Transcodificação de conteúdos multimédia utilizando um sistema de processamento em grid

Ricardo João Pereira Pinto

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Doutor José Carlos Martins Delgado
Orientador: Doutor David Manuel Martins de Matos
Vogais: Doutor Nuno Filipe Valentim Roma
        Doutor Leonel Augusto Pires Seabra de Sousa

Setembro, 2008
Index

Abstract.................................................................................................................................8

Resumo....................................................................................................................................9

1 Introduction..........................................................................................................................10
   1.1 Motivation.....................................................................................................................10
   1.2 Problem overview.........................................................................................................11
      1.2.1 Compressed data .................................................................................................11
      1.2.2 Many formats ......................................................................................................12
      1.2.3 Dependency of data ............................................................................................12
      1.2.4 Parallel Overhead ...............................................................................................12
   1.3 Requirements ...............................................................................................................12
      1.3.1 Scalable video delivery ......................................................................................13
      1.3.2 Finding the right amount of nodes ....................................................................13
   1.4 Thesis Objectives ........................................................................................................13
   1.5 Overview of the approach...........................................................................................14
      1.5.1 Command Generation ........................................................................................14
      1.5.2 Command Execution ..........................................................................................14
      1.5.3 Slave Node .........................................................................................................15
   1.6 Summary .....................................................................................................................15

2 Background and Related Work..........................................................................................16
   2.1 Video is everywhere ....................................................................................................16
   2.2 History of transcoding ..............................................................................................17
2.3 Video bit rates ........................................................................................................17
2.4 Video Processing ...................................................................................................18
2.5 Current Standards ................................................................................................19
  2.5.1 MPEG-1 ............................................................................................................19
  2.5.2 MPEG-2 ............................................................................................................20
  2.5.3 MPEG-4 ............................................................................................................20
  2.5.4 VC-1 ................................................................................................................21
  2.5.5 Comparison .......................................................................................................21
2.6 Compression ..........................................................................................................22
  2.6.1 DCT compression ..............................................................................................22
  2.6.2 Motion compensation .......................................................................................23
  2.6.3 Summing up .......................................................................................................24
2.7 What is available? ..................................................................................................25
  2.7.1 Mencoder ...........................................................................................................25
  2.7.2 Transcode ..........................................................................................................26
  2.7.3 ffmpeg ...............................................................................................................26
  2.7.4 Microsoft Media Encoder .................................................................................26
  2.7.5 Comparison .......................................................................................................27
2.8 Related Work .........................................................................................................27
2.9 Summary ................................................................................................................28
3 Distributed Architecture .........................................................................................29
  3.1 Common architectures ...........................................................................................29
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>A parallel transcoder</td>
<td>41</td>
</tr>
<tr>
<td>3.9</td>
<td>Existing parallel transcoders</td>
<td>42</td>
</tr>
<tr>
<td>3.9.1</td>
<td>DVD::RIP</td>
<td>42</td>
</tr>
<tr>
<td>3.9.2</td>
<td>VisualHub</td>
<td>43</td>
</tr>
<tr>
<td>3.9.3</td>
<td>Summing up</td>
<td>43</td>
</tr>
<tr>
<td>3.10</td>
<td>Summary</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Chosen Architecture</td>
<td>44</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Wrapper based architecture</td>
<td>46</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Modes of operation</td>
<td>48</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Disadvantages</td>
<td>51</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Evolution</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>How does it work?</td>
<td>53</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Load modules</td>
<td>53</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Identifying the media</td>
<td>53</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Tool Selection</td>
<td>53</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Amount of nodes</td>
<td>53</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Setup</td>
<td>54</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Main</td>
<td>54</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Final</td>
<td>54</td>
</tr>
<tr>
<td>4.3</td>
<td>Core components</td>
<td>54</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Command Generation</td>
<td>55</td>
</tr>
</tbody>
</table>
4.3.2 Command Execution ...................................................................................... 56
4.3.3 Slave Node .................................................................................................... 57
4.4 Summary .......................................................................................................... 59
5 Results .................................................................................................................. 60
5.1 INESC grid........................................................................................................ 60
5.1.1 Benchmarks ................................................................................................... 60
5.1.2 1GB Encoding Tests .................................................................................... 62
5.1.3 7 GB Tests .................................................................................................... 70
5.1.4 Summing up .................................................................................................. 73
5.2 L2F Grid ............................................................................................................ 74
5.2.1 Benchmarks ................................................................................................... 74
5.2.2 1GB Tests .................................................................................................... 76
5.2.3 7 GB Tests .................................................................................................... 81
5.2.4 Summing up .................................................................................................. 84
5.3 Evaluating the results ...................................................................................... 85
5.4 Critical Factors ................................................................................................. 85
5.5 Summary .......................................................................................................... 87
6 Conclusion and future work .............................................................................. 88
6.1 Usage Scenarios ............................................................................................... 89
6.2 Future Work ..................................................................................................... 89
7 Bibliography ....................................................................................................... 90
Abstract

This thesis presents several solution designs for distributed transcoding in a cluster environment. The strategy is to use commonly available tools such as mplayer, mencoder, ffmpeg and transcode glued together with PERL to provide a system that can support a wide amount of input formats and provides efficient transcoding for most formats, thus avoiding the need to develop new code. This kind of strategy closely mimics commercial off the shelf acquisition strategies.

During the development of the system various different problems were encountered and solved. In particular, the splitting of video and merging of video are areas where multiple solutions were tested to determine which were the most efficient. The main contribution is the demonstration that it is possible to develop an efficient video transcoder using already existing open source encoding tools and the presentation of some common pitfalls when trying to implement one.

The advantages of this implementation are that it is fast to develop, is robust as it uses tools that are time proven, makes use of the computational power of the cluster to be able to process the video faster and has some intelligence build into it to adapt to what is installed in the system.
Resumo

Esta tese apresenta várias propostas de soluções para o processamento distribuído de vídeo num ambiente em cluster. A abordagem adoptada é a utilização de ferramentas existentes como o mplayer, mencoder, ffmpeg e transcode interligados através de PERL para a criação de uma aplicação capaz de suportar uma grande variedade de formatos de entrada e fornecer transcodificação eficiente para a maioria dos formatos, evitando assim que se tenha de desenvolver código novo. Este tipo de abordagem segue de perto a filosofia das estratégias de aquisição “commercial off the shelf”.

Durante o desenvolvimento do sistema diversos problemas foram encontrados e resolvidos. Em particular, a segmentação de vídeo e junção de segmentos de vídeo áreas onde diversas soluções foram testadas para determinar quais as mais eficientes. A principal contribuição é a demonstração de que é possível desenvolver um transcodificador de vídeo eficiente utilizando ferramentas abertas já existentes e a apresentação dos problemas que precisam de ser ultrapassados quando se tenta implementar um transcodificador distribuído.

As vantagens desta implementação são que é rápida de desenvolver, é robusta porque utiliza ferramentas maduras que existem há bastante tempo, faz uso do poder computacional do cluster para conseguir processar o vídeo rapidamente, e possui algum inteligência incorporada que lhe permite adaptar-se ao que está instalado no sistema.
1 Introduction

Multimedia content has become an everyday commodity. The general proliferation of advanced video services and applications result from the advent of efficient video compression standards, which have been developed to store and broadcast video information in digital form. However, once video signals have been compressed, delivery systems and operators frequently face the need for further manipulation and processing of such compressed video streams, in order to adapt their characteristics not only to the available channel bandwidth or storage medium, but also to the characteristic of the terminal devices. As a consequence, video transcoding has recently emerged as a research area concerning a set of manipulation and adaptation techniques, to convert one video bit stream into another bit stream with a more convenient set of parameters targeted for a given application (1). In this chapter, the basic technical problem and the approaches for tackling them will be described.

1.1 Motivation

The motivation to use a cluster of computers to transcode video is to be able to speed up the transcoding process when modifying the format of the encoding or the actual content that is encoded. However, there are technical challenges when distributing this task across several computational nodes, which need to be overcome.

The goal of this project is to create a distributed transcoding application that makes use of available distributed computer resources to speed up transcoding. To demonstrate how long it takes to transcode video a few tests were done. These tests were done on a Pentium D (2) at 2.8 GHz with two cores. The input media was a VOB MPEG-2-PS (3) 720x576 encoded video lasting about 30 minutes with a file size of about 1 GB. The conversion to MPEG-4 (4) was done using transcode (5) with the export module for ffmpeg (5). The second test done involved a resize to 640x480.

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion to MPEG-4</td>
<td>28m 57s</td>
</tr>
<tr>
<td>Conversion to MPEG-4 and Resize</td>
<td>50m 36s</td>
</tr>
</tbody>
</table>

The time needed for video processing with a single machine will grow linearly, so considering that the conversion and transformation of 30 minutes of video takes 50 minutes, it becomes clear why it is desirable for this process to be quicker. Looking at these results one can also see that converting
a video archive would take a long time as each video with about 2 hours of length would take about 4 hours to convert. Also, adding of video processing operations like resizing increases the time needed to process video by a substantial amount, in this case there was a 78 percent increase in the time needed for processing the video from adding a commonly needed resize operation. The wanted speed up will be gained through the distribution of the transcoding across multiple nodes.

1.2 Problem overview

There are limited amount of resources available in a computer and the sheer amount of compressed data that makes up multimedia content often makes transcoding a time consuming task. As a consequence, clustering a significant number of computers, thus adding their power together, may be considered as a feasible way to speed up the transcoding process. However, although splitting and distributing the work across several machines seems straightforward in many other applications, that in video processing this is not the case. In fact, the temporal dependency resulting from the adopted motion compensation prediction mechanism often imposes strict restrictions in the partition of the input video stream over several processing nodes. Moreover, the size of each partition must also be taken into account. If they are too small, the communication overhead can outweigh the gains from parallelizing the whole process, since the total computational power does not grow linearly with the number of machines participating in the computation.

1.2.1 Compressed data

Compression techniques have been perfected for multimedia content for many years now. These techniques make it possible to achieve very high compression rates though the loss of a small part of the encoded information. The loss of information is a tradeoff against the amount of compression achieved. Decompressing the 1 GB MPEG2-PS (3) file used in the initial test to raw YUV (6) will result in an over 30 GB file with the uncompressed information. When transcoding there is the need to retrieve the raw YUV (6) to be able to compress and encode it into another format. This will be challenging for most computers because it requires a high CPU usage to perform complex mathematical operation like decoding and encoding the raw content, high memory usage to store temporary decompressed video parts and I/O speed to read the input and write the output file both from and to disk or network.
1.2.2 Many formats

Any application wanting to be useful for transcoding has to support a reasonable amount of formats. This is a problem because over the years the amount of formats has been steadily growing. This applies to both video as well as audio formats. And even the standards are not always fully implemented or respected by implementations leading to various variations of a particular format. Another added source of complexity is the need for support for both split and merge operations on all formats to be able to perform the partition of the video as well as the merging afterwards.

1.2.3 Dependency of data

Most modern video formats go beyond the compression of each single video frame. They try to use temporal redundancy to further compress the video. This means they only compress the changes from one frame to the next. Even more recent are bidirectional dependent techniques, in which a picture is represented only by the differences between the previous and the next picture. This means that splitting the video in a random location will not work, as there is a high probability of choosing a frame that requires information from the previous or the next frame. There is also a difference in the order in which frames are stored and displayed.

1.2.4 Parallel Overhead

Processing the video to be able to split it before running in parallel as well as merging it at the end, adds an overhead to the overall time. If the nodes need to communicate, further overhead is added to the transcoding process. This is proven by Amdahl’s law, which states that the maximum theoretical speedup one can achieve is well below the ideal speedup. Keeping this overhead low is essential to be able to efficiently use the distributed resources (7).

1.3 Requirements

The above describes the problems one faces when trying to process multimedia content in a distributed system. These problems are not isolated between them but are related between them and to the problem of developing an effective, efficient and scalable distributed video transcoding application. With these problems in mind a few requirements can be established for the work that needs to be done.
1.3.1 Scalable video delivery

For a node to process video it need to be able to access the source video. This introduces the requirement for an efficient way to deliver the input file to the node and to retrieve the output files from the encoding nodes. There are several solutions for this problem:

- Send the whole file across the network. This requires more network traffic and disk I/O to save the file. The bottleneck in this method is the sending node, because it has to transfer the whole file multiple times.
- Multicast the whole file across the network. This allows the sending node to only send the whole file once, but requires that the multicast be reliable, because traditional multicasting is done by sending unreliable UDP to multicast addresses, which results in packet loss and input file corruption if nothing is done to prevent it. It also requires that all nodes to be already listening when the multicast starts, because once the sending node starts to send the file it is not possible to rewind or resend the missed segments.
- Using a network file system. In this case the available network file system is the Andrew file system. This allows the system to work transparently across the network for multiple nodes, but it does introduce additional I/O delay.

1.3.2 Finding the right amount of nodes

As was referenced before there is a point beyond which there is almost no gain in adding additional nodes and the addition of additional nodes can even have an adverse affect on the performance of the system. This introduces the requirement that the system should choose not all the nodes available to it but should based on the input file the decision of how many nodes would be useful for the task at hand. This will vary from cluster to cluster based on the type of processors available and the networks capacity. To determine how many nodes it needs the system can use the file size of the input file, its length in time or the speed of the nodes.

1.4 Thesis Objectives

This thesis presents an application that is able to transcode video in a distributed system using the condor queuing system. What distinguishes this application from existing applications is that is wraps around existing tools and therefore can remain up to date as improvements are applied to the tools it uses, as well as the command generation which can be adapted to the tools that are present. It uses ffmpeg (5), mencoder (8), mplayer and transcode (9) as tools.
The objectives for this thesis are:

- To develop a scalable distributed transcoding application.
- The application should scale and adapt to the target system where it is being used.
- The application should allow to test various methods of transcoding.

As storage capacity and bandwidth increases so do the demand for better video and sound quality in multimedia. New high definition formats can use up to 1900x1600 resolutions with very high video and sound bitrates. As this high quality content needs to be adapted for slower connection there is the requirement of a transcoding application that can do this fast enough. The bigger the content the more one notices the need for running the transcoding process in a distributed fashion.

1.5 Overview of the approach

In the previous section a few requirements were described for the approach. This section presents and overview of the approach to the problem.

The overall approach was to use already existing encoding application and try to use them in a distributed environment. This cuts development time and allows building a flexible application that can test many different scenarios to determine which is the most favorable for distributed video transcoding.

The key components for the approach are outlined below:

1.5.1 Command Generation

The application need to analyze the output of various tools to determine what kind of input it is being presented with. It then needs to determine how to transcode from the input format to the output format and how many nodes should be used. Once all this has been determined it can generate the needed commands to be run in parallel taking into account the parameters that are requested by the user.

1.5.2 Command Execution

Based on the command line options chosen by the user the application will execute the commands generated by the previous command generation stage. The execution of these generated
commands can vary, as there are different modes of operation available to the user. This means that depending on the mode selected different setups might be needed. This can include creating directories and setting up network connections. It then needs to execute the commands and monitor their execution.

1.5.3 Slave Node

To allow the nodes to execute complex tasks as setting up network connection and creating temporary work directories it was necessary to have the nodes run a small application that can handle these tasks.

1.6 Summary

This chapter was an overview of the motivation for the project as well as the challenges that will have to be overcome. One can see that the challenges are related between themselves. Starting with the large amount of data to be processed that can be in any format, which leads us to search for a parallel solution, which in turn raises the problem of splitting the data and the interdependency of the data as well as the need to merge back the chunks of processed data coherently together.

To succeed in implementing a good distributed transcoding application the main requirements are to be able to feed the working nodes fast enough and retrieve the information they produce. Also, determining the correct amount of nodes to use is a critical component to avoid introducing unnecessary delay. Distributed transcoding will strain not only the CPU but also the network, as it is not only a computationally intensive task but also requires fast I/O.

In the next chapter more background and related research will be presented as well as a further analysis of some of the problems already presented. In chapter 3 an overview of the distribution problem will be presented. In chapter 4 the implementation of the software will be discussed. In chapter 5 the experimental results will be evaluated and in chapter 6 the conclusions and problems for future work will be presented.
2 Background and Related Work

Since the internet evolved from text only websites into content rich websites and due to it becoming a tool for a ever growing amount of people, whether the content needs to be made more suitable for storage or for streaming, the need for transcoding exists and is on the rise. Transcoding is the process of decoding the media transform it and then encode it again. In this chapter, a brief history and state of the art will be presented, pointing out the relations between this thesis and previous work.

2.1 Video is everywhere

Nowadays distribution of videos through the internet is common. This can take many different forms from streaming user created content as in youtube (10) or company created content as in news agencies to downloading video files from video on demand services or peer to peer networks. These videos need to be made smaller to be delivered across the network as most consumers do not have enough bandwidth to download the high quality samples. To do this companies and users need to transcode their content into a more internet suitable format that is more appropriate for the low bandwidth users or offer different quality samples of their content so the user can choose what is the most suitable content for him. Most users don't have access to distributed clusters. However, most companies, if they are distributing video across the internet do have enough resources for encoding clusters for their content. If their business is delivering content it is almost a guarantee the existence of a transcoding cluster. The importance of transcoding for the content distribution industry is present in the International Data Group market reports (1) for 2006. This need for a transcoding application is one of the motivations of the current thesis.

Overall internet traffic in any network already has a sizable multimedia content share. This share cannot be precisely measured, because video is not only streaming video with specific protocols like RTP (11) and RTSP (11), it also involves peer to peer traffic from users trying to download videos, flash videos which are embedded into HTML (12) web pages and videos sent as email attachments. People are in general interested in video content and are using the internet to deliver it.
2.2 History of transcoding

The transcoding of video has its roots in the beginning of the 90s when ordinary consumer quality computers became powerful enough to display video and various companies introduced their own video and audio formats. H.261 (13) was the first video encoding standard with large deployment and subsequent standards like MPEG-1 (14), MPEG-2/H.262 and H.263 (15) are based on its design.

Soon companies responded to costumer interest in video content. Apple was one of the first ones when Bruce Leak from Apple presented the first version of Quicktime (16) it was a major breakthrough to be able to play video in a consumer quality computer. Soon other companies like Microsoft with the Windows Media Video format and Real Networks Real System joined in. This leads to the generalized use of video as more and more people began using video and audio from the MPEG-1 standard and others. Today consumer cameras store their information in digital formats on flash memory abandoning the analog film, making them interoperable with the computers.

With the ubiquity of the electronic devices consumers now demand the ability to move content between devices and that content to be accessible from any device. As different devices are prepared for different formats it is up to the content providers to provide the same content in multiple formats so that the consumer can choose the correct format for himself. A practical example of this is news sites offering the same content in multiple formats. Another common practice is the availability of any to flash converters, which convert user uploaded files to flash video format, so that the visitors only have to be capable of processing that format. Adapting video files for streaming is also a common task.

2.3 Video bit rates

The bit rate of a video is determined by how much the compression algorithms can shrink a sequence of images. These images have a certain color space, which is a mathematical model that represents existing colors from a set of basic characteristics. These can be base colors like in the case of the RGB model, which chose red, green and blue as the base colors for the model, or it can use other characteristics like luminance and chrominance as in the case of YUV.

Video was first developed as an analog technology and therefore the uncompressed signal has an extremely high bit rate. A TV signal with a resolution of 720x480 at 30 frames per second using the
YUV color space with 12 bits per pixel will have an uncompressed bit rate of 125 Mbps. This kept the processing of digital video to only specialized machines for quite some time. However, once the compression techniques were introduced it became possible to have much lower bitrates. For example a MPEG-1 commonly uses a rate of 1.5 Mbps (14). Moreover the hardware has had substantial improvements over the last 15 years. This reduction in bit rates makes it possible to reduce storage and network requirements, making the multimedia content suitable to be streamed across the network.

To make this even lighter on the network several distribution strategies have been developed to reduce the network traffic when broadcasting the video across the network to multiple users. These improvements in multicast allow for the sender to send only once and the underlying network infrastructure do duplicate the packets. For this protocols like IGMP have been developed and routing hardware that is aware of the creation of multicast groups due to a technique known as IGMP snooping (17).

For transcoding, the compressed bit rate is an important factor, because it represents how much I/O will be needed to read and write the file to disk, which has always had slower I/O speeds than memory. Techniques as multicasting can be used do distribute the content to the nodes, but one has to guarantee that all the nodes are ready to receive data before starting to send and that the transmission is reliable. These requirements remove some of the benefits of multicasting, because it adds much complexity to the server which, unlike in the multicast streaming case, needs to verify that the nodes have received the pieces correctly.

IP multicasting allows sending to multiple recipients over a network. These recipients only have to tune into the IP multicast address to be able to access the data. Although not too common today it might become a mainstream application with IPv6 (18), as this is an area of research which has several applications in multimedia broadcasting. Even today several companies are implementing IP based multicast television. (19)

2.4 Video Processing

Processing of video can be interpreted in many different ways. Processing video can be displaying video, applying various visual filters or reading metadata from the video file. However, none of these operations is what we are interested in. Our purpose is to transcode video.
Transcoding is a digital-to-digital conversion from one codec to another. It works by decoding using one codec to an intermediate format and then encoding it into another. This allows for the conversion of the digital media, like videos and music to a format that better fits the limitations of the end device. However, as most codecs are lossy, the quality after each transcode will usually decrease the quality of the media, as more and more errors are introduced by the lossy compression. To prevent this from being a problem, the transcode should always be done from the highest quality medium available and not through a series of codecs as these yields progressively worse quality samples. Sometimes transcoding is used not to convert between codecs, but to modify certain parameters like bitrates or resolutions. This does not change the process, as frames are still fully decoded, manipulated and then recoded using the same codec as usually implementations do not optimize the transcoding process for that special case.

2.5 Current Standards

There are many standards (20), as well as, many variations from the standards. This is not intended to be a full list of available standards. But it is important to refer a couple of the most popular ones. These are the ISO/IEC MPEG (21) standards as well as the ITU-T (22) H standards. These standards represent most of the technology available for multimedia encoding.

2.5.1 MPEG-1

Designed for 1.5Mbps data rates, it is used for VCD (Video Compact Disk). It is the MPEG standard with most support, as it is also the oldest one. It is used when one wants to guarantee that everyone will be able to play back the video. The quality is highest at the resolutions MPEG-1 (14) was meant to have: 352x240 for NTSC (23) and 352x288 for PAL (24). The file size is not very good compared with more recent MPEG standards.

Its main limitation is the lack of support for interlaced images, which was one of the reasons for evolving to MPEG-2.

The well known MP3 is part of this standard. Its proper name is MPEG1 Audio Layer III. MPEG-1 also defines MP1 and MP2 audio standards (14). While MP1 is rarely used, MP2 is the standard for Video CD and Super Video CD. It is used because it's has better compression performance than MP3 on high bitrates and is more error resilient, therefore having good features for broadcasting. However, MP3, although still mainstream, is progressively been abandoned in favor of more recent codecs. One of the reasons is the need to pay royalties to the patent holders Fraunhofer (25) and
Thompson. Although the patent holders do not ask for patent fees from open source and free developers, they do ask it from any company using MP3 in their commercial products. The MP3 patent will expire in April of 2010.

2.5.2 MPEG-2

One of the most common standards used is MPEG-2 (26), it is the current standard for digital video broadcasting, so it is used for digital cable TV, digital satellite TV as well as for DVDs. MPEG-2 offers potential excellent video quality, as seen in the cases of DVDs.

DVDs implement a variant of MPEG-2 known as MPEG2-PS or VOB. It is a packetized elementary stream which adds navigation information to provide us with DVD menus. Another variant is MPEG2-TS, which is used to transmit MPEG-2 video and audio over wired channels, this is the case of systems which use DVB (Digital Video Broadcasting) and ATSC (Advanced Television System Committee). Both DVB and ATSC have many variants that take into account the medium over which the signal has to travel.

MPEG-2 Audio standard enhances MPEG-1 audio with support for more than 2 channels without breaking compatibility with MPEG-1. It also introduces an alternative not backwards compatible format, AAC (Advanced Audio Coding). This format is more efficient than MP3, but it also requires more powerful hardware to decode.

2.5.3 MPEG-4

MPEG-4 (27) is a set of standards that extend MPEG-2, and cover many aspects from audio to video encoding. However, MPEG-4 is still troubled by licensing issues. Although many implementations exist, it is unclear who owns the patents, which leads to legal problems as is the case between Apple and AT&T. MPEG-4 is also known as Xvid or DivX. MPEG-4 is expected to replace MPEG-2 in digital broadcasting.

MPEG-4 part 2 was designed to have object-based compression with makes it ideal for web based content distribution. The reason for this is that object based coding allows for interactivity.

H.264, also known as MPEG-4 part 10, is a video codec which features better compression rates than MPEG-2. H.264 has already been selected as the preferred video codec for NATO as well as some digital satellite and cable broadcasting services.
In the audio area MPEG-4 redefines MPEG-2 Advanced Audio Coding (AAC), it is now split between multiple codecs, LC-AAC(Low Complexity AAC), HE-AAC(High Efficiency AAC), AAC-SSR(Scalable Sample Rate AAC), BSAC(Bit Sliced Arithmetic Coding).

2.5.4 VC-1

This is the SMPTE 421M video codec standard, which is implemented by Windows Media Version 9. Microsoft has for sometime called some of their proprietary codecs MPEG-4 also known as Windows Media Video Format (28) or MSMPEG-4, however, it has a different structure and has nothing to do with the actual MPEG-4 standard, although it makes use of the same base techniques of Discrete Cosine Transform and motion prediction compression. It is the video standard for Blue ray disks as well as HD-DVD. It isn't as widely used as the MPEG standard as it is newer and not as widely used.

2.5.5 Comparison

According to the National Institute of Standard and Technology (NIST) - Convergent Information System Division (CISD) these are the advantages and disadvantages of the MPEG family of standards:

<table>
<thead>
<tr>
<th>Standard</th>
<th>Advantage</th>
<th>Disadvantage</th>
<th>Bit rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG-1</td>
<td>Wide spread support</td>
<td>No support for interlaced video</td>
<td>Up to 1.5 Mbits/s</td>
</tr>
<tr>
<td>MPEG-2</td>
<td>Added support for interlaced video</td>
<td>Licensing requires a payment on development of playback platforms</td>
<td>1.5 to 15 Mbits/s</td>
</tr>
<tr>
<td>MPEG-4 Part 2</td>
<td>Added support for object based compression</td>
<td>Licensing fees</td>
<td>4Kbits/s to more than 1</td>
</tr>
</tbody>
</table>
An interesting trend that is noticeable is the growing bit rate of the standards. Following this trend the bit rate for the new high definition video is expected to be 80 MB/s with a resolution of 1920x1080.

All the mentioned formats are just a small amount of what a transcoding application can encounter. All the mentioned tools have been in development for a very long time by a large amount of developers to be able to have plug-ins for all these different formats. It is a time consuming and bug prone area as not all encoders produce standards compliant video files. This is important for the development of this application, as it is one of the reasons why it is a sensible idea to use already existing tools. If a custom application is to be developed the very least is to use the libraries provided by the various open source projects to have a relatively short development time.

2.6 Compression

The key for video encoding is the compression techniques (3), without them the video would be much larger. So a small introduction to the two most commonly used techniques is in order (29).

2.6.1 DCT compression

DCT stands for Discrete Cosine Transform. Any signal can be described through the sum of various frequencies and amplitudes. This means that any signal can be described through a Fourier transform. DCT uses as subset of all the frequencies needed to describe the signal with full accuracy. The information lost when using DCT is minimal, so how can more compression be obtained? The answer is to do a quantization step, which mean dividing all the DCT coefficients either by a constant or by a matrix. This will reduce most coefficients to zero, which makes the following compression step with run length and Huffman encoding very efficient, because most elements are equal to zero. There might be a big information loss in this step, but is not usually
noticeable to the human senses because most information is in a few low frequencies that can be reconstructed from just a few DCT coefficients. Although the signal was changed by the full compression process it is still usable, as the information lost does not affect the perception of the video or audio in a noticeable way.

2.6.2 Motion compensation

Motion compensation further compresses the video because between most frames of the video there is little movement or movement in the same direction. This can be used to reduce the amount of information that needs to be compressed, because it is possible to reuse part of the previous frame to create the new frame. This can be achieved through different motion compensations:

2.6.2.1 Global motion compensation

Global motion compensation is based on the movement of the camera during the video. It models most common camera moves, like panning and zooming, and therefore works best for still scene where the only thing moving is the camera. Its overhead is very small as it considers the whole frame. It was not designed to support moving objects inside of the frame and therefore doesn't work well in frames sequences with lots of moving objects.

2.6.2.2 Block motion compensation

Block motion compensation works by dividing the frame into block and tries to determine where the blocks are moving through motion vectors. This gives added support for moving objects within the scene. However, sometimes visual artifacts are generated along the border of the blocks. It is the method used in MPEG-1.

2.6.2.3 Variable block motion compensation

This technique is much like the previous one, but allows the blocks to be of different sizes. This allows the encoder to decide how big the motion blocks are. This way it optimizes the compression, because it allows for big areas like backgrounds to be assigned a big area, which mean less information added to the video. It is the technique of the most recent codecs like MPEG-4, VC-1 and H.263.
2.6.2.4 Overlapped block motion compensation

This form of motion compensation allows the motion blocks to overlap and therefore adds more precision and removes visual artifacts that can be created by flaws in the borders between blocks. This is the most advanced method of motion compensation and also the most complex and thus the least used. It is referenced in H.263 Annex F Advanced Prediction mode.

2.6.2.5 Motion estimation

Motion estimation evaluates for each motion block the direction in which it is moving, so that it assigns a vector describing the direction of the movement to each motion block. This improves the compression obtained, but it is also a complex and computationally intensive task, because the neighboring areas around each motion block have to be searched to determine the correct motion vector.

2.6.3 Summing up

These compression techniques create dependencies between the frames as they no longer can be processed without considering neighboring frames, because only the difference between two frames is stored. This created a problem when splitting video as the point of splitting cannot be random. This problem can be removed if all the nodes have access to the whole video sequence. This allows for the nodes to use the complete frames known as intraframes which are present in videos to be able to seek forward and backwards in a video sequence. The video is organized in groups of pictures (GOP), which are considered open when they end in a bidirectionally dependent frame and closed if not. ISO/IEC 13818-2 defines three types of frames: intra coded frames which are independent from their neighboring frames, predicted coded frames which are dependent from their previous frame and bidirectionally predictive frames which are dependent on both previous and next frame.

Besides the dependency between frames there is also a difference between the order in which the frames are shown and the order in which the frames are stored. Because of the previously mentioned dependency the frames are stored in an order that optimized the processing of the video sequence so that the player does not need to seek the file back and forward to be able to show the video sequence.

An example of the difference between how frames are stored and their display order below:
The compression techniques allow achieving very high compression rates and are something a distributed transcoder has to take into account. One approach is to split according to the group of pictures or other kind of structure that exists in the input files. This approach was used in the past (30) and allows for the accurate splitting of the file but introduces the need to scan the file do determine its structure (31).

2.7 What is available?

Most software that supports a large amount of formats usually uses a plug-in architecture where each plug-in adapts a particular encoding or decoding library to their own internal representation. This allows for a easy manipulation of the data after being decoded and works as an abstraction layer between the different video and audio formats and the core feature of the program. It also gives much more scalability as support for a new format is implemented by adding a new plug-in. Current available free software that can transcode between digital formats:

2.7.1 Mencoder

Mencoder\(^1\) is part of the mplayer package released by the mplayerhq team. It is built around the ffmpeg libavcodec that is also released by the same group. It also is able to use some binary codecs from windows in linux. And it had been ported from linux to many different systems like BSD, Solaris, HP-UX, AIX and Windows. It supports most common formats, like MPEG/VOB, AVI (32), Open formats like Ogg/OGM/Matroska, Microsoft ASF/WMA/WMV formats, as well as, Apple QT/MOV/MP4 formats. It supports many other less common formats with its own implementation of the needed codecs as well as the use of native XAnim, and Win32 DLL codecs. Although mencoder does not encode in all the formats that mplayer can read, it can still encode to an outstanding

\(^1\) [http://www.mplayerhq.hu](http://www.mplayerhq.hu)
amount of formats. Another interesting feature is the ability to add filters to the video processing chain. Each stage of video processing allows for user supplied filters to be used.

2.7.2 Transcode

Transcode\(^2\) is a linux text-based video-stream processing tool. It supports video and audio frame transformations. Its transcoding capabilities are limited by which libraries are available in the system, as it does not deploy its own implementation of the codecs, but relies on a series of standard libraries. It is able to use ffmpeg and should therefore be able to support at least as many formats as mencoder. It also has the option of parallelizing the transcoding by using a secure shell connection to other machines to spawn more transcode processes. It requires, however, that all machines have a shared network file system to be able to read the files. Input files have to be previously split and merged back together in the end. It also has experimental support for PVM.

2.7.3 ffmpeg

Ffmpg is a open source project that provides the libffmpeg library which is used by both mencoder as well as transcode. It also provides a tool that uses this library to transcode video and audio. Although it doesn't support as many formats as mencoder or transcode it still supports an outstanding number.

2.7.4 Microsoft Media Encoder

Microsoft\(^3\) is a solution for digital media encoding. It features, however, limited output format options, as only Microsoft’s own WMV formats are supported as output options. It is also a closed source application while the previous ones are open sourced and distributed under the GNU public license.

\(^2\) http://www.transcoding.org

\(^3\) http://www.microsoft.com/windows/windowsmedia/forpros/encoder/default.mspx
2.7.5 Comparison

Although there are other transcoding capable applications, these are the ones that are both free and most easily available for the project. The Microsoft Media Encoder is the least capable of the three and doesn't work on a Linux environment, so it is excluded from further comparison.

Both Transcode and Mencoder are very similar in features and architecture. Both have a plug-in architecture that provides interoperability with various codecs and libraries. This makes it very extensible as adding support for new formats is a matter of adding a plug-in that converts the input format to the internally used format, so that transformation operations can be applied. Depending on how complete the plug-in is it can provide either both importing and exporting functions or just one of them. This is why some formats can be imported but cannot be used as an export format.

Transcode might support more formats, because while mencoder is based on the ffmpeg library, transcode support other libraries besides ffmpeg, so it is expected to support more formats than mencoder.

2.8 Related Work

There are some interesting developments in data intensive super computing brought forth by the need to process large amounts of data. These new approaches put the emphasis on data access speed rather than on computational power, because the computational power is currently growing faster than the I/O speed of storage devices. (33)

These approaches which are used by Google, for example, have proven to be excellent in solving problems where there are large data sets to be analyzed. One of these approaches is hadoop\(^4\) which uses a map/reduce strategy, in which the map replicates the data in equal sized pieces across the network and then assign various reduce operations that run where the pieces are stored, exploiting the locality of the data.

However, using these on video can be problematic as current framework like hadoop requires uncompressed data to work properly because splitting compressed video in equal sized blocks makes processing the video particularly hard. Although this type of strategy can not be entirely

\(^{4}\) http://hadoop.apache.org
dismissed as this strategy if customized for video might yield interesting results. One option to be able to use such a system involves being able to transfer information between the nodes to adjust the boundaries of the splits. However, it might be necessary access to other parts of the input file like the index of frames typically at the start of the file or the streams can be in different segments, making this a subject for future work.

2.9 Summary

In this chapter, one can see that the transcoding of video is an area that has commercial interest for a large portion of the content distribution industry and the trend is to become more important not only to the business but also to the common user. One can see the growth of user created content in social network and other site like youtube, as well as the spread of multimedia through peer to peer networks. This growth can be explained by the introduction of efficient video compression that allows reducing analog signals that would require very high bit rates into sufficiently low bit rate file that can be shared across the current networks.

It also gives an overview of available video technology, standard and the tools available to accomplish the task of transcoding the video. Transcode, mencoder and ffmpeg are all good solution for transcoding video, as they can handle a very width range of input and output formats.

The same techniques that allow for a very high compression rates also make splitting the file difficult, however, there are several options open. Either providing access to the whole file works, or the file has to be processed to determine where the best cut positions are.

Recent developments in data intensive super computing show a great potential, but have so far not been introduced to video processing due to the need to adapt them to work with compressed interdependent data.

In the next chapter the options available for distributed work will be presented.
3 Distributed Architecture

The previous chapter presented some of the techniques and tools available to transcode video on one computer. In this chapter the tools and techniques available for the transcoding of video on several computers will be presented. When using more than one computer to solve any task, one has to consider that the total computational power of the computers is not equal to the sum of all the single computers. This happens because for each additional computer added to solve a parallel problem, it brings more computational power as well as more computational overhead.

3.1 Common architectures

Distributed architectures are the way in which the systems are organized, which can be distributed physically. The following architectures are common:

3.1.1 Client Server architecture

This is the most common way of organizing a computer system. It consists of one server that provides the services to the users that use client software. This software communicates with the server using an established protocol. The main advantage of this architecture is its simplicity. Despite not providing any kind of redundancy and therefore having a single point of failure on the central server, its simplicity makes it the most common architecture found in the internet today.

3.1.2 Clustered architecture

As the complexity of problems rise it becomes necessary to gather significant computation resources to solve them. These clustered architectures can be grouped in different categories, although some clusters might implement a mix of these philosophies, as they are tuned to the specific tasks they are needed for.

3.1.2.1 High-availability clusters

The goal of this type of cluster is to keep mission critical services up and running in spite of hardware failure. Their goal is to have the system running 24 hours a day all days of the year. It implements redundancy of all hardware and try to avoid having single points of failure. One such
project is the Linux-HA project\textsuperscript{5}. These systems are required to be able to keep track of their own components and react to failure of hardware or network components. They also provide support to data replication so that the services are always available. The level of redundancy usually matches the importance of the service or the financial resources available.

\subsection*{3.1.2.2 High performance clusters}

High performance clusters are optimized to solve particular computational intensive task. The simplest architecture would be a master and a huge amount of nodes that use a high speed interconnect to communicate with each other. The nodes are usually homogeneous and dedicated to computation. They are especially useful when the problems partitions depend on each other, making it necessary for the nodes to exchange information with each other. They are not transparent to the users who have to adapt their programs to take advantage of the provided libraries. He also has to decide on how many nodes to run his application on, as the more nodes he uses the more communication overhead he will have, which in some cases might outweigh the gains from distributing the problem across many machines. One way to prevent user from running processes on top of each other is to use a scheduler to manage the workload. This allows for a user to submit a job that will be run as soon as enough nodes are free without affecting the performance of the other users already running. Most high performance clusters are Beowulf clusters using MPI. Beowulf is more of a concept then an architecture as there is no strict definition of a Beowulf cluster.

\subsection*{3.1.2.3 Grid computing clusters}

A grid is an association of computers that can be physically distributed across several locations and can integrate very different computer platforms. This diversity of platforms can introduce the problem of having to be able to run a certain program in many different platforms that can be incompatible with each other. Grid can usually be grouped by different criteria like CPU architecture, hardware and software. These groups can be managed by workload management software like Condor, which provide job queuing, scheduling policies, priorities and resource managing, and toolkits like the Globus toolkit, which allows for more integration and portability of applications. An example of a large grid can be seen at distributed.net, where a massive amount of computers use their idle time to process various problems.

\footnote{www.linux-ha.org}
3.1.3 Peer-to-peer architecture

In this architecture the role of server and client is mixed as all members are equal. Each provides both server and client functionalities. In its pure form this architecture does not have any central servers. However, because of the difficulties of implementing a pure peer-to-peer system, most use central servers to allow the clients to initially find each other so they can cooperate.

One characteristic that differentiates peer-to-peer network is the overlay, which is the way in which the network is organized on top of the TCP/IP network. It can either be structured through a distributed hash table, which organizes the peers so that each peer is responsible for a part of either the network or the content, or it can be unstructured and not assign any responsibilities to the peers making it more difficult to do searches on the content, as all queries are floods to the network and there is no guarantee that they will reach a peer with that content.

3.1.4 Comparison

For computationally intensive tasks a high performance cluster is the best solution, because all the machines are dedicated to the task and provide high speed communications. For video processing either a grid or high performance cluster are equally suitable, because there is only a need for a high speed connection to upload the video for processing and to download the finished version. This because the processing of video that is aimed for does not use a parallelized algorithm, but rather splits the video and processes each one of the parts separately, which mean there is no need for the processing nodes to communicate with each other (34).

3.2 Architecture of a transcoder

A transcoder has one or more decoders which convert a input file into a internal format to whom transformations can be applied. This internal format, usually YUV or RGB for video and PCM for audio, makes the transcoder extensible as new formats only need a new decoder to be able to take advantage of all the feature and filter already present in the transcoder. This also applies to the adding a new encoder to be able to output a new format.
The decoder can be further divided into audio and video decoders which get their input from a demuxer which splits the video and audio streams of the input. Also, the input does not have to be a file it can be a RTSP stream or anything else.

So the general organization of a transcoder will have different building block that can fall into the following categories:

- Input adapters
- Demuxers
- Video and Audio decoders
• Video and audio filters
• Video and audio encoders
• Muxers

The popular adding of subtitles is either a video filter if the subtitles are rendered into the video directly or can also be part of the muxer features if they are kept in a different stream.

### 3.3 Existing transcoders architecture

Before developing a transcode one should look to the architecture of current transcoders, to learn from their year long experience of developing and maintaining transcoding software.

#### 3.3.1 Mencoder

Mencoder is a tool that is developed by the mplayer team and is known in the Linux community as one of the transcoding applications that supports the greatest amount of formats. It takes advantage of the whole infrastructure develop to support the mplayer player, which include the ability to use Windows binary codecs in Linux and all the open source media libraries.

It is written in C mixed with Assembly to achieve high performance in video decoding. The source code for mplayer includes the whole source code for some of the needed libraries instead of linking with shared library which makes the source quite large. Also, they don't use standard autoconf tool, preferring to use custom scripts to achieve the same results. They do have a modular structure in their architecture, but instead of implementing it through standard shared libraries that can be loaded at the start. They use a global structure in which all the modules register a function pointer to their code.

It follows the generic architecture described in the previous section providing an abstraction between the input and the stream, which is used to support playback of remote media file through many different protocols as well as playing directly from media readers as DVD drivers.
Figure 3.3 – Mencoder components

Mencoder is currently not under active development, as the mplayer team is more focused on mplayer, this according to their own members on their IRC channel.

3.3.2 ffmpeg

Ffmpeg is based around the libffmpeg and shares some of the developers with the mplayer project. It does not have such a broad user base as the mplayer project, although it updated much more often. It currently it is not distributed in releases, but directly from the project subversion repository. The project is very similar to mencoder and has a very similar architecture.

The release of projects directly from subversion repositories has the benefit of always providing the latest version, although it also makes it harder for the general public to access it and can in some cases be unstable. Luckily for most people all major Linux distribution have packages for this software.

3.3.3 transcode

Transcode also follows the previously defined generic architecture of a transcoder, but the implementation is very different from both mencoder and ffmpeg, which provide a monolithically build binary with the entire feature in it. Transcode provides a set of tools which work together through the use of pipes.

To support the large number of formats transcode has shared libraries that support importing, transforming and exporting media formats. In most cases these shared libraries do not actually implement the process of encoding or decoding a format but adapt the installed libraries to be usable by the transcode tools. This allows transcode to support a large amount of formats depending on which libraries are already installed in the system. It also allows transcode to stay up
to date with fixes in media libraries as the wrappers it provides in their shared libraries are only affected by big changes in the API of the wrapped library. Another interesting feature of transcode is that it provides support to distributed video processing through their frame accurate splitting for DVD format. Unfortunately only the DVD format is supported and not all other formats.

![Diagram of Transcode components](image)

**Figure 3.4 – Transcode components**

### 3.3.4 Summing up

A transcoder has modules that can be divided into demuxers, decoders, filter, encoder and muxers. Each phase is independent and has well defined input and output, this makes the transcoder extensible, because it makes it easy to add and remove components to support new formats. It is followed by all open source transcoder projects.

Comparing the performance of transcode piped architecture to the monolithic architecture of mencoder and ffmpeg one can see that that transcode should have a better speed improvement than ffmpeg on multiple core systems. The reason for this is that by using multiple processes transcode makes better use of available systems with multiple cores or CPUs. However, ffmpeg can also take advantage of the systems with multiple cores by using threads in the encoding process.
3.4 Parallel Overhead

Once one has parallelized the processing of video, one is tempted to use all available resources to speed up the processing, especially in a problem like this in which the nodes do not have to exchange information to be able to do their work. However, one should always take into account that the more nodes that are used, the greater the parallel overhead becomes. This overhead can be divided into three categories:

3.4.1 Extra Work

This type of overhead is created by tasks that are done because the video is being processed in parallel. The extra work in this case is the preprocessing of the video to be able to define point where is should be split. As well as the time needed to merge all the parts produced by the different nodes. There is also the time spent waiting in the scheduler queue before the job starts to run.

3.4.2 Network delay

This type of overhead is bound to network latency and the need to get the data to the nodes for processing. In this case the video sequence has to be sent to the node and the resulting encoded video sequence has to be retrieved from the node.

3.4.3 Sync delay

This type of delay happens because it necessary to coordinate the work of different nodes. Although in this case one does not have to sync the nodes between them, it is necessary to wait for all the nodes to finish transcoding before their output can begin being merged.

3.5 Parallel Metrics

To be able to quantify how much is gained from parallelizing a process, some metrics have been developed. This allows determining the best amount of nodes to use in transcoding.
3.5.1 Speedup

A common metric to measure the gains of parallelization is to calculate the speedup gained from parallelizing. The speed up show how much is being gained through the use of parallelization. The speedup does not grow linearly and has a maximum theoretical speedup according to Amdahl's law:

\[
Speedup_{\text{max}} = \frac{1}{(1 - f) + \frac{f}{N}}
\]

In the formula above, \( f \) is the proportion of a program that can be made parallel and therefore benefits from parallelization as it can be distributed by \( N \) nodes, and \((1 - f)\) is the proportion that cannot be parallelized and has to remain serial.

Using a graphic from this function we can how far the maximum speedup is from the ideal speedup:

So a good metric to measure how good the speed up was is to compare it with the maximum speedup derived from Amdahl's law. Below the equation that describes the formula to calculate the speedup.

\[
Speedup = \frac{\text{Time 1CPU}}{\text{Time N CPU}}
\]
3.6 What to parallelize?

One of the first issues to consider is which part of the transcoder we want to parallelize. According to Amdahl law the bigger the parallel part of the problem the bigger will be the gain of parallelizing. As we can't encode the video without first decoding and possibly applying filters to it, we have to follow the phases of a serial transcoding process. However, we can start the transcoding process in different parts of the file at the same time.

To determine the most CPU intensive stage each of the stages can be isolated and measured. For these tests a 1 GB VOB was used. The computer was a Pentium 4 laptop.

3.6.1 Demultiplexing

Demultiplexing to split the video into audio and video parts was done with:

```
#mencoder test.vob -of rawvideo -nosound -ovc copy -o dump.mpeg2

Time: 2m41 / 6s / 5s (real, user, system)
```

```
#mencoder test.vob -of rawaudio -oac copy -ovc copy -aid 128 -o dump.ac3

Time: 1m7s/ 6s/ 2s (real, user, system)
```

```
# tcextract -i test.vob -x mpeg2 -t vob > dump.mpeg2

Time: 2m32 1s 5s (real, user, system)
```

```
# tcextract -i test.vob -x ac3 -t vob -a 0 > dump.ac3

Time: 1m4s 1s 2s (real, user, system)
```

As expected, demultiplexing is clearly not a CPU intensive operation, but does require fast reading and writing in the file system.
3.6.2 Decode / Encoding

This encoding stage actually also involves decoding the video to an intermediate format that can be worked with. So this is the expected most CPU intensive stage.

```bash
# mencoder dump.mpeg2 -ovc lavc -lavcopts vcodec=mpeg4 -vf scale=352:288:1 -of rawvideo -ofps 15 -o file.mpeg
Time: 6m3s/ 5m40s/ 6s (real, user, system)

# transcode -i dump.mpeg2 -y ffmpeg -F mpeg4 -Z 352x288 -J fps=25:15 --export_fps 15 --hard_fps -o file.mpeg
Time: 15m53/ 10m56/ 1m13 (real, user, system)
```

This was clearly the step that took the longest time, as was expected.

3.6.3 Multiplexing

Multiplexing is an I/O dependent operation and therefore depends on the I/O of the system.

```bash
# mencoder -audiofile dump.ac3 -oac copy -ovc copy file.mpeg -of avi -o final.avi
Time: 5s/ 3s/ 0s (real, user, system)
```

3.6.4 Conclusion

Demultiplexing and multiplexing the video are I/O intensive operations, while decoding and encoding are CPU intensive operations. This means that the decoding and encoding is the main stage to parallelize, while the demultiplexing and multiplexing stage just need to be provided data fast enough so they do not become a performance bottleneck.

Another question that arises from this is whether demultiplexing and multiplexing could be done by multiple nodes. Demultiplexing can be done in parallel if multiple nodes start at a different point,
however one has to bear in mind that the file has to be transferred to the demultiplexing nodes. Demultiplexing is not computationally intensive enough to cover the cost of transferring the file to the node just to do the demultiplexing. Also, the multiplexing is a serial operation that should not be distributed across nodes as it is even less computationally intensive than the demultiplexing.

To maximize the efficiency of distribution of the work across the nodes one has to give the working nodes enough computational work so that is covers the extra work of sending the data across the network and other overhead introduced by the parallelization that was discussed earlier.

### 3.7 What to transfer?

To have nodes process the video in parallel we have to provide data to them. This gives us several possibilities:

- Process the input file so that it is possible to split the file and send only the needed part to each of the nodes. This reduces the amount of data transferred in the network, as only the needed parts are sent. However, this can be a problem as not all file formats are easy to split, for example the video and audio streams may not be contiguously mixed with each other or the file would not be possible to read without the information of the header at the start of the file.
- Send the whole file. This will increase the network traffic, but it will allow for a straighter forward processing in the nodes as they have access to the whole input file.
- Decode the file to a raw format where splitting is not a problem. This, however, will generate even more traffic than sending the whole file besides the overhead of decoding the whole file.

Attempting to have a node decoding to the network was found to be a inefficient option as that node becomes the bottleneck, besides increasing the total network traffic by 30 times. The best choice to keep network communication as low as possible would be to only send only the needed parts. To determine the needed parts some additional preprocessing of the file is required to determine where the split points are. This extra processing is either done by a specialized application or it can also be done by using already available tools like transcode, ffmpeg or mencoder using time stamps. This improves network efficiency as less data is transferred over the network but can reduce performance if the splitting introduces too much work for the sending node to handle.
Having a network file system, in this case the Andrew file system, which will provide concurrent access to all the nodes, will allow every node to have access to the whole file, making it possible for the program to seek within the file. The ability to seek within the file allows reducing the amount of data transferred and allows programs like transcode and ffmpeg to work normally. Using pipes or FIFOs sometimes raises problems as these operating system objects do not allow seeking within them.

However, after many experiments it became clear that the network file system was a performance bottleneck, because network I/O is slower than disk I/O. This affects not only the encoding and decoding by themselves, but the final merging, which being a serial task doesn’t get data fast enough as the network file systems tries to balance and cache I/O operations, which for merging implies a significant performance loss.

Summing up the AFS is an excellent method to distribute files around but it does have a performance impact on I/O. One should try to avoid doing I/O on the network file system. To do this it is possible to use sockets to try and connect all the nodes between themselves reducing the need for write operations on the network file system.

3.8 A parallel transcoder

In the previous section the architecture of a transcoder was presented, but what are the changes that need to be done to the architecture in order to have a parallel transcoder that uses a job scheduling system?

A parallel transcoder has the following phases:

- Identify the video and audio format of the media it is being presented with and verify that it can convert the input format to the to the output format.
- Process the video to determine how and where to split it
- Submit the job to a scheduler which involves choosing an appropriate number of nodes.
- Process the video in each of the nodes
- Gather the video fragments from the various nodes and merge them together.

Comparing these phases to those of a normal transcoder one can see that the key differences are the existence of splitting, merging and the support for a job scheduling system. In fact splitting and
merging can be done with some of the tools that are used for transcoding, leaving the support for a scheduling system as the only new functionality that does not exist in the already mentioned tools.

3.9 Existing parallel transcoders

3.9.1 DVD::RIP

This open source application is based on transcode. It is designed to convert DVDs to MPEG-4, by making use of some of transcode features, like frame accurate splitting and 2 passes encoding. It requires the existence of a shared network file system.

Figure 3.5 – DVD::RIP components
3.9.2 VisualHub

VisualHub is a video conversion tool for Macintosh. It can make use of the Xgrid framework that takes advantage of already existing Xserver features to distribute the work among the nodes. It is also based on the existing of a common network file system.

3.9.3 Summing up

Existing parallel transcoding applications make use of network file systems to distribute the work among the nodes and merge it back together. They have a central server that controls the execution of the work. While DVD::RIP wraps transcode for the user VisualHub is an application that uses the codecs directly.

Returning to the question of whether it is better to use a wrapper for existing tools or to develop a new tool and integrate with the existing media libraries. From the tools we can learn that using the media libraries will give more control over the processing of data, while using already existing tools allows have a shorter development time with more features. It is essentially a decision to either use a commercial off-the-shelf (35) strategy or not.

3.10 Summary

In this chapter, we have described the common elements of the architecture of some transcoders. All transcoders must have a decode, transform and encode workflow, because that is what transcoding is about. To do this they use a module or plug-in architecture to allow developers to easily add new formats to the system. When trying to implement a distributed transcoder there are some issues that introduce parallel overhead that need to be taken into account and mitigated. The first thing to consider is what makes sense to be executed in parallel and what does it mean for the whole transcoding process. Another thing is what needs to be transferred between nodes in the transcoding process. To answer these questions it is best to rely on experimental results that will be presented in the next chapters. However, it is forecast that to improve performance the disk I/O has to be kept to a minimum as well as the data transferred to the encoding nodes. Trying to avoid using the file system will increase performance but will also make the use of existing tools harder as they were not designed to work with pipes and sockets.

In the next chapter a transcoding architecture using already existing transcoding tools will be presented.
4 Implementation

The previous two chapters were intended to provide the reader with a basic understanding of how distributed video transcoding works and what are the performance tradeoffs. This chapter is about how the transcoder was implemented and how it works.

The implementation of the architecture gathered enough empirical experience to determine what the most efficient way of implementing a system for distributed video transcoding is and what are the problems commonly found when implementing such a system. This application was written in PERL and therefore requires the PERL interpreter installed. It also requires that at least ffmpeg or transcode is installed in the machine as they are needed for encoding and merging of the phases. It will also make use of mplayer as well as mencoder if they are installed in the system.

4.1 Chosen Architecture

There are two options to choose from. The first option is to develop a program that implements the actual video and audio transcoding process like mencoder and ffmpeg, the second option is to develop a program that will wrap something that already implements the phases of processing video. Implementing the actual video and audio transcoding process give more control over the process of transcoding, however tools like mencoder and ffmpeg have had years of development and sizeable developments teams to be able to support so many formats and features. That is why an architecture where one wraps already existing and time-proven tools is much more time efficient to develop than trying to create a new program from scratch.

The proposed solution follows the generic principles of the transcode and DVD::RIP transcoder, this mean it will try to adapt existing tools through the use of modules to be able to transcode video. This will make it possible to connect different tools together to take advantage of their own unique features.

Any application that tries to encoding video using multiple distributed nodes has to have the following functional blocks:

- Video Segmentation, which makes it possible to assign each encoding node to a part of the video.
- Video Decoding, which allows for the extraction of video frames from the input source.
- Video Transformation, which allows the extracted frames to be transformed.
- Video Encoding, which allows for the transformed frames to be encoded into the intended format.
- Video Multiplexing, which allow joining video and audio streams together if audio and video signals were split.
- Video Merging, which allows joining the output of the various encoding nodes in a single file.

These functional blocks when using the developed application will be implemented through a command generated by the command generation component. In the worst case scenario a different command would have to be generated to implement each of the functional block. Fortunately this is a very rare occurrence as for most uses it is possible to implement most of the functional blocks in a single command, because the video encoding tools have been in development for many years and are very mature, thus provide a great number of options which can be used to tailor the commands to implement the needed functional block.

![Diagram](image)

**Figure 4.1 – High level architecture**

This architecture uses various modules that adapt tools so that the application is able to use them. To adapt the tools for use with the application the modules have to provide one or more adaptors. The format adapter is needed because each tool can refer to the same format using different names. This means that there is a need to convert these tool specific designations to a global designation thereby normalizing the format names, so that it is possible to interact with all tools in the same way. The probe adaptor allows the application to use the tool to retrieve information about the input file. This is used to determine the input format as well as which tool is best suited to handle the conversion. The tool adaptor is the main task of the modules because it gives the application the ability to generate commands that can process the input file respecting a set of specified parameters.
The command generation starts by parsing the user supplied options and completing them using a set of defaults values. Once all is determined to be valid the input files are probed by all the modules that have probes registered to determine what the input format is and also which tool supports the input file. Now that the input and the output formats are determined the application has to determine which of the tools that is able to support the input format is also able to support the output format. To determine this each module adds mappings between the tools specific formats names and global format names. Usually more than one tool support both the input and the output format. The application uses a set of simple rules to narrow down the choices to a single tool. This tool selection is relatively straight forward, but sometime there are cases where it is not possible to arrive from the input format to the output format with a single tool and several tools are required. In this case it is necessary to link tools between themselves. This is node by using FIFOs. The end of the command generation results in a list of commands that need to be executed.

The command execution component is passed the command list previously generated by the command generation component. This list of commands is then executed according to the mode in which the application is running. This means it can execute the commands either by using the condor submission system or it can simply use the local command execution. Also, depending on the mode that is being used it may be needed to setup network connection or do some previous work or work in the end of the encoding process. These 3 different phases, the setup, main and final stages allow to have different groups of commands run together at different stages. This is useful as it assures that all commands from one stage have finished before starting the commands from the next stage. For example the merge of the video pieces back together is usually done in the final stage while the splitting can be assigned to the setup phase. It is important to only start the merging when all nodes have ended and to start the encoding only once all the files are split, because failing to the guarantee these conditions will make the next commands fail.

4.1.1 Wrapper based architecture

The components described in the previous section implement an architecture in which the functional blocks of a transcoder are implemented through the tool wrappers. These functional blocks are the decoding, encoding and merging of video and can be implemented using any of the tools like ffmpeg or transcode. This also allows for the implementation of multiple blocks with a single command produced by the command generation component.
Figure 4.2 – Program Flow

This diagram represents the flow between the different components. The actual implementation of the functional components in a distributed transcoder is delegated to the implementation provided by the modules, because tasks as segmenting the video, decoding and encoding it are done by the commands the module generates. The next section will go into more detail on how each of the different modes of operations work and how the commands that are generated in the master node are applied to the slave nodes.
4.1.2 Modes of operation

In this project several modes of operation were implemented to solve encountered problems. They also give an added flexibility to deal with the different setup of hardware that can be found in clusters.

4.1.2.1 AFS mode

The AFS mode was the first mode to be developed. It runs commands on multiple nodes making use of the file system that is common to all slave nodes to not have to distribute the file to all nodes. All slave nodes read the same file and skip large portions of the input files until they get to an assigned time position and start reading and encoding there. When all the slave nodes have finished encoding the master send a final command to condor to merge the files back together. This mode although it is called AFS actually only needs a common file system to all slave nodes, which means it can make use of AFS as well as any storage area network solution or other dedicated hardware based solution that provides a common file system to all encoding nodes.

Figure 4.3 – AFS mode
4.1.2.2 NET mode

The NET mode was developed to solve the inefficient merge of the files in the Andrews network file system, improve the network efficiency of transferring files by reducing the amount transferred across the nodes and also to improve encoding efficiency by having a local cache of the video segment and pipe output instead of writing it to disk.

The main innovation of this mode is the attempt to reduce the use of local disk to store temporary data. Initially this mode was not intended to use any disk access at all on the encoding nodes, however due to the need of the ability to seek in the input files that the tools require, a local cache in the local temporary directory was created.

Figure 4.4 – NET mode

In this mode the master setups up a file distributing server and a socket to retrieve the completely merged video from the merging node. Both the master and the merging node execute UDP broadcasts at regular intervals so that the slaves are able to find them. Each encoding node when started in NET mode will look for both master and merging node. Once an encoding node locates the master it tries to retrieve the video segment it is responsible for encoding by sending to the master its own node identification number. The master then splits the video and sends to the node...
only the video segment it needs to encode. The encoding node caches the segment that it retrieved from the master in the local temporary directory and when encoding pipes the output directly to the merging node. The merging node also locates the master when it starts through the UDP broadcasting and sets up a file server to receive the video segments from the encoding nodes. As different encoding nodes connect to the merging node they have to send their node identification number so that the merging node stores the segments in the proper order for the merge. Once all the encoding nodes have finished encoding and delivering the data to the merging node it will merge all the segments together and pipe the output of this operation across the network to the master node.

### 4.1.2.3 MIXED mode

The MIXED mode was developed because the splitting of all the segments in the master can become a bottleneck if the master is not powerful enough. In this mode both previous modes were merged into a single mode. The idea behind this is to take advantage of a common file system but still to be able to have the merge occur in a dedicated node using the existing technique developed for the NET mode.

![Diagram of MIXED mode](image)

**Figure 4.5 – MIXED mode**

In this mode the file is read by all the encoding nodes from the common file system and they pipe the output of the encoding to a dedicated merging node that caches all these outputs in the local
temporary directory and once all video fragments have arrived it merges all of them together and pipes the output across the network to the master node. Just like in the NET mode both master and merging node are broadcasting a UDP message so that the encoding node are able to find the merging node and establish a connection as well as the merging node which also needs to find the master node to send the output back to him.

4.1.3 Disadvantages

The disadvantages of using the wrapper based architecture are:

- Will have less features if the number of tools available are limited
- Will depend on whatever is installed on the system. Although that cannot be avoided as without libraries nothing works.
- Error handling and detection is much harder
- Some of the tools are not made to work together and therefore present a integration challenge.
- Will not achieve the same potential performance a dedicated solution could achieve, because it cannot be fine tuned in the same way.

4.1.3.1 Advantages

On the other hand the advantages are:

- More features and formats are supported as mature tools are used, however the tools all have different ways of being used and the modules will not implement all the features and formats of the actual tool.
- Will remain up to date, because most tools will remain with the same way of being interfaced with for a long time. This will allow new formats to be supported without need to rewrite the application.
- Will provide a fast development of a reliable tool as it doesn't have to implement error prone algorithms
- Will not be bound to a single tool as it will be able to use whatever is installed on the system. As well as providing some fallback as the same operation can be done in any of the tools installed.
4.1.4 Evolution

The architecture of the solution has evolved into various stages as some of the problems became more visible.

The first approach was to use a simple script written in bash to be able to produce a condor submission file that queue a couple of commands to be executed in parallel. This, however, was getting complicated to maintain as the program got more complex. So a migration to PERL became necessary.

With PERL it became easier to implement the logic, but the application was still very monolithically build. One script did all the work and as it became larger it became difficult to organize. So a division of the code into PERL modules was necessary.

With a few modules the code became more organized. But soon it came necessary to add more and more options to what the slave nodes needed to do. It was necessary to add a slave script to be executed in the nodes instead of the simple commands. Also, the condor module needed more reliability and logs needed to be added to automate the gathering of results.

More and more modules where added to wrap tools and to implement various other features. After extensive testing the results showed that the merging was still not good enough due to the AFS being a bottleneck. So it became necessary to try and remove that bottleneck from the system. The best approach would be to pipe data between slave nodes. However, this proved to be a problem because the merge will need to seek the input files and it is not possible to seek in a FIFO. To solve this problem it was necessary to introduce a cache in the local temporary directory in the merging node so that all output from encoding nodes is cached in that directory for the merge. This proved to be the most efficient way of transcoding in the available clusters.

The last mode of operation uses all the benefits of having access to the input files from all the nodes though the AFS. This allows it to read from the AFS the file and pipe the output to a merging node that stores its output In the local temporary files waiting for all nodes to finish to run the merging. Because it does not segment the input to the nodes it is not the most efficient.
4.2 How does it work?

In this section we will go through the steps of transcoding a video. This is what will be done by the application.

4.2.1 Load modules

The application starts by loading its own modules, this checks which are the available tools on the system as well as provides a self integrity check. Modules check for the tool they wrap on load, making the only available tools visible to the application the ones that are present. They also provide validation for user choices as well as some default values.

4.2.2 Identifying the media

The first thing to do is to identify the media that we are working with and apply the right preprocessing steps. For example a DVD has first to be decoded so that we can freely access its contents. Once we have the free access to the media content we have to identify its video and audio encoding, so that we use the right codec to decode it to a workable format. Most files can be identified through reading the first bytes of the header as they usually contain a binary identification sequence, also known as magic. The transcoder will use the available tools ffmpeg, transcode and mplayer to determine what is the format of the input. Another advantage of being able to identify it directly with the tools is that it also gives an idea of what are the tools that supported the decoding and demultiplexing of that format.

4.2.3 Tool Selection

For each file that is provided the application tries to find the best set of tools that can convert from the input format to the output format. For most standard formats this can be done with only one tool, however there are formats that are only supported by one of the tools, or there are formats that can be only decode or encoded by one tool, or there can be a need to apply a filter that only one tool has. So in this stage it finds the chain of tools that have to be used to accomplish the task.

4.2.4 Amount of nodes

Next it tries to detect if the condor scheduling system is present, if so it will try to define the best amount of nodes. For example in a heavily used system there might be no nodes available. In that
case it makes sense to use only one node as the queuing time for the parallel job would take too long. The system tries do assign 200MB per job, if there are not enough nodes available it will use the available ones, trying not be put into queue.

4.2.5 Setup

The system will try to setup what is needed to run the job. Depending on what the user chose like a frame accurate splitting it can have to preprocess the video, or have to split the video up in smaller pieces. This part will be submitted to the condor scheduling system so that it makes use of the parallelism, if the task can be split. Other setup tasks are setting up network connections, setting up temporary work directories and FIFOs in the local temporary directory.

4.2.6 Main

The system will then run the encoding process in parallel if condor is available or with multiple processes in the same machine if the condor is not present or the user so chooses.

The main execution phase is different for each of the available modes. For the local mode it just tries to sequentially go through the command list sending the execution of some commands to the background if more than one command is needed. For the AFS mode all modes work on a temporary directory created on the AFS. For the NET mode the master node splits the file and sends only the relevant segment to be encoded in the slave node which pipes the output across the network to the merging node. The MIXED mode allows each of the nodes to read from the AFS and pipes the output to the merging node.

4.2.7 Final

The system will run the merging process, which is run in one machine, as it is a very I/O intensive but has no load on the CPU. This is also placed on the condor scheduling system as it is bad practice to use the master node for any work at all.

4.3 Core components

The core components for this application are the command generation and the command execution.
There was also the need to support more complex operations as creating network connections in the slave nodes. To be able to handle these tasks it was necessary to have the nodes run a program instead of just executing commands sent to the condor system.

4.3.1 Command Generation

The generation of commands has to take into account what the results from the different probes were and what the desired output is. It has to create a sequence of commands that can convert from the input to the output. This in most cases can be done with few commands as the tools being used support a wide range of input and output formats. It has also to take into account the mode that is being used and the amount of nodes to use. The system has to assume a set of default values for things the user has not specified.
To generate the command it has to determine which of the tools are capable of processing the input. To do this the probes of the different tools are used. It then need to determine if both video and audio components are presents as well as whether filters are requested and which tools implement these filters. After that it needs to determine which of the tools are able to encode into the desired audio and video formats. All these give the system several different paths that could be followed. The application then has to choose the best path to follow from the various options. The general guidelines for optimizing this choice is to see which of the tools can do most of the decode, filter and encode stages. If there are more than one that can do all the stages the first option is chosen.

Once the tool is determined its module is asked to generate commands. This generation will output a list of commands for each node.

One advantage of the system is that multiple tools implement the same functionality, so there is some functionality redundancy. This allows for some fallback in case some tool is not present or is not capable of processing the input.

4.3.2 Command Execution

The command execution has to determine whether the condor scheduling system is available and if it should be used based on input parameters. If it is to be used a condor submission file has to be generated with various calls to a slave node script. This script has to be passed a node identification number as well as other arguments like the condor temporary work directory created by the master.

The master has to monitor the execution of the condor task. It features some failure detection capabilities. On some versions of condor the jobs seemed to have on some random occasions a tendency to stay in the idle queue indefinitely. To counter this, the master will try to reschedule the condor task if the nodes do not start entering the running state within a set amount of time. This is the available failure detection which was included in the system, because a more advanced failure detection is not feasible as it is not possible to determine if a certain video finished in a correct manner because even if the return value of a system command shows an error it can still have encoded a file properly. This happens when there is a sync problem when the audio and video tracks seem not to be of the same length and the multiplexer runs out of either video or audio. Also, analyzing the output of the tools does not allow differentiating between an error that has a serious
impact on the encoding process and one that does not. Even if a failure is detected there is not much one can do to fix the problem so this is left to the user.

The master also collects statistical data on the execution of the various nodes in the end.

4.3.3 Slave Node

The slave node allows the execution of complex tasks that would be hard to implement as system commands. It also allows the system to have improved logs. As each node reports how much time each task took.

It starts by accessing the command list for the job and determining what are the commands which are to be executed in the current node. To do this each node is passed a node identification number that is present in the beginning of each command line of the orders file which is in the working directory. It is also responsible for the creation of temporary directories and FIFOs for the subsequent commands to use. Its behavior is then determined by what mode is being used.

The nodes can be divided into encoding nodes and merging nodes. The encoding nodes in AFS mode execute the commands that are meant for them. In the MIXED mode the nodes read the input from the AFS and then send it across the network to a merging node. This requires the creation of a work directory in the local temporary directory as it is not possible to create FIFOs in an AFS. To send data across the network a sending thread to read from the FIFO and write to the remote merging node needs to be created. To find the merging node the encoding node has to listen for UDP Broadcasts. After determining where the merging node is, it connects to that node and sends its node id so the merging node knows which of the nodes is connecting to it, so it is able to maintain the proper video fragment order. In the NET mode the above is still true, but the nodes do not read from the AFS instead they read from the master directly and cache video fragments locally before executing the encoding commands.
The merging nodes are only used if a more complex mode is selected, like MIXED or NET as it will have to setup a merging node. When this is not the case no merging node exists. The merging node has to setup a server socket that will allow the different encoding nodes to send their results for local caching before the merging commands are executed. Another problem that the merging node has to solve is how the other nodes know where the merging node is. To solve this problem the merging node needs to send a UDP Broadcast to the network announcing its presence. It also needs to connect to the master server to deliver the output of the merging which is send from the command to the network through the use of a FIFO. The master is also found through listening to UDP broadcasts.

Each of the different tasks in the slave nodes is executed using threads to make it all run at the same time.

The idea behind the extensibility of this system is the use of the PERL modules to make more tools available as transcoding applications for the application. A few features are available to help the integration of new tools with the system by allowing the registration of global to local format mappings and tools to handle them, but in general the modules are free formed and the slave execute the commands they generate without evaluating them.
4.4 Summary

This chapter tried to give an overview of how the transcoding application was implemented and what and why the current architecture was chosen.

The application tries to follow the example of DVD::RIP and wrap different tools, so that it can perform the task in a intelligent manner. It also tries to make smart decision on the use of the condor scheduling system, so as to reduce the queuing time and not waste resources by submitting jobs with pieces that are too small.

The application evolved from a simple script to a far more complex system. However, the core elements of the implementation stayed the same. For the system to wrap around existing programs it is necessary to generate adequate commands that need to take into account the parameters supplied by the user and those that were defined as default for the program. These commands then have to be executed according to the mode the application is using. The modes available are the local, AFS, net and mixed execution. The execution of the commands in the slave nodes is done by a small program because it allows more complex task like setting up network connections to be controlled.

In the next chapter the experimental results of all the modes of the application will be presented, evaluated and compared to other results.
5 Results

In this chapter, an experimental evaluation of the application will be presented. The tests were made in both INESC grid and L2F grid, which have different hardware and utilization profiles. This allows for a better understanding of what the factors that impact performance are and how the application behaves in different scenarios.

5.1 INESC grid

The INESC grid is a cluster of 15 Pentium IV single core computers with AFS as the network file system. Network connection between the machines is a gigabyte Ethernet link that is shared between all machines.

5.1.1 Benchmarks

To have a better understanding of the performance of the cluster and establish a baseline a series of tests were done. These tests measure the network speed of the AFS as well as the encoding speed of a single node.

5.1.1.1 AFS speed test

These tests measure the read and write speed of the AFS which is used by all the nodes and therefore a critical component to the performance of the system as a whole.

The system utility dd was used for the measurements. To measure the write speed, the utility read from /dev/zero and wrote to a file in the AFS, then it read that file and wrote it to /dev/null and finally it simulated cached read and write as it copied the same file to another file in the same directory. The file size was 100 MB to be within cache limits.

The actual commands were:

```
# dd if=/dev/zero of=1gb bs=1M count=100 (Write test)

# dd if=1gb of=/dev/null bs=1M count=100 (Read test)
```
The results for these tests are interesting because they show that the performance of the AFS degrades as more and more nodes are added to the job. One has to take into account that these tests used condor to have multiple nodes access the AFS in a concurrent way, therefore as more and more nodes are added it becomes increasingly more difficult for condor to start all the jobs at the same time and the results begin to vary, because some nodes start either later or sooner than other and therefore get more bandwidth for themselves for a short time. These results are the average of all results reported by the different nodes in various iterations. The table of the above graph:

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Write (MB/s)</th>
<th>Read (MB/s)</th>
<th>R/W cached (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23,25</td>
<td>11,65</td>
<td>23,85</td>
</tr>
<tr>
<td>2</td>
<td>14,5</td>
<td>9,93</td>
<td>18,4</td>
</tr>
<tr>
<td>3</td>
<td>11,67</td>
<td>5,33</td>
<td>13,72</td>
</tr>
<tr>
<td>4</td>
<td>8,49</td>
<td>4,21</td>
<td>8,43</td>
</tr>
<tr>
<td>5</td>
<td>6,85</td>
<td>3,23</td>
<td>9,76</td>
</tr>
<tr>
<td>6</td>
<td>5,83</td>
<td>2,78</td>
<td>8,09</td>
</tr>
<tr>
<td>7</td>
<td>5,45</td>
<td>2,8</td>
<td>6,68</td>
</tr>
<tr>
<td>8</td>
<td>3,89</td>
<td>2,8</td>
<td>6,95</td>
</tr>
<tr>
<td>9</td>
<td>3,13</td>
<td>2,05</td>
<td>6,88</td>
</tr>
<tr>
<td>10</td>
<td>4,39</td>
<td>2,22</td>
<td>6,62</td>
</tr>
</tbody>
</table>
5.1.1.2 **Single Encoding**

To establish a baseline for the encoding speed the same input files that will be used in all subsequent tests were encoded multiple times using a single node. It was determined that the encoding of the 1 GB DVD quality video sequence takes about 11 minutes (362 seconds) on this system and the 7 GB high definition video sequence takes about 3 hours and 48 minutes (13730 seconds). The encoding of the high definition video had to be done one fragment at a time because the machine starts using the swap space which introduces a huge performance reduction when done all at once.

5.1.1.3 **Baseline Analysis**

In the presented baseline results one can see that the single encoding of the video takes 11 minutes to encode 1 GB of information and 228 minutes to encode 7 GB of high definition video. This means that it is processing 2.8 MB of information per second and 0.5 MB of information per second. For the I/O reduction from the use of multiple nodes not to have a negative impact on the encoding it would be advisable to use a maximum of 5 nodes to make sure that there is enough capacity to supply the working nodes with information.

5.1.2 **1GB Encoding Tests**

The input for these tests was the same 1GB DVD VOB with the typical DVD quality and a resolution of 720x576 used in the previous tests.

5.1.2.1 **No scale**

Below, the transcoding using ffmpeg as a tool of a MPEG-2 sequence to MPEG-4 keeping the same resolution. The graph below shows how the number of nodes changes the encoding time and the graph below that one shows the corresponding speedup.
The results below were obtained using the same encoding parameters but using transcode as a tool.
Figure 5.3 – GRID results using AFS mode with transcode

One thing to notice is the improvement in overall performance from transcode in this case. The reason for this is the faster merging of the transcode generated content, which makes some sense as the merging tool being used, avimerge, is part of the transcode package.

Below, the results from doing it using the networked mode with ffmpeg which tried to go around the AFS restrictions by splitting the file and sending only the needed parts across the network.
In this we can see the bottleneck introduced by having one single node splitting and sending the needed file parts to multiple nodes, however there is also an improvement in overall performance when comparing the total time between this approach and the others, as this one is almost twice as fast, because the merging cached the encoded parts of the file, so that it runs in the local disk and not in the network file system.

Figure 5.4 – GRID results using NET mode
Comparing this with the performance from the MIXED mode below, which also caches the merging of the files in a single node, but reads directly from the AFS we can see that bypassing the file system in this case does improve performance significantly.

One can conclude that the best option for encoding the video is the NET mode that bypassed the AFS and is able to send just what is needed by each of the encoding nodes.
5.1.2.2 Splitting Efficiency

Given that splitting and sending the file gave a good performance boost it is interesting to understand if that would also apply to the encoding if the splitting used the AFS to store the fragments.

Below, the transcoding using ffmpeg as a tool of a MPEG-2 sequence to MPEG-4 keeping the same resolution but splitting the VOB in the setup first and processing from the /tmp directory. The graph below shows how the number of nodes changes the encoding time. The graph below that one shows the corresponding speedup.
When splitting the video beforehand, the performance of the encoding grows almost linearly, but the merging is seriously impacted by it, resulting in a worse overall performance. Not accounted here is also the time it takes to split a video file into several smaller parts. One can see that the encoding in itself scales very well, because it presents very good speed up results in the encoding, however the merging operation which cannot be parallelized seriously hampers the performance of the parallel system. In this case the performance penalty introduced by the merging cancels all the benefits of the parallelization.

**5.1.2.3 Adding complexity**

To understand how adding complexity to the encoding affects the processing of video an additional operation was introduced to the encoding which was presented above. Below, the encoding using ffmpeg as a tool of a MPEG-2 sequence to MPEG-4 but doing a conversion to 240x192 which is a 3x pulldown. The graph below shows how the number of nodes changes the encoding time and the graph below that one shows the corresponding speedup.
This encoding test shows that the time depends on the amount of data generated. Although a scaling operation was added to the encoding increasing the complexity of the encoding, one can see that the overall performance increased. The main reason for this is that the merging is the performance bottleneck and as scaling down the video reduces the amount of data to be merge it allows the system to process the video faster.
5.1.3  7 GB Tests

The input for all these tests was the same 7GB Matroska with H.264 high definition video with the resolution of 1920x816 with a length of nearly 2 hours.

Below, the transcoding using ffmpeg as a tool of the Matroska sequence to MPEG-4 keeping the same resolution. The graph below shows how the number of nodes changes the encoding time. The graph below that one shows the corresponding speedup.
Figure 5.8 – GRID results with AFS mode

While the above graph show an improvement compared to the single node the below measurements of performance of the net mode which bypassed the AFS and splits the video in smaller parts shows even better results.
Figure 5.9 – GRID results with NET mode

The mixed mode in the figure below shows that bypassing the AFS is the best solution, because it also gains performance from having less interaction with the AFS.
Figure 5.10 – GRID results with MIXED mode

Although again the mixed mode does not present results as good as the net mode it has a slight improvement over the AFS mode.

5.1.4 Summing up

The various modes show that the time needed to process the video is shortened significantly by the use of additional nodes. However, the speedup shows that the gain for adding more and more nodes is become less for each additional node. The only mode that shows good speedups is the NET mode, which tried to bypass the AFS. It is important to note that the file servers are the main bottleneck, as AFS slows down the merge significantly. Another adverse effect of AFS is the slow I/O on input and output files. Comparing the performance of the smaller and the larger video file one can observe that the performance bottleneck that was seen using more than 5 nodes has disappeared. One reason for this is that when using smaller files the pieces to send are small enough for the sending of file between the nodes to be influenced by other activity on the cluster while when using the larger files the longer sending times dilute the influence of outside interference.

Other conclusions one can draw from these results is that the overall performance bottleneck is the merging of the various encoded file pieces. By looking at the comparison of the amount of time it takes to do the merge on the AFS and the encoding time of the net mode on can see that the merge operation takes about the same time as the whole encoding of the net mode. It is therefore
safe to conclude that the AFS cannot adequately provide enough I/O to merge the files in an efficient way.

Bypassing the AFS by piping the information between the nodes proved to give the best results in all test cases in this cluster. This happens because the AFS does not provide enough I/O for all nodes to work at full potential. Also, the splitting of data to reduce the amount of data sent to each node proved to be a good choice as it reduced the amount of data transferred and therefore the amount of time needed to transfer data before the nodes can start encoding.

5.2 L2F Grid

The L2F grid is a cluster of 24 quad core computers with AFS as the network files system. Comparing to the previous system it is a far more heterogeneous grid with 10 times as many users. It also has machines that can join and leave the grid, depending on whether they are currently being used by the grid or not.

5.2.1 Benchmarks

To have a better understanding of the performance of the cluster there are a couple of standard tests that can be done. This also allows comparing the two clusters between them more accurately.

5.2.1.1 AFS speed test

As already stated for the previous cluster, these tests measure the read and write speed of the AFS which is used by all the nodes and therefore a critical component to the performance of the system as a whole.

The system utility dd was used for the measurements. To measure the write speed the utility read from /dev/zero and wrote to a file in the AFS, then it read that file and wrote it to /dev/null and finally it simulated cached read and write as it copied the same file to another file in the same directory.

The actual commands were:

```
# dd if=/dev/zero of=1gb bs=1M count=100 (Write Test)
```
These results show that, as in the other cluster, the I/O performance of the AFS also degrades as more and more nodes are added to the job.

### 5.2.1.2 Single Encoding

In this cluster the encoding is faster than in the previous one as the computer are more recent and therefore have faster processors. The same files used for the tests in the GRID cluster are also used here. It took 7 minutes (471 seconds) to encode the 1 GB DVD quality video sequence and 3 hours (10830 seconds) to encode the 7 GB high definition video.

### 5.2.1.3 Baseline analysis

In this cluster in spite of the better processors and network it still takes quite some time to process the high definition video. Also, one can see a drop in network capacity when using more than 5 nodes in the performance of the network file system. This is expected as the network file system is also AFS (Andrew File System) like in the previous cluster. This cluster has more users that the
previous cluster, that means there are a greater number of external factors which can influence the measurements.

5.2.2 1GB Tests

The input for these tests was the same 1GB DVD VOB with the typical DVD quality resolution of 720x576 used in previous tests.

5.2.2.1 No scale

Below, the transcoding of a MPEG-2 sequence to MPEG-4 keeping the same resolution and using ffmpeg as a tool. The graph below shows how the number of nodes changes the encoding time and the graph below that one shows the corresponding speedup.
These experiments show that although the encoding time benefits from adding more multiple core nodes to process the job. There is no real gain from it because the merging operation completely removes all benefits of the parallelization. The merge is a serial operation that cannot be parallelized and is therefore always done using only one machine and taking a constant time. One root causes for the erratic nature of the measurements is that the amount of people using the system, and the I/O they consume, varies widely. In the tests above, the merging completely outweighs the benefits as soon as 2 nodes. This led to the development of the NET mode which uses a local cache to speed up the merging process. Although this mode does not work efficiently on the AFS it can bring substantial gains if the nodes of the cluster were connected through a faster file system.

Doing the same using transcode with frame accurate splitting also demonstrated poor performance:
Figure 5.13 – L2F results with AFS mode and transcode

Below, the results using the NET mode, which as expected showed the best results.
As the nodes are quad core the performance of putting the input files in the local temporary directory and using multiple processes running in parallel was also examined.
These results for the local encoding of the video using a node with a quad core processor only achieves a maximum speed up of 2 if considering the encoding by itself. And also here the merging reduces the speed up to a lower value. This last test was run entirely from the local temporary directory and it launched all the encoding processes in the same machine. At 5 processes in a quad core system the performance started to become worse, as all the processes are competing for CPU time to run. Still the actual encoding gained from using multiple cores, but the need to merge the
files back together reduced the overall performance. Another experiment that was done was the use of the thread parameter to control the amount of threads ffmpeg spawns when using the multiple core system. Comparing the performance of defining 4 threads or not defining anything shows that it stays the same, which means ffmpeg will use the adequate number of threads for the number of cores of the system.

5.2.3 7 GB Tests

The input for all these tests was the same 7GB Matroska with H.264 high definition video with a resolution of 1920x816 and a length of nearly 2 hours, which was used for the tests in the other cluster.

Below, the transcoding using ffmpeg as a tool of the Matroska sequence to MPEG-4 keeping the same resolution. The graph below shows how the number of nodes changes the encoding time. The graph below that one shows the corresponding speedup. These tests use the AFS to do all operations. The tests were stopped at 5 nodes because with this amount of nodes the merging operation was already taking up more time than the encoding itself. The NET mode will mitigate this with local caching.

```
\begin{figure}
  \centering
  \includegraphics[width=\textwidth]{figure.png}
  \caption{Graph showing encoding time changes with number of nodes.}
  \label{fig:encoding_time}
\end{figure}
```
Using the NET mode, which tries to avoid using the AFS it is possible to achieve much better results. This is shown in the graphs below:

**Figure 5.16 – L2F results with AFS mode**
Using the MIXED mode the input file is read from the AFS and uses a merging node like in the NET mode. We see that the performance isn't as good as for the NET mode but it is still better than doing the merge with files that are in the AFS.
Figure 5.18 – L2F results with MIXED mode

At 5 nodes the performance seemed to get worse.

5.2.4 Summing up

The maximum measured encoding speed for the 7 GB high definition video sequence is 4 with 5 nodes in the network mode. This mode shows again that it is a good option for encoding the video because it doesn’t need to do operations in the AFS using the local temporary directory instead. All other modes seem to not scale beyond 5 nodes, this is because although the computers of this cluster are better than the previous one there are also more users using the cluster.

This cluster also demonstrates that the merging is the main performance bottleneck because the encoding always scales well. It is the merging, an operation that is not needed when the transcoding is running using only one node, which degrades the system performance so much.

Bypassing the AFS by piping the information between the nodes proved again to give the best results in all test cases in this cluster.
5.3 Evaluating the results

Comparing the results from both clusters one can see that both have the same problem and have similar behaviors when the number of nodes is increased. Their performance is seriously hit by the merging operation, because the AFS is an I/O bottleneck. When this performance bottleneck is bypassed the overall performance is improved in both systems. The performance bottleneck of the file system in the L2F cluster is higher because of the larger amount of users, who tie up more network and I/O resources.

On both systems the best method to encode video was using the net mode which uses mencoder to split the video and pipe it across the network to the slave nodes. These parts are cached locally and are processed and send again across the network to a merging node which caches them locally so that when all are completed the merge operation can be done using only the local disk. This proved to be the best way to reduce the merging impact on the encoding.

It is also useful to compare the results to other studies in the same field to be able to assert if there is something learned or not. Comparing to other studies (30) which also focus on the efficiency of distributed video transcoding one can see that the mode that works best shares the same architecture of having a source computer which splits the video into segments so that a series of computers can transcode them and a dedicated merging computer to assemble the pieces together. The attempted of using existing resources as the AFS proved to not provide fast enough I/O for the merging to be done efficiently, although the actual encoding show good encoding performance.

5.4 Critical Factors

Reviewing the results one can identify a number of critical factors that affect transcoding. These are I/O and CPU speed. CPU speed is needed for providing enough computational power to be able to do the complex mathematical operations needed to decode and encode video content and it is the main factor when doing a single node transcoding. However, once the process is distributed across several machines the main factor becomes I/O speed, because all the different nodes need to be supplied fast enough with information so that they can contribute with their full computational power. Supplying the nodes with data is a problem, because this can create a potential bottleneck when a single server has to supply a large amount of nodes. There are several solutions for this like data replication, to increase the number of possible data suppliers, as well as, increasing the chance that the data is local to the running application. However, existing solutions that split and replicate files
automatically are not ideal for video processing, because it would need to be tailored for the specific video format to really improve performance.

The main performance obstacle is the efficient merging of the various video files produced by the encoding nodes. In some cases, although the encoding speed up is able to grow linearly with the number of encoding node, the merging of the files is so inefficient that it nullifies the advantages of using several nodes. The reason for this inefficiency is the low I/O speed of the AFS on which it is running. To solve this, the merge operation has a special node assigned to it that caches the output of the encoding nodes and does the merge using the local temporary directory, thereby mitigating the impact of the merge, but not removing it. In fact the merge introduces such a big performance hit on the system when using the AFS, that the only way it would be possible to gain performance with several nodes using the AFS is to assign a different file for each node so that the merging would not have to be done. It was also determined that the impact of the merge is proportional to the size of the files being merged.

The solution for the poor performance of the AFS was to bypass it using network connections and the local disks. There are dedicated storage devices that allow for fast enough I/O on the files, so there is some usefulness to the modes that don't pipe information across the network themselves. For example a storage area network allows a large number of devices to share disk using fiber channel connections. This, although not having been tested with this application, does seem like it could make the AFS mode viable for distributed video encoding.

Another interesting question is if adding more network bandwidth would improve performance. While a faster network connection is always good, the encoding processes in the tests are not fully using the available network capacity in both clusters, because the file servers cannot process the write and read operations fast enough. This leads us to the conclusion that before upgrading the network it would be wise to upgrade the file serving nodes because they are a far greater limitation than the network. The I/O, being an important performance bottleneck, should be further minimized. To do this, converting local disk caches to memory buffers should improve even further the performance, but if the file has to be read and written from and to the current AFS, there will still be a performance bottleneck there. The main problem for this conversion it to make sure there is enough memory for large files like in the 7 GB high definition video case, when the number of nodes is small.
5.5 Summary

In this chapter, the results of various tests in two different clusters were presented. The results show that although the distributed encoding is efficient, the merging has such a performance impact that it almost removes all benefits of distributed encoding. To counter this, changes had to be made to the system to reduce the impact of the merging of segments, making the distributed encoding more efficient than single encoding.

One can see that the encoding time is always improved by the adding of additional nodes, but the merging of the output of each node introduces a new task which does not exist when doing single node encoding. This new I/O hungry task can introduce enough extra time to surpress the benefits of distributing the work. To mitigate the time of this new task it was necessary to move it to a dedicated node in which the output of the encoding nodes can be cached to disk, allowing it to use the I/O of the local disks to reduce the impact on the overall encoding time. The merging can be done on an AFS as long as the I/O is fast enough and this was not the case in both clusters, as the I/O is limited so that all users are able to work at the same time. A more dedicated storage solution might be able to provide the needed I/O speeds to make the merging a viable task to be done on the AFS.

In the next chapter the conclusions that can be drawn from this work will be presented.
6 Conclusion and future work

In the previous chapter, the results show how the performance is impacted by various changes to the way the transcoder works. In this chapter, the contributions of the developed work will be summarized and some of the open problems left for future work will be described.

This work has presented a distributed transcoding application that can cope with a number of different situations. Using already existing transcoding tools it was possible to efficiently transcode video using multiple nodes. By efficient transcoding it is understood that transcoding with multiple nodes is faster than transcoding in a single node. To achieve this it is essential to have a fast way of providing concurrent access to the input file, as well as being able to provide enough I/O to the node where the merging of the nodes occurs. The NET mode was the most efficient way to transcode video with multiple nodes, because it reduced I/O on the slower AFS mounted volumes using local caches. To further increase the performance even these disk I/O could be converted to memory I/O.

The main problem of transcoding video is the need to apply CPU intensive transformations to a large amount of compressed and possibly interdependent data. Existing tools where designed to operate on files that are on a file system and therefore assume it is always possible to seek on the files they operate with. This is especially problematic when trying to use pipes or FIFOs with these applications. A network file system allows concurrent access to files for all nodes. However, the I/O on the available clusters network file systems was slower than I/O on local disks. To mitigate this problem, sending the output across the network to a merging node that can cache it on local disk was shown to be a good solution. One also has to take into account that increasing the work done on each node will also increase the performance improvement that local caching bring.

The key for distributed video processing with these cluster setups is efficient splitting and merging of the video fragments, because based on the experimental results the encoding always showed to gain much from the extra computational power of added nodes. Although the addition of more nodes allows gaining a good speed up, the merge introduces a file size dependent constant time, which cannot be reduced through parallelization, as the merge is a serial operation that does not exist when using a single node.
6.1 Usage Scenarios

The usage scenarios for the developed application can range from single file encoding to batch encoding of multiple files. Also, the common need to convert from one format to multiple formats was also included. This covers the most common usage scenarios for transcoding applications, except the real-time encoding and streaming of video (36).

In the available clusters the most efficient way to encode video was to pipe information across the network bypassing the AFS. If that is not possible it would be more efficient to assign one file per node for encoding than to process each file using multiple nodes. Again this is a limitation of the available clusters and not a general consideration. This is, however, important to consider when implementing an encoding cluster using available tools and not having access to more expensive components as dedicated hardware for storage area network.

The merge was clearly an obstacle. However, there are scenarios where the merge is not needed. A very common application for video transcoding tools is to split a video into various pieces, an example of this is transcoding a news channel or any other large video source which is not intended to be delivered as a whole.

6.2 Future Work

One research problem that seems important for future work is the improvement of the merging in a distributed environment, which has to be done by reducing the I/O, as the merge is a purely I/O based operation. This operation should not slow down the transcoding nodes and take into account that there is no guarantee that the nodes will finish in the order that they were started. Also, the each encoding node is producing different length compressed data segments. To correctly merge these segments into a properly formatted container format it may be necessary to seek the file to build video indexes.

Another potential area of research is the customization of a map/reduce system for particular video formats, so that this system is applicable to video processing. For this to work a customized splitting of video would be necessary to split according to the structure of the video. It still would be necessary to customize the reduce function, not only to encode but also to merge the produced pieces. Some of the challenges that might be encountered are the need to have encoding nodes share information, especially in the case of split video segments that end on bi-directionally dependent frames.
7 Bibliography


4. —. **14496-5: Coding of audio-visual objects (MPEG-4).**


17. IGMP snooping - RFC 4541 - Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches. 2006.


