to the transcoder. in chapter one we have seen that transcoding is a computationally demanding task which requires high availability and scalability from the underlying platform that it runs on. at the same time distributed systems provide these required features. this makes a distributed transcoder implementation important.

CHAPTER

SIX

CONCLUSIONS

Currently the advances in distributed computing has provided us with new ways of utilizing computational resources. Cloud computing which is one form of distributed computing offers special feature such as delivering high grade distributed computing infrastructures as a utility. This means that any customer (user) can obtain an easy access to computational resources matching his demand on pay as you go bases. To this end some tests has been made and illustrated in chapter 5. The idea behind using distributed computing approach for transcoding was resource requirement how ever proper utilization of resources without over provisioning is also important and the cloud provides the elasticity required from the different resources utilized by a distributed transcoder.

The distributed transcoder was made using the existing transcoder implementation (tool), Ffmpeg. This tool is used as a process running on several nodes that are distributed across a network. In addition the MPI framework was used to take care of the communication and process management. In some sense the MPI provides most of the required communication protocols along with the process management and scheduling task. However the application level scheduling which involves splitting the stream, scheduling them among

several transcoding processes and merging the result back to a single stream are done with a custom made code.

Another important thing to note is that video content is very dynamic in terms of processing time and a very good estimation mechanism is needed to estimate the expected computational requirement. For this different approaches has been suggested. Accurate estimates are also important in order to deliver higher quality of services in terms of admission control or utilizing the elastic nature of the cloud. The scheduling and load distribution methods used in this thesis were very simple and utilizes a static values for different compute nodes in a given system. Auto scaling of the system according to dynamic analysis of an input video stream is not implemented. However suggestions are made for different approaches and experimental testes has been given on functionality, performance and scalability of the system.

CHAPTER

SEVEN

FUTURE WORKS

Distributed stream processing is a new area of research and requires new theories and practices than the classical distributed computing concepts that are used to deal with stored data in a batch processing mode. In stream processing the system has no stored information on the amount of data it has to deal with and has to have a clever way to accurately estimate this information for a proper utilization of resources and some sort of QoS.

It is believed that future distributed system has to deal with a stream of data more and more. Some examples could be search engines that have to deal with finding answers from a stream of inputs from on-line communities and different devices, sensor networks that are used in intelligent homes and infrastructures, video streaming servers and etc.

We will look at some of the challenges that we faced and believe that could become a good future work as categorized into theoretical and technical works.

7.0.1 Theoretical Works

- *transcoding time estimation*: it has been discussed, stream processing time estimation is very important and finding a good approach for such estimation possibly through comparison of different existing concepts would enable the system to become more robust in terms of quality of service and resource management.
- *modelling and simulation languages/frameworks*: System modelling and simulation plays important role in development of good systems (frame works). Some distributed system modelling tools already exist especially for classical distributed systems that do not deal with streams. however it will be good to have such frameworks for the future.
- *optimization*: Distributed processing system optimizations could be done for many parameters such as economic cost or quality of service in terms of latency or some other parameter. Different algorithms that are based on existing concepts from various fields could be studied and developed.
- *Functional parallelism* : it this thesis we have only tried to exploit data parallelism, however, it is also possible to map the different functions that make up a transcoder among several processors. This fact could enable us to utilize extra specific hardware such as DSP cards that are found along side the main CPU.

7.0.2 Practical Works

- *Minimal Transcoder implementation*: The transcoder used in this thesis work (Ffmpeg) is a wide collection of codecs and is very generic. A minimal version of this transcoder would be very important and helps us to focus once effort in the more theoretical and new areas.
- *MPI framework features*: The use of MPI as a starting point has definitely helped us to focus on the application. However a custom made communication framework would help future research.

Dynamic attachment of a new process into a pool of already running processes is a required feature in the MPI framework in order for dynamic scaling of the transcoder application at run time. This feature is promised to be implemented in the future versions of different MPI implementations.

- *Private cloud platform*: Full featured low power cloud infrastructures would be convenient to host such application as the distributed transcoder and a good contribution towards a green IT.

BIBLIOGRAPHY

[1] folding@home: A distributed computing project which studies protein folding. http://folding.stanford.edu/. Accessed: Jan 20, 2012.

[2] Fujitsu MB86H52: ASIC MPEG-2 to H.264 HD transcoder from fujitsu. http://www.fujitsu.com/downloads/MICRO/fme/documentation/m13.pdf. Accessed: Jan 20, 2012.

[3] The MPI standard: A specification for MPI implementations. http://www.mcs.anl.gov/research/projects/mpi/. Accessed: Jan 20, 2012.

[4] Starcluster: An open source cluster computing tool kit for amazon's EC2. http://web.mit.edu/stardev/cluster/index.html. Accessed: Jan 20, 2012.

[5] MPI: The complete reference. http://www.netlib.org/utk/papers/mpi-book/mpi-book.html, 1996. Accessed: Jan 20, 2012.

[6] Ffmpeg: Cross platform solution to record and convert and stream audio and video. http://ffmpeg.org/, Dec 2000. Accessed: Jan 20, 2012.

[7] Big Buck Bunny: A short computer animated film by the blender institute. www.bigbuckbunny.org, April 2008. Accessed: Jan 20, 2012.

[8] Amazon elastic compute cloud: Delivers scalable pay-as-you-go compute capacity in the cloud. http://aws.amazon.com/ec2/, Nov 2010. Accessed: Jan 20, 2012.

[9] Cloud Software Finland: Finland national cloud software program. http://www.cloudsoftwareprogram.org/, 2010. Accessed: Jan 20, 2012.

[10] ISO/IEC 144 6. Information technology â coding of audio-visual objects. Technical report, 2003.

[11] Jokhio Fareed Ahmed, Lafond Sebastien, and Lilius Johan. Analysis of video segmentation for spatial resolution reduction video transcoding. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems*, 2011.

[12] J.G. Apostolopoulos and S.J. Wee. *Video Compression Standards*. Wiley Encyclopedia of Electrical and Electronics Engineering, JohnWiley and Sons, Inc., New York, 1999.

[13] John G. Apostolopoulos, Wai tian Tan, and Susie J. Wee. Video streaming: Concepts, algorithms,and systems. Technical report, HP Laboratories Palo Alto, Sep 2002.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, Feb 2009.

[15] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, Boston, Massachusetts, 1997.

[16] Hamann C.J., Roitzsch M., Reuther L., Wolter J., and Hartig H. Probabilistic admission control to govern real-time systems under overload. In *19th Euromicro Conference Real-Time Systems*, pages 211–222, 2007.

[17] Andrey Khorlin. Scheduling in distributed stream processing systems. Master's thesis, 2006.

[18] G. Morrison. Video transcoders with low delay. In *IEICE Trans. Communication*, pages 963–969, 1997.

[19] Michael Roitzsch and Martin Pohlack. Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 271–280, 2006.

[20] Michael Roitzsch and Martin Pohlack. Video quality and system resources: Scheduling two opponents. In *Journal of Visual Communication and Image Representation*, pages 473–488, 2007.

[21] Yasuo Sambea, Shintaro Watanambe, Dong Yu, Taichi Nakamura, and Naoki Wakamiya. High-speed distributed video transcoding for multiple rates and formats. In *IEICE TRANSACTIONS on Information and Systems*, pages 1923–1931, 2005.

[22] Gary Shao. Adaptive scheduling of master/worker applications on distributed computational resources. Master's thesis, 2005.

[23] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding architectures and techniques: An overview. In *Signal Processing Magazine, IEEE*, pages 18–29, 2003.

[24] Clemens C. W, Liesbeth Steffens, Reinder J. Bril, and Wim F.J. Verhaegh. Qos control strategies for high-quality video processing. In *15th Euromicro Conference on Real-Time Systems*, 2007.

[25] J. Watkinson. *The MPEG Handbook*. Focal Press, Woburn, Massachusetts, 2001.

[26] J. Xin, C. W. Lin, and M. T. Sun. Digital video transcoding. In *Proceedings of the IEEE*, pages 84–97, 2005.

[27] Andreopoulos Y. and Van der Schaar M. Adaptive linear prediction for resource estimation of video decoding. In *Circuits and Systems for Video Technology, IEEE*, pages 751–764, 2007.

[28] Jeongnam Youn, Ming-Ting Sun, and Jun Xin. Video transcoder architectures for bit rate scaling of h.263 bit streams. In *Proc. ACM Multimedia*, pages 243–250, 1999.

[29] Jeongnam Youn, Ming-Ting Sun, and Jun Xin. Elastic stream computing with clouds. In *IEEE International Conference on Cloud Computing*, pages 195–202, 2011.

APPENDIX

A

PROFILING RESULTS OF DIFFERENT TYPES OF TRANSCODING

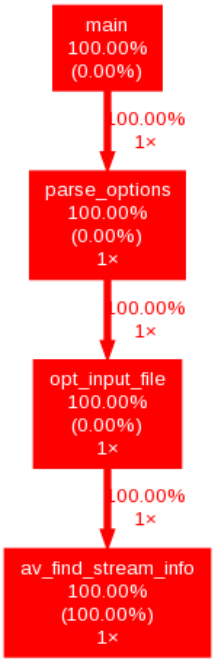

Figure A.1: Gprof profiling results as plotted by Dot for format transcoding alone with codec copy

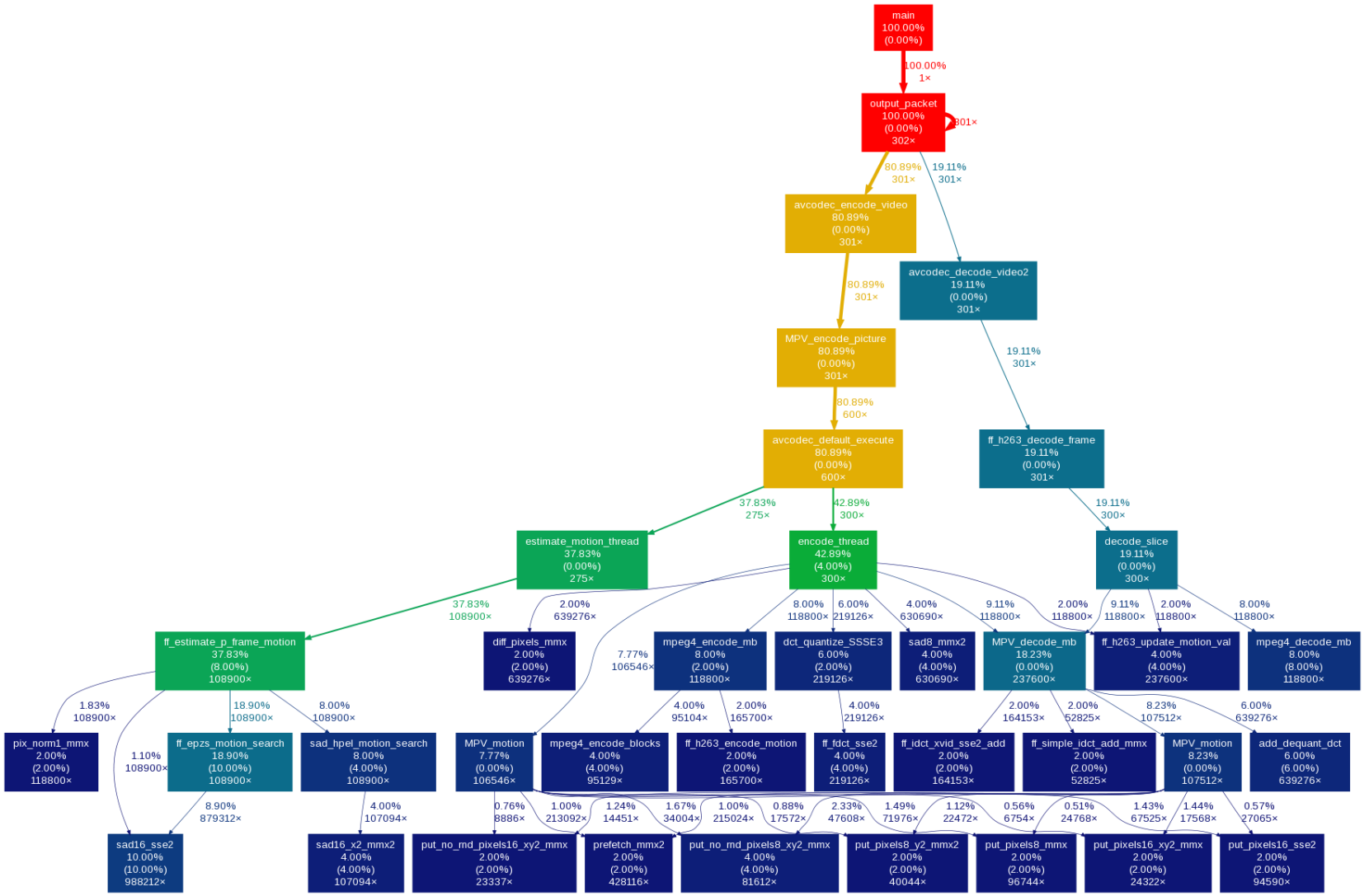

Figure A.2: Gprof profiling results as plotted by Dot for codec transcoding alone

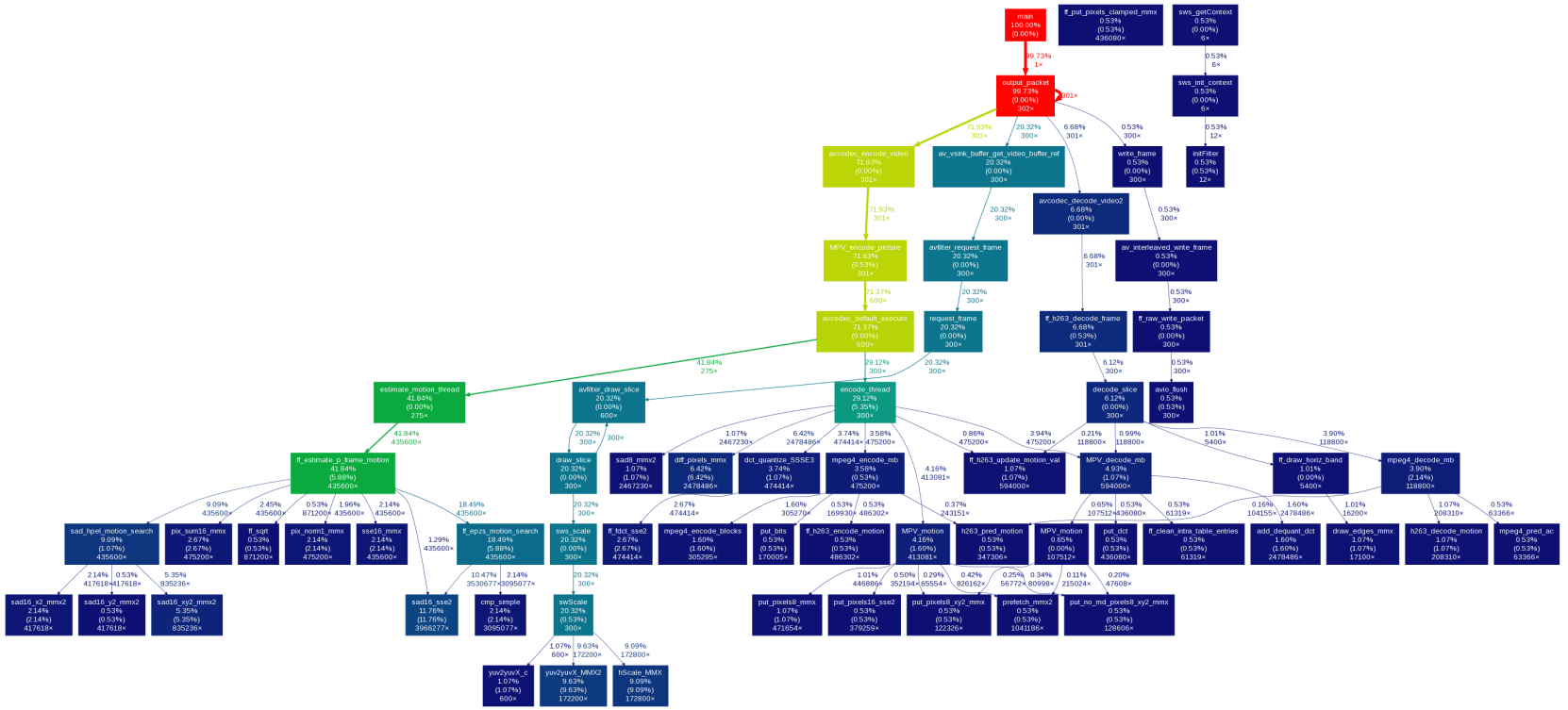

Figure A.3: Gprof profiling results as plotted by Dot for resolution and codec transcoding