Distributed Video Content Analysis

Jonas Lidén

January 20, 2015
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Johan Tordsson
Examiner: Fredrik Georgsson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

Video Content Analysis (VCA) is usually computationally intense and time consuming. In this thesis the efficiency of VCA is increased by implementing a distributed VCA architecture. Automatic speech recognition is used as a case study to evaluate how the efficiency of VCA can be increased by distributing the workload across several machines. The system is to be run on standard desktop computers and need to support a variety of operating systems. The developed distributed system is compared to a serial system in use today. The results show increased performance, at the cost of a small increase in error rate. Two types of load balancing algorithms, static load balancing and dynamic load balancing, is evaluated in order to increase system throughput and it is concluded that the dynamic algorithm outperforms the static algorithm when running on a heterogeneous set of machines and that the differences are negligible when running on a homogeneous set of machines.
# Contents

1 Introduction ............................................. 1
   1.1 Partners ........................................... 2
   1.2 Problem Statement .................................. 2
   1.3 Purpose and Goals .................................. 3
   1.4 In-depth Study ...................................... 3

2 Methodology ............................................. 5
   2.1 Issues and Solutions ................................. 6

3 Load Balancing and Video Processing in Distributed Systems 9
   3.1 Preliminaries ........................................ 9
   3.2 Static Load Balancing ................................. 10
   3.3 Dynamic Load Balancing .............................. 11
   3.4 Adaptive Load Balancing ............................. 14
   3.5 Task Granularity ..................................... 14
   3.6 Video Data Components ............................... 14
   3.7 Video Compression ................................... 15
   3.8 Video Bit Stream Structure ......................... 16
   3.9 Audio Bit Stream Structure ......................... 18
   3.10 Conclusions ........................................ 18

4 Distributed Video Content Analysis Architecture .......... 21
   4.1 Introduction ........................................ 21
   4.2 Overall System Design ............................... 21
   4.3 Software and Libraries ............................... 25
   4.4 System Description ................................... 26

5 Performance Evaluation .................................. 33
   5.1 Setup .............................................. 33
   5.2 Speech Recognition Accuracy ......................... 34
   5.3 Speedup and Efficiency ............................... 35
   5.4 Crossover Point ..................................... 39
5.5 Throughput ........................................... 40

6 Conclusions .............................................. 45
   6.1 Restrictions ....................................... 45
   6.2 Limitations ....................................... 46
   6.3 Future Work ..................................... 46

7 Acknowledgements ........................................ 47

References .................................................. 49

A Installation and Configuration .......................... 51

B Additional Result Data .................................. 55
   B.1 Speech Recognition Accuracy ................. 55
   B.2 Speedup and Efficiency ....................... 61
   B.3 Crossover Point ................................. 62
   B.4 Throughput ...................................... 66

C List of Acronyms .......................................... 67
List of Figures

3.1 A distributed system without load balancing. Some of the servers are under heavy load while others are idle. ............................................. 10
3.2 A distributed system with load balancing. All servers are moderately loaded. ......................................................... 10
3.3 SLB algorithm classes. ...................................................... 10
3.4 Classes of DLB algorithms. ............................................... 12
3.5 The components of a DLB algorithm. ................................... 13
3.6 Interframe codings showing relationship between MPEG frame types. .............................................................. 16
3.7 A simplified view of the MPEG video bit stream. .................. 17
3.8 A simplified view of the MPEG audio bit stream. ..................... 19

4.1 The nodes in the distributed architecture are connected to each other in a unstructured peer-to-peer network. ....................................... 22
4.2 Illustration of the master-worker relationship. The master scatters the workload across the workers that does the analysis and returns the partial metadata result to the master that gathers and compiles the final metadata result. .......... 23
4.3 The logical architecture for the distributed application. The arrows show the data flow in the system and how the different modules interact. .............. 24
4.4 The Vidispine Transcoder HTTP XML/REST API. A HTTP client is able to post and queue jobs for execution, status of jobs can be retrieved in HTTP responses. Metadata generated by plug-ins are returned by a HTTP PUT request to a url defined in the job request, this requires a HTTP server instance that can receive the request. ...................................................... 25
4.5 Description of the Partitioner Module functionality (FFmpegPartitioner). ............................................................... 27
4.6 Description of the Task Monitoring Module functionality. ................................................................. 28
4.7 Description of the Load Balancing Module functionality. ......................... 29
4.8 Description of the Group Management Module functionality. ......................... 29
4.9 Description of the Work Handler Module functionality. ............................................................... 30
4.10 Description of the SpeechToTextAnalyser ................................. 30

5.1 WER and WAR for different partition sizes ........................................... 35
5.2 Speedup for executions with different node counts. .................... 37
5.3  Efficiency for executions with different node counts.                        37
5.4  Predicted speedup for an increased number of nodes.                       38
5.5  Average execution time for videos of increasing duration running with partition size 25, 35 and 45 seconds. 39
5.6  Throughput for the two load balancing algorithms.                        41
5.7  Load balance during throughput measurements on the homogeneous cluster. 42
5.8  Load balance during throughput measurements on the heterogeneous cluster. 42

B.1  Video 1 (AJJacobs_2014A-480p.mp4): WER and WAR for different partition sizes. 56
B.2  Video 2 (JorgeSoto_2014G-480p.mp4): WER and WAR for different partition sizes. 56
B.3  Video 3 (KareAnderson_2014S-480p.mp4): WER and WAR for different partition sizes. 57
B.4  Video 4 (SusanEtlinger_2014S-480p.mp4): WER and WAR for different partition sizes. 58
B.5  Wall clock time (in seconds) measurements for execution with different node counts. 61
B.6  Wall clock time (in seconds) measurements for executions of video partitions of increasing length running the current serial system. 63
B.7  Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 25. 63
B.8  Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 35. 64
B.9  Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 45. 65
B.10 Throughput measurements for heterogeneous and homogeneous clusters running the two different load balancing implementations. 66
List of Tables

5.1 Amazon AWS platform used for the performance evaluation. .................... 33
5.2 Test videos used in the performance evaluation. ................................. 34
5.3 Speedup results for executions with different node counts. ..................... 36
5.4 Calculated overhead compared to the current serial system. ........................ 40
A.1 Descriptions of the configuration options. ........................................ 54
B.1 Average WER and WAR for different partition sizes for all four videos. ........ 55
B.2 Video 1 (AJJacobs_2014A-480p.mp4): WER and WAR for different partition sizes. ......................................................... 57
B.3 Video 2 (JorgeSoto_2014G-480p.mp4): WER and WAR for different partition sizes. ......................................................... 58
B.4 Video 3 (KareAnderson_2014S-480p.mp4): WER and WAR for different partition sizes. ......................................................... 59
B.5 Video 4 (SusanEtlinger_2014S-480p.mp4): WER and WAR for different partition sizes. ......................................................... 60
B.6 Wall clock time (in seconds) measurements for execution with different node counts. ......................................................... 61
B.7 Average execution time for videos of increasing duration running with partition size 25, 35 and 45 seconds. ......................................................... 62
B.8 Wall clock time (in seconds) measurements for executions of video partitions of increasing length running the current serial system. ......................................................... 62
B.9 Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 25. ......................................................... 62
B.10 Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 35. ......................................................... 64
B.11 Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 45. ......................................................... 64
B.12 Throughput measurements for heterogeneous and homogeneous clusters running the two different load balancing implementations. ......................................................... 66
Chapter 1

Introduction

According to a study performed by the International Data Corporation (IDC) – a market analysis company – the digital universe will grow by a factor of 10, from 2013 to 2020. The amount of data that is collected and stored is more than doubled every year according to the study [30]. One of the challenges with this increasing amount of data is cataloging and organizing the data to simplify search and retrieval. Describing the contents of data files is done by storing metadata – data about the data – and increases usefulness and value of data. For example, a video may include metadata about participants, spoken language, subtitles, and tools used to create it etc. It is costly and time consuming to create metadata manually, therefore a lot of metadata is created automatically by data analysis software.

Media Asset Management (MAM) systems automate tasks such as ingesting, transcoding, cataloging, distributing, and retrieving media, typically video. MAM systems are commonly used by broadcasters and media producers to organize and monetize their digital assets. Modern MAM systems are expected to handle very large quantities of data. For example, the BBC digital archive alone contains material dating back to 1890, and offers 1 million hours of playable material [13]. To facilitate search and retrieval, descriptive metadata is often entered on ingestion. Due to the size of the data sets, there is shift from manual to automated annotation. This calls for algorithms such as face detection and recognition, speech recognition, logotype detection, quality assessment, music/speech discrimination.

As automated annotation cannot compete in terms of quality, it has to do so with efficiency. In the case of Automatic Speech Recognition (ASR), modern transcription software such as CMU Sphinx [27] run slightly faster than real time on a standard desktop computer. This is not sufficient as systems have difficulties keeping up with the pace with which new material is produced, let alone time to annotate existing archives. Part of the solution is a distributed analysis system, where the input video is partitioned and the work is divided over several machines. Archives such as BBCs are still out of reach, but it is enough for the needs of a medium sized production company.

There are systems today that perform automated annotation analysis. However these systems perform analysis on one media asset at the time. When dealing with media assets like video or audio with long durations it is beneficial to split the media asset in parts and process each part concurrently at different machines in order to increase the efficiency of the analysis.

In a typical office environment, machines are commonly idle, either because they are not used to their full capacity or that they are not in use at all. This unused capacity could be used to assist in Video Content Analysis (VCA). With this follows certain limitations.
Chapter 1. Introduction

Users must be able to use their computers even if they assist in the media analysis, the machines cannot be overloaded, so they become inoperable. This makes load balancing an important part of the architecture. Furthermore a number of different operating systems are in use and this sets requirements on a wide platform support.

CodeMill AB – a software consultancy firm located in Umeå – have partners in the media industry one of which is Vidispine AB. Vidispine develops a media asset management system that is used, among other things, for VCA to extract metadata from media assets kept in the system. CodeMill have developed a number of VCA plug-ins for Vidispine’s system, including ASR i.e. translating spoken words into text and Face Detection (FD) i.e. determination of locations and sizes of human faces. Now CodeMill want to examine the possibilities to develop a distributed system running on standard desktop computers that can assist in the VCA, this is of value for potential customers of CodeMill.

The focus of this thesis lies in creating a distributed architecture for VCA that uses the current tools provided by Vidispine and CodeMill. The distributed architecture is evaluated and compared with the current system based on performance metrics like speedup, throughput and accuracy. The idea is that the distributed architecture should be easy to extend to other types of VCA.

1.1 Partners

CodeMill AB is the main partner who laid the foundation for this thesis. They supply knowledge, information, workspace and tools that makes this thesis work feasible. CodeMill is an IT consultancy firm located in Umeå [1].

Vidispine AB is an IT-company that develops a MAM system called Vidispine [2]. Parts of the Vidispine platform is used in this thesis project. Vidispine is a partner of CodeMill.

1.2 Problem Statement

The primary task of this project is to design and implement a distributed VCA architecture. In order to focus on the distribution of the workload and not on performing the actual analysis, the VCA tools available from Vidispine and CodeMill is used. The system is to run on standard office workstations with a variety of operating systems (Linux, Mac OS X, and Windows) or on a dedicated Linux cluster. It is important that the machines are operable for the users to be able to carry out their normal tasks even if they participate in the distributed system. The machines can turn off and turn on at any time as users arrive or leave, this requires a robust system. Load balancing is an important part of the system as it contributes to even workloads and increased throughput. The research questions addressed in this thesis are:

1. What speedup, relative to the current system, can be expected using the distributed architecture?
2. How long video duration (problem size) must be analyzed for it to be beneficial to use the distributed architecture over the current system?
3. What method of load-balancing achieves the highest throughput for a given set of machines?
1.3 Purpose and Goals

The purpose of the project is to increase the efficiency of automated annotation. Existing systems cannot keep up with the rate at which new media material is produced and at the same time annotate existing archives. There is thus a need for a system that can annotate media assets with a higher throughput compared to the system that is being used today.

The goal is to design and implement an architecture for distributed VCA to be used for automated annotation. As a case study, the work is focused on increasing efficiency for ASR because it is deemed important by many of CodeMills customers.

1.4 In-depth Study

The in-depth study of this thesis reviews load balancing for distributed systems and tries to identify weaknesses and strengths of various load balancing algorithms with the intent of finding out what options that are available and how they affect the overall performance of the distributed VCA system.

Video and audio data structures is examined in order to get a deeper understanding of how workloads of video data can be distributed. Questions to answer include: How is a video bit-stream be partitioned without loosing important information needed to decode the video? How does this affect the task granularity? How does this in turn affect the load balancing?
Chapter 2

Methodology

The thesis project is conducted in three phases: a research phase, where existing literature is studied to gain an understanding of the problem and what has been done before; a development phase, in which the system is developed; and an evaluation phase where the system is evaluated and tested.

In the research phase, a literature review on load balancing for distributed systems is performed for the sake of increasing knowledge of the different load balancing types, their performance, and how input data partitioning choices affects load balancing. In the VCA case, the input data is video and audio data. Hence, a study of the literature on digital video is included to gain understanding of how video streams are stored and how this affects how video can be partitioned. How the video data is partitioned does not only affect load balancing, the results of the VCA may also be affected especially in the case of ASR. For example, if ASR is performed on video that is split in the middle of a word, the word is most likely erroneously classified.

Relevant research regarding load balancing and video compression is identified by searching for scientific publications and journals in research databases, in libraries, and on the Internet. Detailed examination of collected papers are performed, and exclusions are made if the content is irrelevant or if inconsistencies are found. The collected research is later summarized and documented in the thesis report. The process of searching, examination, and documentation is an iterative process that is performed until the literature review is complete. The results of the literature review is presented in Chapter 3. In addition to the literature review, the current VCA system and ASR system is examined during the research phase to get an understanding of how they are used and if they need modification to fit the distributed architecture.

When the research phase is completed, the development phase commences. The knowledge gathered in the research phase is used in order to design the distributed architecture. The goal is to produce a modular design to make it easy to replace parts and extend the architecture to support any kind of VCA, not only ASR even though the latter is the target of study in this thesis. The hope is that the modular design makes it easy to replace the load balancing component in order to test different load balancing techniques. The development process is performed in an iterative process where design, implementation and test make up the main parts in the cycle. The development phase is permeated by an agile approach to software development. The result of the development phase is presented in Chapter 4.

When the development phase is completed, the system is evaluated using quantitative measurement of its performance. Experiments are performed on the the implementation of
the distributed architecture. Tests are designed, performed and evaluated in order to answer the research questions stated in Chapter 1. The results are presented and conclusions are made about the performance differences of the new and the old system. Descriptions of the tests and results of the evaluation is presented in Chapter 5.

2.1 Issues and Solutions

The result of this project is an implemented prototype of a distributed VCA system. Distributed VCA is a parallel application of VCA, thus the analysis workload is split between several computing units. Several issues must be solved for the project to be a success. Descriptions of the identified issues and methods for solving those issues follows.

Video partitioning

The first step towards distributed processing is finding a way to split the input data such that computations can be performed in parallel. Video data is often stored in compressed form for transfer and space limitation reasons. In order to split compressed video data – without loosing any information in the process – a basic understanding of video compression is required. How the video data is partitioned may affect the VCA results if the analysis relies on information which exists between a set of frames. In the case of ASR this is true. Speech recognition is done on word or sentence basis, which means that the video cannot be split in the middle of a word or sentence without introducing errors.

To solve this issue, the theory of video compression is studied to understand how video can be partitioned without loosing any information or cause corruption to the data. Existing libraries for video processing is examined to find practical solutions for how video partitioning can be done once video compression is understood. An effort is made to split the video in between words or sentences to solve the issue related to ASR. Voice Activity Detection (VAD), i.e. a technique to detect presence or absence of speech is used to solve this problem. VAD is also a form of VCA and impact the system performance. However, simple approaches to VAD have been shown to be tractable for real-time applications [17]. Using this kind of approach makes the performance impact of VAD negligible.

Automatic discovery of nodes

The participating nodes may join and leave the system at any time, this requires a mechanism to handle discovery issues, i.e. finding available nodes. This requires a robust system that can handle failing nodes in the middle of running VCA.

This issue is solved by implementing group management and failsafe mechanisms into the system. Existing toolkits for distributed systems is used to speed up development as this can be cumbersome work and the time for this project is limited.

Share video data between nodes

Another issue is how to share the video data between nodes. Should the whole file be shared and analysis start from different timestamps? Should the video be split and only partitions be sent? Are there other possible solutions?

There are benefits and drawbacks with both approaches. Distributing the whole video data between all participating nodes avoids a single point of failure and is beneficial for system robustness once the data have been received by all nodes. For large files this requires a lot of bandwidth, which have an impact on system performance. Splitting and sending only the portion of video data that is required by the node analyzing it saves bandwidth, but imposes a single point of failure as the complete video data is
only available to one node. The latter approach is used in this project because the performance gains are valued higher than the robustness. It is possible to consider other options as well, e.g., using a centralized storage for the video to be analyzed. Centralized storage is a common solution for organizations involved in collecting, analyzing, and storing large amounts of video data and is also a viable solution. However, to mimic a video storage solution in development by CodeMill and Vidispine, where each node keeps a local storage, the shared data solution is picked.

**Collection and compilation of results**

When the analysis is complete, the results have to be collected and compiled into a final result. Annotations like subtitles from the results of ASR have time stamps. Video partitions start from different time stamps in relation to the original video, but seen as single units they all start from zero. This causes an issue with erroneous timestamps in the final result.

This issue requires a solution that keeps track of the offset in time where each partition starts. Post-processing of the collected results is used to correct erroneous timestamps and compile a result with fewer errors.

**Multi-platform**

The system is to be run on standard desktop computers that run Linux, Mac OS X, and Windows. This may limit the choice of programming languages, libraries, etc. that can be used in the development of the system.

To solve this issue, a multi-platform language is used. If this solution is infeasible, e.g., due to dependencies or limited time, this issue may be left as unresolved.

**Load balancing**

One of the motivations for developing a distributed VCA system is to decrease the real time taken to perform the analysis. If nodes are overloaded in a distributed system, chances are that the performance gains of the distributed architecture are lost.

Load balancing solves this issue by reducing congestion and averaging workload among nodes. There are many types of load balancing algorithms available and a lot of research has been done in the area over the years. Load balancing research is studied to get a better understanding of what kind of load balancing technique is suitable for this project. Two types of load balancing algorithms are tested in the evaluation of the distributed system to determine what kind of algorithm that produces the best throughput.
Chapter 3

Load Balancing and Video Processing in Distributed Systems

This chapter discusses the concept of load balancing and different methods and techniques used to achieve load balance. The focus is on how load balancing can affect the performance of distributed video processing systems. With growing importance of Internet-scale services, more complex networked systems are designed and developed. Increasing resources, throughput and availability for this kind of services gives the problem of load balancing high attention and importance. Load balancing involves many different issues, including: load estimation, load level comparison, performance indices, system stability, information exchange, task requirements estimation, task transfer, and more.

3.1 Preliminaries

In a distributed system, where nodes share their computational power, load balancing is the process of distributing and redistributing workload across those computing resources. The aim is to avoid a situation where some of the computing resources are under heavy load and others are idle, see Figure 3.1. The main goal is to produce a global improvement in system performance [4]. Similar techniques exist, like load sharing, which avoids having idle computing nodes, when others have too much work. But load balancing differs from load sharing in one important aspect: load balancing strives to balance the load on all machines at all times whereas load sharing just avoids idle computing resources [10].

Load balancing enables tasks to be moved between computing resources in a distributed system. This improves resource utilization, improves the overall throughput of the system and minimizes response times. The workload is roughly equalized among all nodes in the distributed system and avoids overloading single resources, see Figure 3.2. There are three main categories of load balancing algorithms: Static Load Balancing (SLB) algorithms, that base decisions on knowledge available in advance about the global state of the system; Dynamic Load Balancing (DLB) algorithms, that base decisions on current state of the system [4]; and Adaptive Load Balancing (ALB) algorithms, that are a special class of DLB algorithms that adapt their activities by dynamically changing their parameters [26]. The three categories of algorithms are further explained in the following sections.
Chapter 3. Load Balancing and Video Processing in Distributed Systems

Figure 3.1: A distributed system without load balancing. Some of the servers are under heavy load while others are idle.

Figure 3.2: A distributed system with load balancing. All servers are moderately loaded.

3.2 Static Load Balancing

SLB is achieved by mapping a set of tasks to a set of computing resources such that a performance function – that models the time taken to process the task – is minimized. Figure 3.3 illustrates the two types of SLB algorithms: deterministic algorithms that always produce the same load balancing with a given input, and probabilistic algorithms where the load balancing outcome also depends on chance. Neither type consider the current state of the system [4]. SLB algorithms performs load balancing tasks before program execution begins. For example, a task is always executed on the node that is assigned the task, i.e., SLB is always non-preemptive [23]. This is one of the major drawbacks with this type of approach as this has a huge impact on the overall system performance if the load is unpredictable and fluctuates. On the other hand, if the load is predictable SLB may be a good approach. Some of the more commonly used SLB algorithms are: Round Robin, Randomized, Central Manager and Threshold [25, 22, 20].

Figure 3.3: SLB algorithm classes.
3.3 Dynamic Load Balancing

**Round Robin** is a deterministic algorithm, that evenly distributes tasks to all nodes in Round Robin order, i.e. in circular order. Each node is assigned the same number of tasks without considering task size. Thus a Round Robin algorithm works well if the workload for each task is close to equal. Varying workloads creates a situation as described by Figure 3.1, which makes the performance of Round Robin bad [25].

**Randomized** is a probabilistic algorithm, that uses random numbers to assign tasks to nodes. Random numbers are generated based on a statistic distribution and nodes are chosen following those numbers [20]. Just like Round Robin, Randomized can be subject to the situation described by Figure 3.1.

**Central Manager** is a deterministic algorithm, that lets a central master node do all load balancing decisions. The master node is able to gather load state information from the other nodes. Tasks are assigned to nodes based on the load state information. However, tasks are non-preemptive, thus a task assigned to a node must be executed on the same node [20, 25].

**Threshold** is a deterministic algorithm, that uses load thresholds to categorize nodes into one of three categories: underloaded, medium, or overloaded. At first, all nodes are considered underloaded. When a node moves from one load state to another, the load state is broadcast to all other nodes. This way each node keeps track of the state of all nodes. Tasks are allocated locally if the node is not overloaded. If the node is overloaded the node tries to assign the task to a node that is not. If no such node is found, the task is allocated locally. A disadvantage with Threshold algorithms is that if all remote nodes are overloaded all tasks are allocated locally. The load on one overloaded node can be much higher than on another overloaded node, and this may cause disturbances and decreased performance [25].

### 3.3 Dynamic Load Balancing

The main difference between SLB and DLB is that the workload is redistributed among nodes during runtime. The load can be redistributed among the nodes if some of the nodes are underloaded. Depending on the nature of the applied strategy, DLB algorithms can be characterized as **distributed** or **non-distributed**: non-distributed algorithms are classified as either **centralized** or **semi-centralized** [4]; and distributed algorithms are classified as either **local** or **global**. Global algorithms are further classified as either **cooperative** or **non-cooperative**. This distinction is depicted in Figure 3.4.

Distributed algorithms let all nodes in the network take part in the load balancing decisions. This often requires a lot of communication among the nodes in the distributed system. There are generally two strategies: local strategies that divide nodes in groups where load balancing decisions are made in each group locally; and global strategies where all nodes are considered as one global group [21]. In the global strategy, the interaction between the nodes take two forms: cooperative form where all nodes in the system work together towards a global objective, e.g., to improve the overall system response time; and non-cooperative form where each node in the system works towards a independent goal, e.g., to decrease local response time [4]. The main advantage of distributed algorithms is
that they are failure tolerant. The failure of one node does not cause the load balancing operation to halt, it merely degrades its performance, i.e. a single point of failure is avoided.

Non-distributed algorithms delegate the load balancing responsibility to a single node or to some of the nodes. This makes for two types of non-distributed algorithms: centralized, where one central node is solely responsible for the load balancing decisions; and semi-centralized, where the nodes are divided into clusters, load balancing is then performed by a central node in each cluster [4, 26]. The central nodes of the clusters cooperate with each other to load balance the whole distributed system.

Distributed load balancing algorithms tend to generate a lot more communication overhead than non-distributed algorithms. This is due to the fact that each node takes part in the load balancing decision as opposed to one or a few nodes in the non-distributed case. Conversely, the centralized approach has a single point of failure that is avoided by distributed algorithms, namely the solely responsible node. Centralized algorithms are more suited for small sized distributed systems, this has been shown by Zhou [32]. For larger sized distributed systems, Ahmad et al. [3] shows that distributed algorithms or semi-centralized approaches are more efficient.

DLB algorithms are required to make decisions based on current system state. The information that the decisions are based on need to be collected from the nodes in the system. The algorithm must be able to determine if a node can participate in a task transfer, i.e., the movement of a task from one node to another, identify which tasks that are qualified for load balancing, and locate other nodes that can participate in task transfers. Consequently, the four components of a DLB algorithm are a information, transfer, selection, and location policy [4]. The relationship between the different components is depicted in Figure 3.5 and the different components are explained further below.

**Information policy**

One of the most important parts of a DLB algorithm is the collection of system state information [16]. The information policy decides what, when and where information should be collected. It is responsible for providing the state information needed by the transfer, selection and location policies. Mainly two types of state information are gathered: local and global state information. Local state information is data gathered from neighboring nodes and the local node itself. Global state information is data gathered from all the nodes in the system. There is a trade-off between the quality of the collected information and the communication overhead that is created by it.
Figure 3.5: The components of a DLB algorithm.

More qualitative data require larger data structures that creates more communication overhead [4].

**Transfer policy**

The load state of the node is determined by the transfer policy [16]. Using the load information collected by the information policy the transfer policy can determine whether the node is suitable to participate in a task transfer [22]. This is done by determining if the node is under- or overloaded. This is commonly done using threshold values expressed in units of load, e.g., if \( \text{load} < T_1 \) the node is underloaded, and if \( \text{load} > T_2 \) the node is overloaded [16].

The transfer policy determine whether the node can act as a **sender** or a **receiver** of a task in a task transfer. Overloaded nodes want to ease their workload and are thus classified as senders. Underloaded nodes are willing to take on more work and are thus classified as receivers. Nodes that are either, i.e. moderately loaded, cannot participate in task transfers [4].

**Selection policy**

If the transfer policy determines that the node is a **sender**, a selection policy decides which task is to be transferred. If the selection policy cannot find a task suitable for transfer the node is no longer considered a sender. There are several factors to take into account when selecting a task for transfer, but the basic criterion is that the reduction of response time of the system must make up for the overhead of the transfer [16].

**Location policy**

The responsibility of the location policy is to find the best available node among all available nodes to transfer the task to. The location policy have to consider which nodes that fulfill the necessary prerequisites to execute the task, the availability of the destination node, how many times a task is eligible for transfer etc. There are a variety of different location policies proposed in literature: random policies that selects a node at random [4]; polling/probing policies that sends polls to collect load state from nodes, this information is then used to select a suitable node for task transfer, e.g., select a node at random, poll node for load state, repeat until a light loaded node
is found \cite{16}; negotiation policies that let nodes negotiate with each other in order to select the best node for task transfer. This kind of policy may use some kind of bidding or voting mechanism to make a decision.

### 3.4 Adaptive Load Balancing

The difference between dynamic and adaptive load balancing algorithms is small. Adaptive algorithms is a special class of dynamic algorithms. Adaptive algorithms adjusts their load balancing behavior by dynamically changing their parameters or policies \cite{7, 26}. For instance, if some policy works great for heavily loaded systems and another for lightly loaded systems, a simple adaptive algorithm could change policy depending on the system load. Notable, also in cases when no improvement would be gained by transferring a task under heavy load, a non-adaptive dynamic algorithm would continue to operate according to its policies, and thus increase overhead.

### 3.5 Task Granularity

In parallel and distributed computing, the term task granularity refers to how jobs are divided up into smaller atomic tasks. The granularity of the tasks have an impact on the efficiency of load balancing and scheduling of task executions. Fine-grained processing splits a job into a large number of smaller tasks, while coarse-grained processing splits a job into larger and more time-consuming tasks. The more fine-grained a job is split the more work can be done in parallel (assuming we have enough computing resources). More tasks results in more time spent on synchronization and transferring of tasks between nodes. There is a trade-off between time spent on communications and time spent on computations. Larger number of tasks gives more opportunities for load balancing as more tasks can be selected for transfer between computation resources. But as the primary goal of load balancing is to increase performance of the system, a balance need to be found between task granularity and load balancing opportunities. The task granularity is determined by how the data is partitioned. In the next section we look at how video files are structured and how this affects how we can partition the video data and thus how this affects the task granularity and in turn load balancing of video processing systems.

### 3.6 Video Data Components

In a parallel or distributed processing systems the throughput is usually increased by dividing the work between a set of computing resources. This requires partitioning of larger tasks into smaller subtasks while ensuring that each subtask can be performed individually. In this scenario, understanding the structure of the input data is an important step to understanding how the data can be partitioned. In a video processing system the input is video and audio data, typically video and audio streams, that are compressed and contained in a media container. The container file contains metadata describing how video and audio data co-exist in the file, i.e., data needed to extract the video and audio from the file.

Basically a digital video stream is a sequence of digital pictures (also known as frames) displayed one after the other at some rate (Frames per Second (FPS)) creating the appearance of a moving picture. A digital video frame is a 2-dimensional array of pixels, where each pixel value represents the color and intensity of a specific spatial location at a specific
3.7. Video Compression

3.7. Video Compression

Digital video have significant redundancies and video compression is performed by reducing or removing those redundancies. There are mainly four types of redundancies that are exploited namely perceptual, temporal, spatial, and statistical redundancies [15].

The RGB color-space that is usually used when capturing video creates perceptual redundancies that exist because of the Human Visual System (HVS). Everything that the human eye cannot perceive can be discarded without affecting how humans perceive the picture. The Luminance, Chrominance blue, Chrominance red (YCbCr) color-space matches the HVS much better than the RGB color-space, thus converting the picture color space from RGB to YCbCr allows better exploitation of the perceptual redundancies. The HVS is more sensitive to brightness (Luminance (Y)) than color (Chrominance blue (Cb) and Chrominance red (Cr)) thus decreasing the resolution or subsampling the color components can save a lot of space without changing how humans perceive the video [15, 23].

The HVS is also sensitive to temporal frequencies – the eye retains pictures for a period of time after the picture have been removed. The length of this time period depends on the lighting condition, under normal conditions a picture is present 1/16th of a second. To exploit this redundancy the frame rate can be selected such that there is just enough time between each frame in the video.

In a video sequence two consecutive images look largely similar. Instead of coding each individual frame, the next frame can be predicted using the previous frame. A lot of space can be saved by only saving the difference between the previous picture and the predicted image, thus exploiting temporal redundancies between consecutive images. Current video encoding systems use a process known as motion-compensated prediction. Predictions are
made by compensating for relative motion between two frames by dividing the current frame into blocks, finding the best matching block from one frame to the other, and calculating an associated motion vector that describes the movement of that block. This creates a strong dependency between frames, as we need to know previous frames in order to decode the current frame. Partitioning of video thus become more difficult as it limits the choice of where a video stream can be cut. For example, in the Moving Picture Experts Group (MPEG) standard [18] three types of pictures are considered:

**Intra coded frames (I-frames)** are coded without reference to other frames, they provide random access points (seek points) for where the decoding can start. I-frames do not require any other frames to decode, they are used as the reference frames for **Bi-directional coded frames (B-frames)** and **Predictive coded frames (P-frames)**.

P-frames are coded with motion compensated prediction from a previous frame (I-frame or P-frame) and are therefore more compressed than I-frames as they only contain the difference from the previous frame.

B-frames are coded using both a past and a future reference frame and offers most compression. However B-frames cannot be used as references themselves.

![Figure 3.6: Interframe codings showing relationship between MPEG frame types.](image)

The relationships between the different frame types are illustrated in Figure 3.6. The figure also shows, what happens if a video sequence is split and does not start with an I-frame. Frames go lost as they cannot be decoded. This difficulty limits the partitioning choices of video streams.

### 3.8 Video Bit Stream Structure

There exist a lot of different standards that specify the structure of compressed video that have been produced to ensure that video producing and consuming devices are interoperable. The structure of the bit-stream depends on the compression algorithm and/or standard that is used, but the general structure is similar.

Figure 3.7 shows a simplified view of the video bit-stream structure that is the result of MPEG video compression [18, 14], one of the most common compression standards in use today. The structure of the video bit stream can be considered a hierarchy in which the
Figure 3.7: A simplified view of the MPEG video bit stream.
structure contains one or more underlying structures. The figure shows a simplified view of the hierarchy. Appropriate headers exist for each layer that contain information necessary to process the data in the bit stream. The layers are as follows:

**The Sequence Layers** contains information about video parameters like width, height, aspect ratio of pixels, bit rate, buffer size etc., as well as one or more Group of Pictures (GOP).

**The GOP Layers** contains time code information, structure of the GOP, frame order etc., as well as one or more pictures. A GOP always starts with an I-frame to ensure that the decoding process can start from the beginning of any GOP. The GOPs provide random access in the video, so called seek points. A GOP is a decodable unit, i.e., it can be decoded by itself.

**The Picture Layers** contains information about each picture, like the picture type (I-frame, P-frame or B-frame), buffer parameters, encode parameters, and all slices that make up a picture.

**The Slice Layers** contains information about where each slice starts, quantizer scale codes etc., a slice consist of one or more macroblocks.

**The Macroblock Layers** contains macroblocks, that are the basic coding unit for motion compensation, basically it is a 16 x 16 pixel segment in a slice and contains the color information for the block (Y, Cb, and Cr blocks).

**The Block Layers** contains blocks, that are the smallest component and consists of 8 x 8 pixels that represent either Y, Cb or Cr values.

From the perspective of video partitioning this hierarchy allows segmentations on different levels. Depending on which layer the partitioning is performed, the task granularity may differ.

### 3.9 Audio Bit Stream Structure

There are a variety of different standards for audio compression, but the basic structure of the bit stream is the same. Again, the MPEG standard [19] is used as an example. The audio bit stream structure is illustrated in Figure 3.8. Depending on the sampling rate and bit rate of the stream, the frames are of different sizes. Generally, frames are independent, i.e. each frame can be decoded independently of one another. However for some compression techniques this is not true, e.g., the internal data organization for MPEG Layer III files may make frames dependent on previous frames.

When partitioning audio bit streams this set some limitations, the smallest partition size is set by the frame size in the case where there is no data dependency. For audio streams with data dependencies like MPEG Layer III some data might be lost (a fraction of a second) if the stream is partitioned between two frames. However it is possible to decode the audio stream and re-encode it to ensure that no data is lost.

### 3.10 Conclusions

There are multiple approaches to load balancing, that can be divided into static, dynamic, and adaptive methods. The choice of load balancing technique depends on the problem
3.10. Conclusions

A qualitative comparison between some of the more known load balancing algorithms is performed by Sharma et al. [25] and Rajguru et al. [22]. Some of the conclusions made by Rajguru et al. [22] are that SLB algorithms are more stable than dynamic ones, but DLB algorithms are more fault tolerant, adaptable, and reliable. As different qualitative measurements are important in different cases, it is not possible to design a load balancing algorithm that is the best for all kinds of situations, even if the problem domain is restricted to video processing.

One of the factors that affect the choice of load balancing algorithm is the task granularity. When partitioning the data that is to be distributed, task granularity must be considered. By looking more closely on how video data is organized, we can conclude that there are many different layers in the data hierarchy that leaves many partitioning choices. The type of processing that is performed on the video bit-stream affects the partitioning options. Most likely an application wants to decode the video stream and do computations on the contents of it, this leaves partitioning on the the GOP layer as the most likely choice as it is decodable by itself.

If the type of work is known and if the workload can be estimated, a SLB algorithm performs well. However many real world problems have variable or dynamic workload. Video processing is such a problem because the bit-stream structure differs between video compressions and therefore it is not possible to predict the workload at compile time. Thus, a more dynamic approach is suitable for video processing. What type of dynamic or adaptive load balancing algorithm is the best for video processing is hard to say at this point, real experimentation would need to be performed in order to determine this. Some experiments on DLB algorithms are performed by Zaki et al. [31]. Dynamic distributed and non-distributed algorithms are tested on a homogeneous cluster. The experimental results show that when the workload for each task is uniform, global strategies work better when the computation to communication ratio is large, and that this tilts towards local strategies when the ratio decreases. Depending on how a video or audio stream is partitioned both local and global dynamic and/or adaptive load balancing algorithms may perform well.

Figure 3.8: A simplified view of the MPEG audio bit stream.

at hand. Different load balancing schemes are best for different applications under various system conditions and parameters.

Audio Frame

<table>
<thead>
<tr>
<th>Frame</th>
<th>Frame</th>
<th>Frame</th>
<th>Frame</th>
<th>Frame</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Audio stream
Chapter 4

Distributed Video Content Analysis Architecture

This chapter describes the distributed VCA architecture, the overall design and the implemented systems description. The interaction between the different parts of the system is described and the data flow is explained in more detail.

4.1 Introduction

VCA is split into three categories: audio content analysis, which includes analysis of voice (e.g. ASR), music, and various kind of sound; visual content analysis, which includes analysis of visual features, e.g. color, motion, object tracking, and face detection; and audiovisual content analysis which combines the audio and visual content analysis to get better content understanding. VCA is performed by segmenting the video sequence into smaller building blocks, e.g. into a series of frames and performing analysis on them individually. Distributed VCA is a parallel application where several individual instances of a program do VCA but only to a given part of the input video.

CodeMill previously developed a ASR plug-in as a part of a MAM platform developed by Vidispine. The ASR plug-in performs audio content analysis to extract subtitles from a video source. Processing video with the ASR plug-in is time consuming and in an effort to increase the efficiency a distributed VCA architecture is proposed and implemented. The goal is that the architecture should be general for all types of VCA.

4.2 Overall System Design

The distributed VCA system consists of a peer-to-peer network of nodes (individual instances), as illustrated in Figure 4.1, each running a distributed application with VCA capabilities. VCA is enabled by plug-ins to the Vidispine Transcoder (VT), in this case a ASR plug-in. The basic idea is that a VCA task can be started from any node, the task is then divided into sub-tasks that are split between the nodes and the result is returned to the node that started the task, in a scatter-gather pattern, see Figure 4.2.
There are two main parts within the peer-to-peer application: a worker and a master part, Figure 4.2 illustrates their relationship. A node that starts a VCA task is the master of the task, all other nodes are considered workers during the execution of the task. The master node is responsible for preprocessing (e.g., partitioning and task monitoring) and post processing (e.g., correct metadata timestamps). The worker nodes receives and performs VCA on the assigned partitions and return the generated metadata to the master.

The logical architecture of the distributed application is depicted in Figure 4.3. All nodes act as master for tasks initiated at the node and as worker for incoming sub-tasks from other nodes. When a VCA task is started:

- The target video file is partitioned by the Partitioner.
- New atomic sub-tasks are created for each video file partition by the Sub-Task Generator.
- The sub-tasks are registered with the Task Monitor that keeps track of the execution status of each sub-task.
- Sub-tasks that are not assigned to a worker are sent to the Load Balancer, that determines for each sub-task if they are to be executed locally or transfered to another node.
- Locally scheduled sub-tasks are handled and queued by the Work Handler.
- Local workers handled by the Work Handler process sub-tasks using the Analyzer.
- Completed sub-task results are sent over the network or locally back to the Task Monitor that keeps track of which sub-tasks that are completed and which are awaiting results.
Figure 4.2: Illustration of the master-worker relationship. The master scatters the workload across the workers that does the analysis and returns the partial metadata result to the master that gathers and compiles the final metadata result.

- When all sub-tasks are completed the Task Monitor returns the completed result.

- The completed results may be run through a Post Processor that performs various kind of post processing on the results, e.g. correct errors or write the results to a file.

- Connections with other nodes in the network are handled by the Group Manager, that keeps track of nodes joining and leaving the system.

The architecture have many similarities to a Map-Reduce system as it coordinates the distributed nodes, runs tasks in parallel, manages communication and data transfers, and provides redundancy and fault tolerance [6]. However there are many differences as well, e.g., the Map-Reduce model parallelizes preprocessing (Map), in this architecture the pre-processing is performed by a single node (the master). Furthermore, Map-Reduce models are usually very general while this architecture is specific to VCA.
Figure 4.3: The logical architecture for the distributed application. The arrows show the data flow in the system and how the different modules interact.
4.3 Software and Libraries

A number of softwares and libraries is used to aid the development of the distributed application. The focus of the thesis is on the work of creating a distributed architecture for VCA and not on the implementation of VCA algorithms. Therefore existing software from Vidispine and CodeMill is used for the actual VCA. Furthermore some open source software and libraries are used to help speed up the development and implementation of the distributed architecture. A description of the main softwares and libraries and their roles are as follows:

**Vidispine Transcoder**

The VT is a part of the MAM platform developed by Vidispine and is a general purpose transcoder with VCA capabilities. The VT supports plug-ins that can be used to implement various analysis algorithms on the video and audio data passing through the transcoder. The VT is a proprietary software and owned by Vidispine, thus it is only available to Vidispine customers and partners. The VT can be compiled and installed on common operating systems like Linux, Mac OS X, and Windows. In this project the VT is used in order to run ASR analysis using a plug-in developed by CodeMill. The VT have a HTTP XML/REST API that e.g., is used to queue jobs, get status of jobs and to get job results. Metadata generated by the plug-ins is returned via HTTP PUT callbacks, the interaction with the VT is described in Figure 4.4.

![Figure 4.4: The Vidispine Transcoder HTTP XML/REST API. A HTTP client is able to post and queue jobs for execution, status of jobs can be retrieved in HTTP responses. Metadata generated by plug-ins are returned by a HTTP PUT request to a url defined in the job request, this requires a HTTP server instance that can receive the request.](image)

**Codemill Speech-To-Text**

*CodeMill Speech-To-Text* (CSTT) is a plug-in to the VT. The plug-in implements ASR analysis on audio passing through the VT. The plug-in uses CMU Sphinx, which is an open source speech recognition toolkit to perform the ASR analysis [27]. The plug-in produces subtitle metadata from the audio stream containing start and stop timestamps and text.

**Apache Zookeeper**

*Apache Zookeeper* (Zookeeper) is a coordination service for distributed applications that exposes services such as naming, configuration management, synchronization, and group services [33]. The Zookeeper data model uses a hierarchical name-space of that is much like that of a standard file system, which makes it easy to work with and learn. According to Brewer's CAP theorem [9], in a distributed system it is impossible to
guarantee: (strong) consistency i.e. a total order on all operations such that all nodes see the same data at the same time, (high) availability i.e. every request received by a non-failing node result in a response, and partition tolerance i.e. continuous operation despite arbitrary data loss or failures of a part of the system. It is however possible to guarantee two out of the three. Zookeeper aims to guarantee consistency and partition tolerance, and tries to offer availability through replication. The fast learning curve, and consistency guarantee (makes it possible to build distributed synchronous consistency primitives like queues and locks) made Zookeeper a good choice for this project.

FFmpeg

FFmpeg is a multimedia framework that is able to decode, encode, transcode, mux, demux, stream, filter and play most media formats [8]. The FFmpeg framework is widely used in multimedia applications, and offers an API available for the C language. The FFmpeg library is used in this project in order to implement partitioning of video and audio files, an important step towards distributing the analysis workload.

JavaCPP

JavaCPP is a Java to C/C++ bridge library. JavaCPP provides access to native C/C++ libraries inside Java by mapping C/C++ features using Java Native Interface (JNI). JavaCPP provides configurations for common C/C++ libraries including FFmpeg.

4.4 System Description

There are no restrictions to programming language or tools used in this project. However, the system must run on standard desktop computers running Linux, Mac OS X, and Windows. Java is used as the programming language for the project due to its cross platform benefits. Zookeeper is a main building block for the application and its main interface is available in Java, which further prompts the choice. The VT that is used to perform ASR is written in C++, but have a REST interface, which makes it possible to use and invoke it from Java. The only drawback with using Java for the project is that there is no library available that is comparable with FFmpeg for processing and partitioning video. This is solved using JavaCPP, which creates a bridge for C/C++ libraries to Java. This way it is possible to gain access to the native FFmpeg libraries. Although, in order to still support multiple architectures the native libraries have to be compiled for each platform (Linux, Mac OS X, and Windows).

The main part of the software developed in this project is the peer-to-peer application running on each node. The application consists of eight modules; a Partitioning Module (PM) that segments video and/or audio files; a Sub-Task Builder Module (SBM) that is used to create sub-tasks that describe the analysis work to be done for each video partition; a Task Monitoring Module (TMM) that monitors the task execution progress and ensures that all sub-tasks are executed; a Load Balancing Module (LBM) that make load balancing decisions; a Group Management Module (GMM) that handles group membership issues and keeps track of other nodes connected to the network; a Work Handling Module (WHM) that handles and queues the work to be performed for each sub-task; a Analysis Module (AM) that performs the actual VCA analysis on the partial files; and a Post Processing Module (PPM) that performs post processing on tasks after completion, e.g., correct errors and generate subtitle files. Next, the individual models are described in more detail.
4.4. System Description

Partitioning Module
The PM is responsible for partitioning video into one or several video segments. Two different partitioners are available: *FFmpegPartitioner* and *NoPartitioner*. The FFmpegPartitioner is implemented using the *FFmpeg* API. The functionality of the FFmpegPartitioner is depicted in Figure 4.5. The input media container is *de-multiplexed* (step 1), i.e., unpacked, to expose the video and audio streams that make up the video. The video and audio streams are then divided into segments of approximately equal length (step 2) and later *multiplexed* (step 3), i.e., packed, to separate media containers containing the segmented streams. The video is segmented such that each segment start with a video *keyframe*, or I-frame as described in Chapter 3, to make the segment a single decodable unit. The minimum segment size is determined by the GOP size (number of frames between two keyframes). This guarantees that each segment is decodable by itself but limits the cutting choice. By using native libraries like FFmpeg, the platform is no longer independent despite the use of Java. FFmpeg must thus be compiled for each platform that is supported. The FFmpegPartitioner also supports the use of VAD to limit video cuts to areas of silence in the video, however the functionality is highly experimental and may cause errors. The *NoPartitioner*, does not partition the input video at all and is useful for testing purposes. The PM can be extended with new partitioners by implementing a *Partitioner* interface.

Sub-Task Builder Module
The SBM packages each video partition generated by the PM into sub-tasks that can be distributed to other nodes. The sub-tasks contain descriptive data about the video partition and the work to be performed on it, e.g., the type of analysis to be performed, the location of the partition file in case it needs to be downloaded, where to return the result, start and end timestamps in relation to the original video etc. One builder exists for each type of VCA, currently one builder is available: the *SpeechToTextSubTaskBuilder*, which builds sub-tasks with the information needed to perform ASR analysis on the video partition. More builders can be added by extending the *SubTaskBuilder* class.

Task Monitoring Module
One of the requirements set for the system is that it must handle machine failures gracefully. The TMM handles machine failures throughout the distributed VCA. When a task is started it is registered with the TMM. The TMM keeps a record of the state of each sub-task that make up a task. There are three different states: *unassigned*, the
starting state before the sub-task have been assigned to a node; in-progress, set by a node when the sub-task is assigned to the node; and completed when the result for the sub-task is completed. The module periodically checks the state of each sub-task and take actions depending on the state. For example, unassigned tasks may be sent to the load balancer to be forwarded to another node, and sub-tasks that are in-progress for too long may be rescheduled. When a node fails or crashes, the sub-tasks that was set in-progress by the crashing node is reset and assigned to a healthy node. This induces a slight overhead in execution time, but makes the system more reliable and robust to failures. The Task Monitoring Module is asynchronous and runs in a separate thread. Machine failures are detected by utilizing Zookeeper ephemeral nodes, which are special data nodes in Zookeeper that are removed when the client that created the node disconnects. The operation of the TMM is depicted in Figure 4.6.

![Figure 4.6: Description of the Task Monitoring Module functionality.](image)

### Load Balancing Module

The LBM make load balancing decisions for locally generated sub-tasks, and sub-tasks arriving from remote nodes. The main functionality of the load balancing module is depicted in Figure 4.7. The LBM queues sub-tasks sent to it (step 1) and processes the sub-tasks in a separate load balancing thread. For each sub-task, the active load balancing implementation decides (step 2) whether the sub-task is executed locally or transferred to another node (step 3 or 4). Two types load balancing algorithms is implemented by the LBM: RoundRobinLoadBalancer, which implements a static Round Robin load balancing algorithm; and the DynamicLoadBalancer, which implements a DLB algorithm. The DLB algorithm uses an information policy that gathers global system state information, i.e., load state is gathered from all nodes in the system. The information that the information policy gathers is: CPU load, memory usage, work queue length, average task waiting time, sub-task processing speed, and the total number of executed sub-tasks. The load state information is gathered on demand, i.e., the information is updated when the node take part in any load balancing decision. The threshold location policy determine node load states by measuring work queue length and sub-task processing speed. The selection policy estimates queue time, execution time and download time in order to select sub-tasks that increase overall system performance. The location policy randomly polls nodes to find transfer partners. The LBM is designed to make it easy to create new load balancing implementations, by
extending the LoadBalancer base class.

**Group Management Module**

The GMM handles all peer-to-peer connections to other nodes and keeps track of connected and available nodes. The GMM uses Zookeeper to share node connection information for peer-to-peer connections. The functionality of the GMM is depicted in Figure 4.8. When a node is started the GMM registers the client as available (step 1) by writing its connection information to a preconfigured location in Zookeeper. To discover available nodes the GMM queries Zookeeper for all connection information (step 2). Once the connection information is read (step 3), the GMM uses Java RMI for the actual peer-to-peer communications (step 4) with the remote GMMs. Disconnecting clients get deregistered by removal from Zookeeper.

---

**Figure 4.7**: Description of the Load Balancing Module functionality.

**Figure 4.8**: Description of the Group Management Module functionality.
Work Handling Module

The WHM handles the local queue of sub-tasks with pending execution and all worker threads that consume sub-tasks from the local queue. The functionality is illustrated in Figure 4.9: The worker threads consume sub-tasks in the local queue (step 1), finds the type of analysis to perform by inspecting the sub-task, downloads the video data if necessary from the master node (step 2), finds the appropriate analyzer, executes the analysis (step 3) and returns the result of the analysis to the master node that created the sub-task (step 4).

Analysis Module

The AM contains analyzers that perform VCA on video partitions, currently one type of analyzer is available, namely the *SpeechToTextAnalyzer*, which utilizes the VT and the CSTT plug-in to do ASR analysis. The operation of the SpeechToTextAnalyzer is illustrated in Figure 4.10. When the SpeechToTextAnalyzer receives an analysis request (step 1) it starts an ASR analysis job by sending a job request via HTTP/REST/XML to the VT (step 2). During the analysis the VT sends generated metadata via HTTP PUT requests to a url defined in the original job request (step 3). The SpeechToTextAnalyzer utilizes a minimal HTTP server to collect the generated metadata. The VT is polled by the SpeechToTextAnalyzer until the job is completed (step 4 and 5). When the job is completed, all collected metadata results sent to the analyzer is time-stamp corrected and sent back as the result of the analysis request (step 6). It is possible to extend the module with more kind of analyzers by implementing the Analyzer interface.
Post Processing Module

The PPM contains post processors that can be used to post process the result of the VCA task, e.g., to correct time stamps and generate subtitle files from ASR analysis. There exists two post processors: SubRipPostProcessor, that writes subtitle metadata in SubRip format to a file; and TimesPostProcessor, that writes execution time data to a file. The module can be extended with more post processors by implementing the PostProcessor interface.

In addition to the modules presented above there is an interactive Command Line Interface (CLI) that e.g., can be used to start VCA tasks, view statistics of the tasks in progress, and to list available nodes. For installation and configuration instructions, see Appendix A.
Chapter 5

Performance Evaluation

In this chapter the performance evaluation of the distributed VCA system is presented. The VCA system currently in use is compared to the distributed version developed in this thesis. Accuracy, speedup, crossover point, and throughput of the system is evaluated. The goal of this chapter is to answer the questions stated in Section 1.2:

1. What speedup, relative to the current system, can be expected using the distributed architecture?

2. How long video duration (problem size) must be analyzed for it to be beneficial to use the distributed architecture over the current system?

3. What method of load-balancing achieves the highest throughput for a given set of machines?

5.1 Setup

The evaluations are run using Amazon Web Services (AWS) using the Elastic Compute Cloud (EC2) web service that offers re-usable cloud hosting [24]. Using Amazon EC2 it is easy to scale horizontally, i.e., add or remove nodes. This makes EC2 a good tool for testing scalability and speedup. It is also possible to scale vertically, i.e., add or remove node resources. This makes it possible to run both heterogeneous and homogeneous clusters of computers. AWS offers a pay-per-use billing model, i.e., a hourly rate per server instance. Furthermore, EC2 offer dedicated tenancy (single-tenant hardware), i.e., instances that do not share hardware with other instances. All tests are run using dedicated tenancy. Table 5.1 shows the main properties of the virtual machines used for the experiments.

Table 5.1: Amazon AWS platform used for the performance evaluation.

<table>
<thead>
<tr>
<th>Machine</th>
<th>CPU</th>
<th>Memory</th>
<th>Network</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3 medium</td>
<td>1 core vCPU (Intel Xeon E5-2670 v2)</td>
<td>3.75 GiB</td>
<td>100/100 Mbps</td>
<td>SSD</td>
</tr>
<tr>
<td>M3 large</td>
<td>2 core vCPU (Intel Xeon E5-2670 v2)</td>
<td>7.5 GiB</td>
<td>100/100 Mbps</td>
<td>SSD</td>
</tr>
<tr>
<td>C3 xlarge</td>
<td>4 core vCPU (Intel Xeon E5-2670 v2)</td>
<td>3.75 GiB</td>
<td>100/100 Mbps</td>
<td>SSD</td>
</tr>
</tbody>
</table>

The dataset used in the evaluations consists of videos of TED talks. TED is a nonprofit organization devoted to spreading ideas, usually in the form of short, powerful talks [29].
The videos from TED are free to download and have transcriptions available. The amount of available videos and transcriptions of TED videos make them good for testing ASR. The videos are about 10 - 20 minutes long, have both male and female speakers, and have high resolution video and audio streams. Four different videos are used in evaluations, with an equal number of female and male speakers. This gives varying input in the case of ASR, as each speaker has differences in tone and voice. Table 5.2 contains descriptions of the videos used in the evaluation.

Table 5.2: Test videos used in the performance evaluation.

<table>
<thead>
<tr>
<th>No.</th>
<th>Video File</th>
<th>Video Stream</th>
<th>Audio Stream</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AJJacobs_2014A-480p.mp4</td>
<td>H.264/MPEG-4</td>
<td>AAC/MPEG-4</td>
<td>00:09:45</td>
</tr>
<tr>
<td></td>
<td>24 FPS</td>
<td>44100 kHz</td>
<td>Male speaker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420p</td>
<td>Stereo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>JorgeSoto_2014G-480p.mp4</td>
<td>H.264/MPEG-4</td>
<td>AAC/MPEG-4</td>
<td>00:11:21</td>
</tr>
<tr>
<td></td>
<td>24 FPS</td>
<td>44100 kHz</td>
<td>Male speaker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420p</td>
<td>Stereo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>KareAnderson_2014S-480p.mp4</td>
<td>H.264/MPEG-4</td>
<td>AAC/MPEG-4</td>
<td>00:09:50</td>
</tr>
<tr>
<td></td>
<td>24 FPS</td>
<td>44100 kHz</td>
<td>Female speaker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420p</td>
<td>Stereo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SusanEdinger_2014S-480p.mp4</td>
<td>H.264/MPEG-4</td>
<td>AAC/MPEG-4</td>
<td>00:12:27</td>
</tr>
<tr>
<td></td>
<td>24 FPS</td>
<td>44100 kHz</td>
<td>Female speaker</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420p</td>
<td>Stereo</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2 Speech Recognition Accuracy

The speech recognition accuracy of the system is measured using two common performance metrics for ASR: Word Error Rate (WER) and Word Accuracy Rate (WAR). WER is derived from the Levenshtein Distance (LD) which is a string metric for measuring the difference between two character sequences. The LD is the number of deletions, insertions or substitutions required to transform a character sequence to another [11]. The WER is calculated using Equation 5.1, where \( D \) is the number of deletions, \( I \) is the number of insertions, \( S \) is the number of substitutions and \( N \) is the number of words in the reference. The reference in this case is the TED talk transcripts. WER is the de-facto standard used to measure the performance of speech recognition systems. WAR is the inverse of WER and is calculated using Equation 5.2. WER and WAR is measured for different partition sizes to examine the impact made by the distributed system, in particular the partitioning, to the correctness of the transcriptions. All four videos in Table 5.2 are used in the WER and WAR evaluations.

\[
WER = \frac{D + I + S}{N} \tag{5.1}
\]

\[
WAR = 1 - WER \tag{5.2}
\]

It is important that the correctness of the analysis is not affected by using the distributed system instead of the serial system. The WER and WAR experiments try to determine how the accuracy is affected by workload distribution, i.e. partitioning. Figure 5.1 illustrates how the partition size affects the WER and WAR average for the four videos. A detailed table of the average test data and the results for each individual video can be found in
5.3 Speedup and Efficiency

Appendix B. As shown by Figure 5.1 the WER is larger for the smaller partition sizes and decreases as the partition size increases until the partition size is roughly greater than 25 s.

The speech recognition accuracy results show that small partition sizes produce more errors. A small partition size produces more partitions that lead to a higher chance that the video sequence is cut in the middle of a spoken word, which causes more erroneous classifications. The results, however, sometimes show a performance increase even though partitioning is performed, e.g., a partition size of 120 in the results of the WER analysis of video 1 (see Table 5.2) in Figure B.1 and Table B.2, Appendix B. This effect may be caused by the inner workings of the ASR algorithm. A review of the ASR algorithm shows that it uses *Gaussian Mixture Models* (GMMs) - parametric probability density functions - to model speech, whose parameters are estimated from training data taken from the beginning of the input video. Thus, when partitioning is used, more training is performed on data temporally closer to the data that is being analyzed, which may be the cause of the improved performance of the ASR analysis.

5.3 Speedup and Efficiency

In order to answer the first question the speedup and efficiency is compared between the distributed and the serial VCA system. The speedup is the ratio of the time taken by the serial system (as we are interested in speedup relative to the current system) to execute VCA to the time taken by two or more nodes to execute VCA. Equation 5.3 is used to calculate the speedup. As this equation only holds for homogeneous architectures, M3 medium instances are used in the whole cluster. \(T_{\text{serial}}\) is the average execution time for the serial system running ASR and \(T_N\) is the average execution time for a distributed execution of the ASR analysis using \(N\) nodes. Because of limited running time available on the
AWS test cluster, one of the four videos take part in this evaluation, video 1 in Table 5.2. The theoretical maximum speedup that can be achieved with an architecture of \( N \) nodes working concurrently is \( N \). This is the ideal speedup, but in practice the speedup is much lower, because of conflicts, communication delays, synchronization delays, uneven workload etc. Using Equation 5.4, a efficiency metric (value between 0 and 1, where 1 is ideal) can be calculated to estimate how well-utilized the nodes are compared to how much time is wasted on communication, synchronization, start, and stop etc. \( N \) is the node count and \( S_N \) is the calculated speedup using \( N \) nodes.

Furthermore, Amdahl’s law\(^1\) can be used to predict the speedup for a different number of nodes than measured. By using Equation 5.5 where \( P \) is the proportion of the program that can be made parallel, a predicted speedup can be calculated for \( N \) nodes. \( P \) is hard to determine, but can be estimated using Equation 5.6 where \( S \) is the measured speedup using \( N \) number of nodes.

\[
S_N = \frac{T_{\text{serial}}}{T_N} \tag{5.3}
\]

\[
E_N = \frac{S_N}{N} \tag{5.4}
\]

\[
S_N = \frac{1}{(1 - P) + \frac{P}{N}} \tag{5.5}
\]

\[
P_{\text{estimated}} = \frac{S}{N} - 1 \tag{5.6}
\]

In the speedup evaluation a fixed partition size of 35 s was used to avoid errors and at the same time produce enough sub-tasks to allow scaling with more machines. The execution time (latency) for running the serial system was measured as well as the execution times for the distributed system with an increasing node count. Because the underlying distribution of the execution time was unknown, 30 measurements were made per setup to get an approximate normal distribution of the measurements. Table B.6 and Figure B.5 in Appendix B presents the skewness and distribution of the speedup measurements. The speedup and efficiency results are calculated from the mean and are presented in Figures 5.2 and 5.3, and Table 5.3. The dashed line represents the ideal speedup, i.e., the linear speedup.

The predicted speedup, calculated using Amdahl’s law, for an increasing node count can be found in Figure 5.4. \( P_{\text{estimated}} \) was calculated from the speedup results for a node count of 8.

Table 5.3: Speedup results for executions with different node counts.

<table>
<thead>
<tr>
<th>Node count</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial (Reference)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.53</td>
<td>0.53</td>
</tr>
<tr>
<td>2</td>
<td>1.03</td>
<td>0.51</td>
</tr>
<tr>
<td>3</td>
<td>1.42</td>
<td>0.47</td>
</tr>
<tr>
<td>4</td>
<td>1.98</td>
<td>0.49</td>
</tr>
<tr>
<td>6</td>
<td>2.62</td>
<td>0.44</td>
</tr>
<tr>
<td>8</td>
<td>3.77</td>
<td>0.47</td>
</tr>
</tbody>
</table>

\(^{1}\)http://en.wikipedia.org/wiki/Amdahls_law
5.3. Speedup and Efficiency

Figure 5.2: Speedup for executions with different node counts.

Figure 5.3: Efficiency for executions with different node counts.
Figure 5.4: Predicted speedup for an increased number of nodes.

The results indicate that a speedup of just under four times can be expected for a cluster of eight nodes and that the efficiency of each node is about 40 to 50%. The speedup prediction shows a drastic deterioration in efficiency as the node count increases. However, many factors impact the speedup results, e.g., partition size, video duration, and load balancing. The results obtained here are limited to a fixed partition size, video duration and load balancing algorithm. Different results may be obtained if any of these factors change.
5.4 Crossover Point

In the interest of answering the second question, execution times for video segments of increasing duration are measured. We are interested in finding the crossover point for when the distributed system outperforms the serial system. Measurements for both the serial and the distributed system are gathered and compared with each other. Video 1 in Table 5.2 is used in the experiments. Videos of 10 to 100 seconds in steps of 10 seconds are captured into segments such that each segment start from the beginning of the original video. The distributed system runs with four nodes and with partition sizes 25, 35 and 45 seconds. The goal is to demonstrate how the different partition sizes affect the performance of the distributed system and when it is more profitable to run the serial system over the distributed system.

The experiment results of analyzing videos of increasing duration can be found in Figure 5.5. The data used for the plot can be found in Table B.7 in Appendix B. The results show that the distributed system outperforms the serial system when it can make use of the computing power offered by the extra nodes. Before the distributed system can take advantage of this, the serial system perform slightly better. The overhead introduced by the distributed system for the different video durations is presented in Table 5.4. Similar to in the speedup measurements, 30 samples were collected for each setup to get an approximate normal distribution of the collected data. The skewness and distribution of the measurements are found in Figures B.6, B.7, B.8, and B.9 and Tables B.8, B.9, B.10, and B.11 in Appendix B.

![Figure 5.5: Average execution time for videos of increasing duration running with partition size 25, 35 and 45 seconds.](image)

At which video duration the distributed system outperforms the serial system is dependent on the partition size in use. In Figure 5.5 and Table 5.5 it is shown that the serial system outperforms the distributed system up until the video duration where it can take
advantage of the parallelization. For example, when using a partition size of 25 s, the distributed system outperforms the serial at a video duration of 30 s; when using a partition size of 35 s: 40 s; when using a partition size of 45 s: 50 s. When the distributed system cannot take advantage of the parallelization, there is a small overhead, < 10 %, see Table 5.4. If the overhead is tolerable for small videos the distributed approach is the better choice. However, if it is not possible to draw any benefit from the parallelization – e.g., if only one node is available in the distributed cluster – more errors and overhead is introduced by using the distributed approach, mostly due to partitioning. The measurement results in Appendix B show a large interquartile range for some video durations and partition sizes, e.g., in Figure B.7. This seems to happen when a sufficient number of video partitions are available to activate another machine. If the machine have been idle or unused it may need some extra time, e.g., to fill up cache, run JVM optimizations etc, before it can operate at full capacity, thus causing deviations in the measurements.

5.5 Throughput

The third question is answered by evaluating the implemented load balancing algorithms. A SLB algorithm of the Round Robin type is implemented as well as a sender initiated DLB algorithm. The DLB algorithm uses a threshold transfer policy, a polling location policy, a simple selection policy that always selects the last queued sub-task, and an information policy that fetches information on-demand, such that information from other nodes are requested when needed. The evaluation of the load balancing algorithms are performed by measuring system throughput. The throughput in this case is the number of seconds of video processed per minute of real time (wall-clock time). All four videos listed in Table 5.2 are used in the evaluation. The videos are queued instantly from one node, and the execution time for when all videos are finished is used to calculate the throughput by using Equation 5.7, where $V_{D_{tot}}$ is the total duration of the videos, and $T$ is the execution time. The evaluation is done using two types of machine clusters (of size four): a homogeneous cluster consisting of $M3$ medium instances only (see Table 5.1); and a heterogeneous cluster consisting of two $M3$ medium instances, one $M3$ large instance, and one $C3$ xlarge instance. Comparisons of the two load balancing implementations are made for both cluster types based on the throughput metric. The load distribution is also be measured to demonstrate the load balancing schema differences.
5.5. Throughput

\[ T = \frac{V_{D_{tot}}}{T} \]  

(5.7)

Throughput measurements for the two load balancing algorithms for homogeneous and a heterogeneous clusters are presented in Figure 5.6. The experiments showed virtually no difference in throughput for the two load balancing schemes when the system was run on a homogeneous cluster. When running on a heterogeneous cluster, the distributed architecture could take advantage of the faster machines and the throughput was doubled when using the DLB implementation compared to the SLB implementation. The average number of executed sub-tasks per machine in the homogeneous cluster is presented in Figure 5.7. The same kind of data for the heterogeneous cluster is presented in Figure 5.8. Similar to previous results, 30 measurement samples was made per setup, the skewness and distribution of the throughput measurements is found in Table B.12 and Figure B.10 in Appendix B.

![Figure 5.6: Throughput for the two load balancing algorithms.](image)

When the cluster machines have identical performance the SLB algorithm works well because sub-tasks are processed at the same rate. The DLB algorithm also works satisfactorily in this case, it maintains a steady workload across all machines and even shows slightly better performance than the SLB algorithm. It could be expected that the SLB algorithm would outperform the DLB algorithm because of the added synchronization delays, but the algorithms have almost no difference in throughput performance. The test cluster contains a small number of nodes, thus the communication overhead created by the DLB algorithm is small. In a larger cluster the SLB might perform better than the DLB because of increasing amount of communication, but not in this case. When running on a heterogeneous cluster the performance of the SLB algorithm is limited to the machine with the worst performance. As all machines get an approximately equal chunk of the workload, the faster machines finish
Figure 5.7: Load balance during throughput measurements on the homogeneous cluster.

Figure 5.8: Load balance during throughput measurements on the heterogeneous cluster.

before the slower machines, but the final result is not ready until all machines are finished, thus no advantage is taken from the machines with greater performance. This is not the case for the DLB algorithm, as most work is put on the machines with greater performance.
than the others. This is shown by the load distribution in Figure 5.8. The DLB algorithm is clearly the better choice when running on a small heterogeneous cluster. Considering the heterogeneity of desktop computers in a standard office the better choice would be the DLB algorithm.
Chapter 6

Conclusions

This work demonstrates that it is possible to increase efficiency for VCA by using a distributed architecture. The performance evaluation of the architecture shows that a speedup just under four times can be expected for a cluster of eight nodes. It is concluded that the efficiency of each node is about 40 to 50% as long as there are sufficient work available up to a node count of 8. The predicted speedup, calculated using Amdahl’s law, shows a drastic deterioration in efficiency as the node count is increased beyond 8. The evaluation shows that the video duration required for the distributed system to outperform the serial depends on the partition size in use. The performance gains enabled by the distributed system can be achieved as soon as the workload can be parallelized. However, the results also show that small partition sizes introduce more errors. A slight overhead can be expected before the workload can be parallelized. The load balancing results shows that the DLB algorithm outperforms the SLB algorithm when the distributed system runs on a heterogeneous cluster of computers. If a homogeneous cluster of computers is used, the performance differences are negligible for a small cluster. Different results can be expected when using a larger number of nodes.

It may be possible to further increase efficiency through improvements on e.g., load balancing, partitioning, and analysis. The architecture uses tools which are not adapted for the distributed purpose. Adaptation to the analysis plug-ins to better fit the distributed case may increase performance. As mentioned in Chapter 5, the ASR plug-in trains its speech model on the beginning on the input video. This becomes wasteful in the distributed case, as training is performed for each partition. If the plug-in is altered such that training is only performed once, per node and VCA task, it is possible to save time.

6.1 Restrictions

Originally the plan was to test more variable setups in the performance evaluation. Executing ASR analysis is time consuming, and the time available for testing was limited, thus restrictions had to be made in order to finish on time. There was also a plan to use VAD to decrease the error rate caused by the partitioning. Limited implementation attempts were made, but the results were not satisfactory. As the project progressed, focus was put on other areas of the project. If more time would have been available, more research may have been put into this area. With limited time available, the evaluation was limited to two load balancing algorithms. There are a lot of different load balancing algorithms, and it would have been interesting to evaluate a broader variety.
6.2 Limitations

The distributed VCA system and the thesis is limited to ASR. However, the architecture is made to make it easy to add support for other kinds of analysis. Essentially, the architecture works as an overlay to the VT that enables distribution of workload, and all VCA supported by the VT is easy to make available in the architecture as well. The partitioning part of the architecture have limited support for video and audio codecs. The partitioner relies on the codec support enabled in the compilation of FFmpeg. As a consequence, some videos may not be valid input to the distributed application, while they are valid input to the VT. The failover enabled by the distributed architecture is also limited. The master node – the node starting a VCA – is a single point of failure. If the master is not available, no cleanup is made by the other nodes, e.g., sub-tasks queued by the failed node is still executed by the workers even though the recipient of the result is dead. This can be improved in several directions. The performance tests in the evaluation are limited to the use of TED talk videos, which covers only a limited set of possible inputs. Consequently, the results and conclusions are mainly applicable to similar data.

6.3 Future Work

Certain improvements are needed before the system can be used in a production environment. A better interface option is necessary if the system is to be used for anything other than a proof of concept, like in this thesis project. Currently the system provides a simple command-line interface with limited operations. Future work may include a programmable API, e.g., REST, that can be used to operate the nodes in the system. This would provide more possibilities for integration with other applications. Because the application runs on standard desktop computers in an office connected via a local area network, no concern is taken when it comes to security. All communication between the nodes are done in plain text, and limited sanity checks are performed on the data sent between nodes. Improvements on e.g., authentication, data integrity and encryption may be included in future work. Stress tests on the system have been performed to a limited extent, more tests are necessary to ensure that the system can stay alive during hard workloads and longer periods of time.

There is also room for improvements in partitioning. Future work may include support for more media formats and encodings. The current system is limited to the formats and encodings supported and compiled with FFmpeg. Another partitioning feature that may be a part of future work is an adaptive partitioner, that sets an appropriate partitioning size depending on the number of available and healthy nodes. This may improve both latency and throughput of the system. Another improvement, specific to the ASR case, is to implement a partitioner that cuts video only on non-speech to reduce errors caused by partitioning. Other load balancing algorithms may be included in future work to further improve system throughput.

Currently the system supports one kind of VCA, namely ASR. In future work, support for more kinds of VCA may be included, e.g., face recognition, object tracking, and dynamic masking. Support for face recognition and object tracking is fairly easy to add because the VT already supports it. Possibilities to expand the application to the cloud would be possible. The application may benefit from automatic and elastic scaling offered by IaaS providers. Future work may include extensions that allow the distributed application to scale up and down depending on system workload for example. One possibility may be to use available standard office computers and cloud burst extra nodes if the system is under heavy load.
Chapter 7

Acknowledgements

I would like to express my gratitude to my supervisor Johan Tordsson at the department of Computing Science, Umeå University who gave me feedback and supported me throughout the work of this thesis. I would also like to thank my supervisor at Codemill, Urban Söderberg for assisting me, taking the time to answer my questions, and discussing ideas with me. I also thank Rikard Lönneborg and Johanna Björklund at CodeMill who came up with the idea for the thesis and wrote the initial specification of the project. Finally, I thank my fiancée, Sofie Linnér, for having read the report and given me feedback.
References


Appendix A

Installation and Configuration

The system requires some setup before it can be used properly and for all functionality to work. The following list of dependencies must be installed before the application is installed:

1. **Java Runtime Environment**
   The distributed application is a Java program thus a *Java Runtime Environment* (JRE) is required to run the application. In particular a JRE for Java 7 is required. Installation instructions are available on the Java homepage [12].

2. **Apache Maven**
   The project makes use of the project management tool *Apache Maven* (Maven) to compile and install the application. Installation and usage instructions for Maven are available on the Maven homepage [5].

3. **Apache Zookeeper**
   Core functionality of the implemented distributed application requires a running Zookeeper instance. Zookeeper is a centralized service and can be installed on a single server (for development purposes) or on a cluster of machines (recommended for production use) called an ensemble. Instructions on how to install a Zookeeper ensemble can be found in the administrators guide on the Zookeeper homepage [33].

4. **Vidispine Transcoder**
   In order to run the actual VCA analysis the application requires an instance of the VT to be running. It is recommended to run one instance of the on for each node. Installation instructions for the VT is available to customers of Vidispine AB and their partners on the partner portal.

5. **CodeMill Speech-To-Text plug-in**
   In order to perform ASR analysis the CSTT plug-in must be enabled for the VT. Installation instructions for the plug-in is available to customers of CodeMill AB and Vidispine AB.

The distributed application is compiled and installed using Maven. Based on the *Project Object Model* (POM) for the project, Maven can manage, build and install the distributed application. First, the application need to be compiled and packaged. In the root of the project run the command:
$ mvn package

This will compile and package the application in a executable jar file in the \texttt{target} directory.

Before running the application some configuration need to be performed. The application will try to read the configuration file located in the current working directory. The format is XML and the file must be named \texttt{config.xml}. An example configuration can be found in Listings A.1, explanations of the different configuration options are explained Table A.1. The configuration can also be created using the configuration script in the root of the project:

$ python config.py -h

The configuration file must be placed in the directory from which the application is started and adhere to the configuration format in Listing A.1. When the configuration is completed, the application is started by running the jar file:

$ java -jar DVCA-1.2.jar

This starts the application running the interactive command line interface. To start the application in non-interactive mode, add the flag \texttt{-non-interactive}:

$ java -jar DVCA-1.2.jar -non-interactive

This is useful if the application is going to be used as a Worker-only node.
Listing A.1: Configuration Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <zookeeper>
    <sessionTimeout>15000</sessionTimeout>
    <servers>
      <server>
        <host>zookeeper1.host.se</host>
        <port>2181</port>
      </server>
      <server>
        <host>zookeeper2.host.se</host>
        <port>2181</port>
      </server>
      <server>
        <host>zookeeper3.host.se</host>
        <port>2181</port>
      </server>
    </servers>
  </zookeeper>
  <transcoder>
    <server>
      <host>localhost</host>
      <port>8888</port>
    </server>
    <secure>false</secure>
  </transcoder>
  <rmi>
    <server>
      <host>10.12.10.65</host>
      <port>1099</port>
    </server>
  </rmi>
  <partitioner>
    <targetDuration>25.0</targetDuration>
  </partitioner>
  <numWorkers>2</numWorkers>
  <groupName>default</groupName>
  <httpdPort>8080</httpdPort>
</configuration>
```
Table A.1: Descriptions of the configuration options.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;configuration&gt;</code></td>
<td>Base tag, all other configuration options are placed inside this tag.</td>
</tr>
<tr>
<td><code>&lt;groupName&gt;</code></td>
<td>The name of the node group to join. Nodes with the same <code>groupName</code> setting is visible to each other.</td>
</tr>
<tr>
<td><code>&lt;httpd&gt;</code></td>
<td>The port to use for the internal HTTP server, e.g. used to fetch metadata from the transcoder.</td>
</tr>
<tr>
<td><code>&lt;numWorkers&gt;</code></td>
<td>The number of threads to use for processing sub-tasks.</td>
</tr>
<tr>
<td><code>&lt;partitioner&gt;</code></td>
<td>Partitioner specific settings are placed inside this tag.</td>
</tr>
<tr>
<td><code>&lt;rmi&gt;</code></td>
<td>RMI connection settings for P2P communications. Contains a single server tag with hostname and port that will be used to listen for P2P communications.</td>
</tr>
<tr>
<td><code>&lt;server&gt;</code></td>
<td>Contains the host and port configurations for a server. This tag occurs on several locations.</td>
</tr>
<tr>
<td><code>&lt;targetDuration&gt;</code></td>
<td>Partitioner specific setting. This sets the target partition duration length when partitioning video files. When partitioning is performed key frame locations must be considered in order to keep each partition decodable, hence this is an approximate length of the partitions.</td>
</tr>
<tr>
<td><code>&lt;transcoder&gt;</code></td>
<td>VT specific settings are placed inside this tag.</td>
</tr>
<tr>
<td><code>&lt;secure&gt;</code></td>
<td>VT specific setting. Determines whether HTTP or HTTPS is used when connecting to the transcoder.</td>
</tr>
<tr>
<td><code>&lt;zookeeper&gt;</code></td>
<td>Zookeeper specific settings are configured inside this tag.</td>
</tr>
<tr>
<td><code>&lt;sessionTimeout&gt;</code></td>
<td>Zookeeper specific setting. Defines the default timeout before the Zookeeper session is considered lost.</td>
</tr>
<tr>
<td><code>&lt;servers&gt;</code></td>
<td>Zookeeper specific setting. Contains connection information for the Zookeeper ensemble/cluster. It is recommended to have at least 3 servers in the Zookeeper ensemble/cluster.</td>
</tr>
</tbody>
</table>
Appendix B

Additional Result Data

B.1 Speech Recognition Accuracy

This section contains the additional test data for the WER and WAR for different partition sizes for each test video.

Table B.1: Average WER and WAR for different partition sizes for all four videos.

<table>
<thead>
<tr>
<th>Partition size [s]</th>
<th>WER [%]</th>
<th>WAR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>43.1</td>
<td>56.9</td>
</tr>
<tr>
<td>5</td>
<td>48.3</td>
<td>51.7</td>
</tr>
<tr>
<td>10</td>
<td>46.5</td>
<td>53.5</td>
</tr>
<tr>
<td>15</td>
<td>46.2</td>
<td>53.8</td>
</tr>
<tr>
<td>20</td>
<td>45.5</td>
<td>54.5</td>
</tr>
<tr>
<td>25</td>
<td>44.5</td>
<td>55.5</td>
</tr>
<tr>
<td>30</td>
<td>44.6</td>
<td>55.4</td>
</tr>
<tr>
<td>35</td>
<td>44.7</td>
<td>55.3</td>
</tr>
<tr>
<td>40</td>
<td>44.2</td>
<td>55.8</td>
</tr>
<tr>
<td>45</td>
<td>43.9</td>
<td>56.1</td>
</tr>
<tr>
<td>50</td>
<td>44.7</td>
<td>55.3</td>
</tr>
<tr>
<td>55</td>
<td>43.9</td>
<td>56.1</td>
</tr>
<tr>
<td>60</td>
<td>43.8</td>
<td>56.2</td>
</tr>
<tr>
<td>90</td>
<td>44.3</td>
<td>55.7</td>
</tr>
<tr>
<td>120</td>
<td>44.4</td>
<td>55.6</td>
</tr>
<tr>
<td>150</td>
<td>43.5</td>
<td>56.5</td>
</tr>
<tr>
<td>180</td>
<td>43.8</td>
<td>56.2</td>
</tr>
<tr>
<td>210</td>
<td>43.1</td>
<td>56.9</td>
</tr>
<tr>
<td>240</td>
<td>43.6</td>
<td>56.4</td>
</tr>
<tr>
<td>270</td>
<td>43.7</td>
<td>56.3</td>
</tr>
<tr>
<td>300</td>
<td>43.9</td>
<td>56.1</td>
</tr>
</tbody>
</table>
Figure B.1: Video 1 (AJJacobs_2014A-480p.mp4): WER and WAR for different partition sizes.

Figure B.2: Video 2 (JorgeSoto_2014G-480p.mp4): WER and WAR for different partition sizes.
Table B.2: Video 1 (AJJacobs_2014A-480p.mp4): WER and WAR for different partition sizes.

<table>
<thead>
<tr>
<th>Partition size [s]</th>
<th>WER [%]</th>
<th>WAR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>30.4</td>
<td>69.6</td>
</tr>
<tr>
<td>5</td>
<td>34.9</td>
<td>65.1</td>
</tr>
<tr>
<td>10</td>
<td>32.4</td>
<td>67.6</td>
</tr>
<tr>
<td>15</td>
<td>32.8</td>
<td>67.2</td>
</tr>
<tr>
<td>20</td>
<td>32.5</td>
<td>67.5</td>
</tr>
<tr>
<td>25</td>
<td>30.1</td>
<td>69.9</td>
</tr>
<tr>
<td>30</td>
<td>31.2</td>
<td>68.8</td>
</tr>
<tr>
<td>35</td>
<td>30.4</td>
<td>69.6</td>
</tr>
<tr>
<td>40</td>
<td>30.9</td>
<td>69.1</td>
</tr>
<tr>
<td>45</td>
<td>30.4</td>
<td>69.6</td>
</tr>
<tr>
<td>50</td>
<td>30.6</td>
<td>69.4</td>
</tr>
<tr>
<td>55</td>
<td>30.9</td>
<td>69.1</td>
</tr>
<tr>
<td>60</td>
<td>30.6</td>
<td>69.4</td>
</tr>
<tr>
<td>90</td>
<td>30.6</td>
<td>69.4</td>
</tr>
<tr>
<td>120</td>
<td>29.9</td>
<td>70.1</td>
</tr>
<tr>
<td>150</td>
<td>30.2</td>
<td>69.8</td>
</tr>
<tr>
<td>180</td>
<td>31.4</td>
<td>68.6</td>
</tr>
<tr>
<td>210</td>
<td>30.4</td>
<td>69.6</td>
</tr>
<tr>
<td>240</td>
<td>30.3</td>
<td>69.7</td>
</tr>
<tr>
<td>270</td>
<td>30.5</td>
<td>69.5</td>
</tr>
<tr>
<td>300</td>
<td>30.1</td>
<td>69.9</td>
</tr>
</tbody>
</table>

Figure B.3: Video 3 (KareAnderson_2014S-480p.mp4): WER and WAR for different partition sizes.

<table>
<thead>
<tr>
<th>Partition size [s]</th>
<th>WER [%]</th>
<th>WAR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>58.9</td>
<td>41.1</td>
</tr>
<tr>
<td>5</td>
<td>63.5</td>
<td>36.5</td>
</tr>
<tr>
<td>10</td>
<td>59.6</td>
<td>40.4</td>
</tr>
<tr>
<td>15</td>
<td>62.0</td>
<td>38.0</td>
</tr>
<tr>
<td>20</td>
<td>60.5</td>
<td>39.5</td>
</tr>
<tr>
<td>25</td>
<td>60.4</td>
<td>39.6</td>
</tr>
<tr>
<td>30</td>
<td>59.8</td>
<td>40.2</td>
</tr>
<tr>
<td>35</td>
<td>60.1</td>
<td>39.9</td>
</tr>
<tr>
<td>40</td>
<td>60.8</td>
<td>39.2</td>
</tr>
<tr>
<td>45</td>
<td>60.4</td>
<td>39.6</td>
</tr>
<tr>
<td>50</td>
<td>59.2</td>
<td>40.8</td>
</tr>
<tr>
<td>55</td>
<td>58.6</td>
<td>41.4</td>
</tr>
<tr>
<td>60</td>
<td>58.9</td>
<td>41.1</td>
</tr>
<tr>
<td>90</td>
<td>60.1</td>
<td>39.9</td>
</tr>
<tr>
<td>120</td>
<td>62.1</td>
<td>37.9</td>
</tr>
<tr>
<td>150</td>
<td>59.2</td>
<td>40.8</td>
</tr>
<tr>
<td>180</td>
<td>59.6</td>
<td>40.4</td>
</tr>
<tr>
<td>210</td>
<td>59.1</td>
<td>40.9</td>
</tr>
<tr>
<td>240</td>
<td>59.9</td>
<td>40.1</td>
</tr>
<tr>
<td>270</td>
<td>60.9</td>
<td>39.1</td>
</tr>
<tr>
<td>300</td>
<td>61.4</td>
<td>38.6</td>
</tr>
</tbody>
</table>

Figure B.4: Video 4 (SusanEtlinger_2014S-480p.mp4): WER and WAR for different partition sizes.
Table B.4: Video 3 (KareAnderson_2014S-480p.mp4): WER and WAR for different partition sizes.

<table>
<thead>
<tr>
<th>Partition size [s]</th>
<th>WER [%]</th>
<th>WAR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>41.4</td>
<td>58.6</td>
</tr>
<tr>
<td>5</td>
<td>47.8</td>
<td>52.2</td>
</tr>
<tr>
<td>10</td>
<td>48.5</td>
<td>51.5</td>
</tr>
<tr>
<td>15</td>
<td>45.2</td>
<td>54.8</td>
</tr>
<tr>
<td>20</td>
<td>44.8</td>
<td>55.2</td>
</tr>
<tr>
<td>25</td>
<td>43.2</td>
<td>56.8</td>
</tr>
<tr>
<td>30</td>
<td>44.8</td>
<td>55.2</td>
</tr>
<tr>
<td>35</td>
<td>44.2</td>
<td>55.8</td>
</tr>
<tr>
<td>40</td>
<td>43.8</td>
<td>56.2</td>
</tr>
<tr>
<td>45</td>
<td>41.9</td>
<td>58.1</td>
</tr>
<tr>
<td>50</td>
<td>45.0</td>
<td>55.0</td>
</tr>
<tr>
<td>55</td>
<td>43.3</td>
<td>56.7</td>
</tr>
<tr>
<td>60</td>
<td>43.2</td>
<td>56.8</td>
</tr>
<tr>
<td>90</td>
<td>43.3</td>
<td>56.7</td>
</tr>
<tr>
<td>120</td>
<td>42.2</td>
<td>57.8</td>
</tr>
<tr>
<td>150</td>
<td>42.5</td>
<td>57.5</td>
</tr>
<tr>
<td>180</td>
<td>42.2</td>
<td>57.8</td>
</tr>
<tr>
<td>210</td>
<td>41.2</td>
<td>58.8</td>
</tr>
<tr>
<td>240</td>
<td>41.9</td>
<td>58.1</td>
</tr>
<tr>
<td>270</td>
<td>41.3</td>
<td>58.7</td>
</tr>
<tr>
<td>300</td>
<td>42.3</td>
<td>57.7</td>
</tr>
</tbody>
</table>
Table B.5: Video 4 (SusanEtlinger_2014S-480p.mp4): WER and WAR for different partition sizes.

<table>
<thead>
<tr>
<th>Partition size [s]</th>
<th>WER [%]</th>
<th>WAR [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>41.6</td>
<td>58.4</td>
</tr>
<tr>
<td>5</td>
<td>46.9</td>
<td>53.1</td>
</tr>
<tr>
<td>10</td>
<td>45.6</td>
<td>54.4</td>
</tr>
<tr>
<td>15</td>
<td>44.7</td>
<td>55.3</td>
</tr>
<tr>
<td>20</td>
<td>44.1</td>
<td>55.9</td>
</tr>
<tr>
<td>25</td>
<td>44.2</td>
<td>55.8</td>
</tr>
<tr>
<td>30</td>
<td>42.8</td>
<td>57.2</td>
</tr>
<tr>
<td>35</td>
<td>43.9</td>
<td>56.1</td>
</tr>
<tr>
<td>40</td>
<td>41.4</td>
<td>58.6</td>
</tr>
<tr>
<td>45</td>
<td>43.0</td>
<td>57.0</td>
</tr>
<tr>
<td>50</td>
<td>44.0</td>
<td>56.0</td>
</tr>
<tr>
<td>55</td>
<td>42.9</td>
<td>57.1</td>
</tr>
<tr>
<td>60</td>
<td>42.5</td>
<td>57.5</td>
</tr>
<tr>
<td>90</td>
<td>43.0</td>
<td>57.0</td>
</tr>
<tr>
<td>120</td>
<td>43.3</td>
<td>56.7</td>
</tr>
<tr>
<td>150</td>
<td>42.1</td>
<td>57.9</td>
</tr>
<tr>
<td>180</td>
<td>42.1</td>
<td>57.9</td>
</tr>
<tr>
<td>210</td>
<td>41.7</td>
<td>58.3</td>
</tr>
<tr>
<td>240</td>
<td>42.5</td>
<td>57.5</td>
</tr>
<tr>
<td>270</td>
<td>42.2</td>
<td>57.8</td>
</tr>
<tr>
<td>300</td>
<td>41.9</td>
<td>58.1</td>
</tr>
</tbody>
</table>
B.2 Speedup and Efficiency

This section contains measurement statistics for the speedup tests measuring the execution time for increasing node count.

Table B.6: Wall clock time (in seconds) measurements for execution with different node counts.

<table>
<thead>
<tr>
<th>#</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>1923.05</td>
<td>1893.44</td>
<td>1914.80</td>
<td>1919.45</td>
<td>1929.87</td>
<td>1963.23</td>
<td>15.82</td>
<td>2.94</td>
</tr>
<tr>
<td>1</td>
<td>3653.99</td>
<td>3620.59</td>
<td>3627.04</td>
<td>3654.43</td>
<td>3671.05</td>
<td>3710.21</td>
<td>25.28</td>
<td>4.69</td>
</tr>
<tr>
<td>2</td>
<td>1873.06</td>
<td>1806.59</td>
<td>1844.97</td>
<td>1867.16</td>
<td>1900.64</td>
<td>1950.69</td>
<td>36.92</td>
<td>6.86</td>
</tr>
<tr>
<td>3</td>
<td>1358.70</td>
<td>1229.12</td>
<td>1334.37</td>
<td>1361.75</td>
<td>1398.96</td>
<td>1461.45</td>
<td>51.66</td>
<td>9.59</td>
</tr>
<tr>
<td>4</td>
<td>972.44</td>
<td>930.31</td>
<td>955.66</td>
<td>972.48</td>
<td>984.05</td>
<td>1045.17</td>
<td>27.39</td>
<td>5.09</td>
</tr>
<tr>
<td>6</td>
<td>734.25</td>
<td>705.28</td>
<td>716.92</td>
<td>733.13</td>
<td>747.20</td>
<td>778.93</td>
<td>21.58</td>
<td>4.01</td>
</tr>
<tr>
<td>8</td>
<td>509.91</td>
<td>479.98</td>
<td>499.32</td>
<td>507.36</td>
<td>520.46</td>
<td>542.63</td>
<td>16.34</td>
<td>3.04</td>
</tr>
<tr>
<td>10</td>
<td>510.65</td>
<td>470.47</td>
<td>497.01</td>
<td>504.10</td>
<td>526.42</td>
<td>563.00</td>
<td>21.49</td>
<td>3.99</td>
</tr>
</tbody>
</table>

Figure B.5: Wall clock time (in seconds) measurements for execution with different node counts.
B.3 Crossover Point

This section contains measurement statistics for the crossover tests measuring the execution time for videos of increasing length.

Table B.7: Average execution time for videos of increasing duration running with partition size 25, 35 and 45 seconds.

<table>
<thead>
<tr>
<th>Video duration</th>
<th>Exec. time [s]</th>
<th>Exec. time [s]</th>
<th>Exec. time [s]</th>
<th>Exec. time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial</td>
<td>Distributed 25s</td>
<td>Distributed 35s</td>
<td>Distributed 45s</td>
</tr>
<tr>
<td>10</td>
<td>52.12</td>
<td>56.83</td>
<td>55.48</td>
<td>56.06</td>
</tr>
<tr>
<td>20</td>
<td>104.34</td>
<td>107.79</td>
<td>107.21</td>
<td>108.23</td>
</tr>
<tr>
<td>30</td>
<td>156.59</td>
<td>138.88</td>
<td>160.00</td>
<td>162.19</td>
</tr>
<tr>
<td>40</td>
<td>208.02</td>
<td>139.15</td>
<td>189.36</td>
<td>211.85</td>
</tr>
<tr>
<td>50</td>
<td>275.07</td>
<td>156.58</td>
<td>189.02</td>
<td>247.39</td>
</tr>
<tr>
<td>60</td>
<td>331.72</td>
<td>215.12</td>
<td>189.75</td>
<td>247.99</td>
</tr>
<tr>
<td>70</td>
<td>376.42</td>
<td>221.82</td>
<td>203.87</td>
<td>248.70</td>
</tr>
<tr>
<td>80</td>
<td>406.27</td>
<td>219.96</td>
<td>224.86</td>
<td>252.33</td>
</tr>
<tr>
<td>90</td>
<td>427.08</td>
<td>257.55</td>
<td>224.54</td>
<td>252.92</td>
</tr>
<tr>
<td>100</td>
<td>453.67</td>
<td>292.66</td>
<td>224.60</td>
<td>341.24</td>
</tr>
</tbody>
</table>

Table B.8: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running the current serial system.

<table>
<thead>
<tr>
<th>Length [s]</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>52.12</td>
<td>51.06</td>
<td>51.45</td>
<td>52.38</td>
<td>52.61</td>
<td>53.25</td>
<td>0.67</td>
<td>0.12</td>
</tr>
<tr>
<td>20</td>
<td>104.34</td>
<td>102.50</td>
<td>102.96</td>
<td>104.52</td>
<td>105.51</td>
<td>106.61</td>
<td>1.34</td>
<td>0.24</td>
</tr>
<tr>
<td>30</td>
<td>156.59</td>
<td>154.80</td>
<td>155.69</td>
<td>155.89</td>
<td>156.90</td>
<td>160.48</td>
<td>1.54</td>
<td>0.29</td>
</tr>
<tr>
<td>40</td>
<td>208.02</td>
<td>205.20</td>
<td>205.81</td>
<td>207.64</td>
<td>211.14</td>
<td>211.96</td>
<td>2.97</td>
<td>0.54</td>
</tr>
<tr>
<td>50</td>
<td>275.07</td>
<td>273.94</td>
<td>275.67</td>
<td>276.54</td>
<td>279.03</td>
<td>2.86</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>331.72</td>
<td>323.63</td>
<td>331.32</td>
<td>331.60</td>
<td>332.24</td>
<td>335.07</td>
<td>1.94</td>
<td>0.35</td>
</tr>
<tr>
<td>70</td>
<td>376.42</td>
<td>375.62</td>
<td>376.05</td>
<td>377.79</td>
<td>379.07</td>
<td>2.21</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>406.27</td>
<td>405.56</td>
<td>406.33</td>
<td>407.00</td>
<td>410.48</td>
<td>1.72</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>427.08</td>
<td>420.63</td>
<td>429.77</td>
<td>431.30</td>
<td>448.46</td>
<td>7.03</td>
<td>1.28</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>453.67</td>
<td>448.89</td>
<td>453.62</td>
<td>459.66</td>
<td>461.69</td>
<td>5.42</td>
<td>0.99</td>
<td></td>
</tr>
</tbody>
</table>

Table B.9: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 25.

<table>
<thead>
<tr>
<th>Length [s]</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>56.83</td>
<td>54.71</td>
<td>55.88</td>
<td>56.53</td>
<td>57.59</td>
<td>60.52</td>
<td>1.57</td>
<td>0.29</td>
</tr>
<tr>
<td>20</td>
<td>107.79</td>
<td>104.15</td>
<td>106.52</td>
<td>107.38</td>
<td>108.53</td>
<td>112.61</td>
<td>2.10</td>
<td>0.38</td>
</tr>
<tr>
<td>30</td>
<td>138.88</td>
<td>134.53</td>
<td>137.82</td>
<td>138.80</td>
<td>139.89</td>
<td>145.98</td>
<td>2.34</td>
<td>0.43</td>
</tr>
<tr>
<td>40</td>
<td>139.15</td>
<td>136.30</td>
<td>137.78</td>
<td>138.70</td>
<td>140.65</td>
<td>144.40</td>
<td>2.11</td>
<td>0.38</td>
</tr>
<tr>
<td>50</td>
<td>156.58</td>
<td>153.38</td>
<td>155.75</td>
<td>156.40</td>
<td>157.20</td>
<td>160.81</td>
<td>1.60</td>
<td>0.29</td>
</tr>
<tr>
<td>60</td>
<td>215.32</td>
<td>209.41</td>
<td>214.40</td>
<td>186.06</td>
<td>218.79</td>
<td>322.94</td>
<td>54.99</td>
<td>10.04</td>
</tr>
<tr>
<td>70</td>
<td>221.82</td>
<td>181.70</td>
<td>183.70</td>
<td>186.25</td>
<td>265.83</td>
<td>325.97</td>
<td>52.59</td>
<td>9.60</td>
</tr>
<tr>
<td>80</td>
<td>219.96</td>
<td>179.60</td>
<td>183.18</td>
<td>185.81</td>
<td>266.30</td>
<td>343.07</td>
<td>61.29</td>
<td>11.19</td>
</tr>
<tr>
<td>90</td>
<td>257.55</td>
<td>182.21</td>
<td>215.38</td>
<td>272.45</td>
<td>310.45</td>
<td>340.97</td>
<td>51.65</td>
<td>9.43</td>
</tr>
<tr>
<td>100</td>
<td>292.66</td>
<td>181.65</td>
<td>282.50</td>
<td>307.72</td>
<td>320.60</td>
<td>348.72</td>
<td>49.81</td>
<td>9.09</td>
</tr>
</tbody>
</table>
Figure B.6: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running the current serial system.

Figure B.7: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 25.
Table B.10: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 35.

<table>
<thead>
<tr>
<th>Length [s]</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>55.48</td>
<td>54.49</td>
<td>54.63</td>
<td>55.46</td>
<td>55.72</td>
<td>57.69</td>
<td>0.98</td>
<td>0.18</td>
</tr>
<tr>
<td>20</td>
<td>107.21</td>
<td>105.89</td>
<td>106.08</td>
<td>106.94</td>
<td>108.19</td>
<td>109.45</td>
<td>1.18</td>
<td>0.22</td>
</tr>
<tr>
<td>30</td>
<td>160.00</td>
<td>155.15</td>
<td>158.03</td>
<td>159.13</td>
<td>161.38</td>
<td>168.17</td>
<td>3.15</td>
<td>0.57</td>
</tr>
<tr>
<td>40</td>
<td>189.36</td>
<td>185.64</td>
<td>187.43</td>
<td>188.81</td>
<td>191.01</td>
<td>197.84</td>
<td>2.96</td>
<td>0.54</td>
</tr>
<tr>
<td>50</td>
<td>189.62</td>
<td>185.53</td>
<td>187.58</td>
<td>189.40</td>
<td>190.13</td>
<td>192.52</td>
<td>1.86</td>
<td>0.34</td>
</tr>
<tr>
<td>60</td>
<td>189.75</td>
<td>185.42</td>
<td>188.23</td>
<td>189.52</td>
<td>191.29</td>
<td>196.52</td>
<td>2.37</td>
<td>0.43</td>
</tr>
<tr>
<td>70</td>
<td>263.87</td>
<td>200.67</td>
<td>202.26</td>
<td>203.70</td>
<td>204.49</td>
<td>212.43</td>
<td>2.37</td>
<td>0.43</td>
</tr>
<tr>
<td>80</td>
<td>224.96</td>
<td>220.91</td>
<td>223.30</td>
<td>224.49</td>
<td>225.90</td>
<td>231.59</td>
<td>2.27</td>
<td>0.41</td>
</tr>
<tr>
<td>90</td>
<td>224.54</td>
<td>218.15</td>
<td>223.35</td>
<td>224.79</td>
<td>226.26</td>
<td>227.81</td>
<td>2.22</td>
<td>0.40</td>
</tr>
<tr>
<td>100</td>
<td>224.60</td>
<td>217.53</td>
<td>222.45</td>
<td>224.19</td>
<td>226.20</td>
<td>232.40</td>
<td>2.90</td>
<td>0.53</td>
</tr>
</tbody>
</table>

Figure B.8: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 35.

Table B.11: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 45.

<table>
<thead>
<tr>
<th>Length [s]</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>56.06</td>
<td>53.90</td>
<td>55.67</td>
<td>55.84</td>
<td>56.71</td>
<td>59.70</td>
<td>1.02</td>
<td>0.19</td>
</tr>
<tr>
<td>20</td>
<td>108.23</td>
<td>105.45</td>
<td>106.40</td>
<td>107.31</td>
<td>108.27</td>
<td>115.49</td>
<td>2.66</td>
<td>0.49</td>
</tr>
<tr>
<td>30</td>
<td>162.19</td>
<td>156.35</td>
<td>158.77</td>
<td>159.54</td>
<td>161.91</td>
<td>176.20</td>
<td>6.13</td>
<td>1.12</td>
</tr>
<tr>
<td>40</td>
<td>211.85</td>
<td>204.93</td>
<td>207.29</td>
<td>208.45</td>
<td>211.37</td>
<td>226.43</td>
<td>7.48</td>
<td>1.37</td>
</tr>
<tr>
<td>50</td>
<td>247.39</td>
<td>241.21</td>
<td>246.02</td>
<td>247.15</td>
<td>248.81</td>
<td>252.17</td>
<td>2.69</td>
<td>0.49</td>
</tr>
<tr>
<td>60</td>
<td>247.99</td>
<td>241.70</td>
<td>246.75</td>
<td>248.06</td>
<td>249.06</td>
<td>254.06</td>
<td>2.87</td>
<td>0.52</td>
</tr>
<tr>
<td>70</td>
<td>248.70</td>
<td>240.89</td>
<td>247.11</td>
<td>248.20</td>
<td>250.74</td>
<td>255.48</td>
<td>3.16</td>
<td>0.58</td>
</tr>
<tr>
<td>80</td>
<td>252.33</td>
<td>244.82</td>
<td>249.50</td>
<td>250.80</td>
<td>253.70</td>
<td>273.54</td>
<td>5.60</td>
<td>1.02</td>
</tr>
<tr>
<td>90</td>
<td>272.92</td>
<td>248.73</td>
<td>251.70</td>
<td>272.95</td>
<td>274.36</td>
<td>257.64</td>
<td>2.08</td>
<td>0.38</td>
</tr>
<tr>
<td>100</td>
<td>341.24</td>
<td>276.69</td>
<td>282.82</td>
<td>287.31</td>
<td>333.03</td>
<td>535.37</td>
<td>98.50</td>
<td>17.98</td>
</tr>
</tbody>
</table>
Figure B.9: Wall clock time (in seconds) measurements for executions of video partitions of increasing length running with partition size 45.
B.4 Throughput

This section contains measurement statistics for the throughput measurements.

Table B.12: Throughput measurements for heterogeneous and homogeneous clusters running the two different load balancing implementations.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Mean</th>
<th>Min</th>
<th>25 %</th>
<th>Median</th>
<th>75 %</th>
<th>Max</th>
<th>S.Dev</th>
<th>S.Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous – Static</td>
<td>30.64</td>
<td>29.38</td>
<td>30.28</td>
<td>30.71</td>
<td>30.98</td>
<td>32.14</td>
<td>0.75</td>
<td>0.24</td>
</tr>
<tr>
<td>Homogeneous – Dynamic</td>
<td>32.32</td>
<td>31.61</td>
<td>32.24</td>
<td>32.36</td>
<td>32.47</td>
<td>32.78</td>
<td>0.31</td>
<td>0.10</td>
</tr>
<tr>
<td>Heterogeneous – Static</td>
<td>31.66</td>
<td>29.80</td>
<td>31.10</td>
<td>31.47</td>
<td>32.65</td>
<td>33.05</td>
<td>1.07</td>
<td>0.34</td>
</tr>
<tr>
<td>Heterogeneous – Dynamic</td>
<td>62.94</td>
<td>60.54</td>
<td>61.37</td>
<td>62.83</td>
<td>64.46</td>
<td>65.58</td>
<td>1.75</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Figure B.10: Throughput measurements for heterogeneous and homogeneous clusters running the two different load balancing implementations.
Appendix C

List of Acronyms

ALB
Adaptive Load Balancing. 9

AM
Analysis Module. 26, 30

ASR
Automatic Speech Recognition. 1–3, 5–7, 21, 25–27, 30, 31, 34, 35, 45, 46, 51

AWS
Amazon Web Services. 33, 36

B-frame
Bi-directional coded frame. 16, 18

Cb
Chrominance blue. 15, 18

CLI
Command Line Interface. 31

Cr
Chrominance red. 15, 18

CSTT
CodeMill Speech-To-Text. 25, 30, 51

DLB
Dynamic Load Balancing. 9, 11, 12, 19, 28, 40–43, 45

EC2
Elastic Compute Cloud. 33
FD
   *Face Detection.* 2

FPS
   *Frames per Second.* 14

GMM
   *Gaussian Mixture Model.* 35

GMM
   *Group Management Module.* 26, 29

GOP
   *Group of Pictures.* 18, 19, 27

HVS
   *Human Visual System.* 15

I-frame
   *Intra coded frame.* 16, 18, 27

IDC
   *International Data Corporation.* 1

JNI
   *Java Native Interface.* 26

JRE
   *Java Runtime Environment.* 51

LBM
   *Load Balancing Module.* 26, 28

LD
   *Levenshtein Distance.* 34

MAM
   *Media Asset Management.* 1, 2, 21, 25

Maven
   *Apache Maven.* 51

MPEG
   *Moving Picture Experts Group.* 16, 18
P-frame
   *Predictive coded frame.* 16, 18

PM
   *Partitioning Module.* 26, 27

POM
   *Project Object Model.* 51

PPM
   *Post Processing Module.* 26, 31

RGB
   *Red Green Blue.* 15

SBM
   *Sub-Task Builder Module.* 26, 27

SLB
   *Static Load Balancing.* 9–11, 19, 40, 41, 45

TMM
   *Task Monitoring Module.* 26–28

VAD
   *Voice Activity Detection.* 6, 27, 45

VCA

VT
   *Vidispine Transcoder.* 21, 25, 26, 30, 46, 51, 54

WAR
   *Word Accuracy Rate.* 34, 55

WER
   *Word Error Rate.* 34, 35, 55

WHM
   *Work Handling Module.* 26, 30

Y
   *Luminance.* 15, 18

YCbCr
   *Luminance, Chrominance blue, Chrominance red.* 15

Zookeeper
   *Apache Zookeeper.* 25, 26, 28, 29, 51, 54