TUCS

Tewodros Deneke

Proactive Management of Video Transcoding Services

Turku Centre *for* Computer Science

TUCS Dissertations
No 222, May 2017

Proactive Management of Video Transcoding Services

Tewodros Deneke

To be presented, with the permission of the Faculty of Science and Engineering of the Åbo Akademi University, for public criticism in Auditorium XX on May 4, 2017, at 12 noon.

Åbo Akademi University
Faculty of Science and Engineering
LT1, Universitetsbacken, 20520 Åbo, Finland

2017

Supervisors

Docent Sébastien Lafond
Åbo Akademi University
Faculty of Science and Engineering
LT1, Universitetsbacken, 20520 Åbo
Finland

Reviewers

Professor Jean-François Nezan
Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes - CS 70839
35708 Rennes Cedex 7
France

Assistant Professor Jarno Vanne
Department of Pervasive Computing
Tampere University of Technology
Korkeakoulunkatu 10, 33720 Tampere
Finland

Opponent

Professor Jean-François Nezan
Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes - CS 70839
35708 Rennes Cedex 7
France

ISBN 978-952-12-3536-8
ISSN 1239-1883

Abstract

The consumption of digital video has tremendously increased in the last decade due to advances in information technology, computational capability and communication networks. Following that, video processing applications such as video encoding, transmuxing and transcoding are increasingly being deployed as a service in cloud-based environments. However, these services exhibit high variations in required computational power depending on input video characteristics, processing parameters and request arrival rate patterns. Such differences critically affect the efficient management of computing resources and in turn the running cost of these services. Also, since cloud and distributed computing infrastructures are made up of modern multicore processors, multimedia processing services running on top of them need to be parallelized to enhance overall system utilization further. The primary focus of this thesis is thus the proactive management of multimedia processing services and their parallelization. A solution has been sought to these problems in the context of video transcoding services.

To obtain enough information on the characteristics of real world videos and their transcoding parameters needed to model transcoding workload, a video characteristics dataset was built using data collected from a large video-on-demand system, YouTube. The dataset contains a million randomly sampled video instances listing various fundamental video features. Analysis of the dataset provides insightful statistics on the main online video characteristics that can be further exploited to optimize or model components of a multimedia processing systems. Driven by the insight gained from our datasets an approach to predict transcoding workload of a video given transcoding parameters was proposed. The approach treats video transcoding time as a target variable and predicts it using a model learned from past observations. These observations are divided into training and validation sets while learning the prediction models. The method predicts the transcoding time as a function of several parameters of the input and output video streams. The effectiveness of the method is shown via comparing the resulting predictions with the actual transcoding times on unseen video streams. A squared correlation (predicted vs. measured) of 0.958 has been achieved by the best prediction model on 20-second video segments. Furthermore, simulation results show that the proposed prediction method enables significantly better resource management approaches. More specifically, a proactive load balancing method for transcoding jobs across virtual machines is presented and compared with standard load balancing methods in terms of total system utilization and quality of service. Up to 15% improvement has been achieved in terms of system throughput over the round robin and queue length approaches on a fixed server capacity. Also, a proactive provisioning method that enables provisioning of the right amount of transcoding servers for a given quality of service was presented and evaluated. An Experiment has shown that using proactive provisioning rather than fixed worst case based provisioning saves up to (86%) in terms of virtual machine (VM) hours over the course of a week. Also up to 8% in VM hours has been shown to be saved when augmenting proactive provisioning with proactive load balancing.

In addition to the proactive management of computing resources proper, parallelization of the video transcoding application is important in order to ensure efficient utilization of modern multicore processing units that constitute modern cloud and distributed computing infrastructures. To this end, this thesis provides a solution that allows convenient integration of fine-grained parallel dataflow based implementations of video processing components into existing transcoding frameworks. The primary aim of this proposed solution was to facilitate fine-grained parallelization of video processing components better. Course grained parallelization approaches for video transcoding applications at a group of pictures level and based on a message passing programming model are also presented and evaluated in terms of scalability and throughput. Results show that the proposed parallelization approach scales almost linearly to the number of cores.

Sammanfattning

Konsumtionen av digital video har ökat oerhört under det senaste decenniet som ett resultat av framsteg inom informationsteknologi, beräkningskapacitet och kommunikationsnät. Till följd av detta används videobearbetningsapplikationer såsom videokodning, transmuxning och transkodning alltmer som tjänster i molnbaserade miljöer. Dessa tjänster uppvisar dock höga variationer i beräkningskraft som behövs beroende påegenskaper hos ingångsvideon, processeringsparametrar och ankomstmönster påbegäran. Sådana variationer har en betydande inverkan påeffektiviteten av använda datorresurser och därmed ocksådriftskostnaderna för dessa tjänster. Eftersom moln och distribuerade datorinfrastrukturer består av moderna flerkärniga processorer, måste multimedia bearbetningstjänster som körs pådem parallelliseras för att ytterligare förbättra det totala systemutnyttjandet. Tyngdpunkten i denna avhandling handlar alltsåom de olika forskningsfrågorna relaterade till proaktiv hantering av multimediabearbetning och deras parallellisering. Vi söker lösningar pådessa problem i samband med videotranskodningstjänster.

För att fåtillräckligt med information om egenskaperna hos verkliga filmer och de transkodningsparametrar som behövs för att modellera transkodningens arbetsbelastning, byggde vi en ett dataset med videoegenskaper med hjälp av data som samlats in från ett stort video-på-begäran-system. Datamängden innehåller en miljon slumpmässigt utvalda videor som uppvisar olika grundläggande videoegenskaperna. Analys av datasettet ger intressant statistik om grundläggande egenskaper hos online videor som kan utnyttjas ytterligare för att optimera eller modellera komponenter i ett multimediabehandlingssystem. Grundat påinsikterna från vår datamängd föreslogs en metod för att förutsäga transkodningens arbetsbelastning från en video till en annan med givna transkodningsparametrar. Tillvägagångssättet behandlar videotranskodningstiden som en stokastisk variabel och förutsäger den från statistik av tidigare observationer. Metoden förutsäger transkodningstiden som en funktion av flera parametrar hos ingångs- och utgångsvideo strämmar. Effektiviteten av metoden visas genom att jämföra de resulterande förutsägelserna med de faktiska transkodningstidena påosedda video strömmar. Simuleringsresultat visar att vår förutsägningsmetod möjliggör betydligt bättre resurshanteringsmetoder. Närmare bestämt presenteras proaktiv lastbalansering strategin för transkodning över virtuella maskiner och jämförs med klassiska lastbalansering metoder när det gäller total systemutnyttjande och kvaliteten påtjänsterna.

Förutom proaktiv hantering av datorresurser är en lämplig parallellisering av videotranskodningsprogrammen viktig för att garantera en effektiv användning av moderna multikärnarkitekturer som utgör moln och distribuerade datorinfrastrukturer. För det här ändamålet presenterar avhandlingen en lösning som möjliggör bekväma fingranulära parallella implementeringar av videobearbetningskomponenter med hjälp av dataflödes programmeringsspråk. Grovgranulära parallella metoder för videotranskodningapplikationer pågroup of picture nivån, baserade påmessage passing programmeringsmodellen, presenteras ocksåoch utvärderas med avseende påskalbarhet och kapacitet.

Acknowledgements

This thesis has resulted from efforts and support of many people and it is my pleasure to express my sincere gratitude to all those who have contributed in one form or another towards its completion.

Firstly, I would like to thank my supervisor Sébastien Lafond for believing in my efforts and giving me the constant support need to continuously improve my research and finally complete this thesis. At the same time, I would like to thank Professor Johan Lilius for giving me the opportunity to pursue doctoral studies and to work at the Embedded Systems Lab.

I also wish to thank Professor Jean-François Nezan and Assistant Professor Jarno Vanne for their time and efforts in reviewing my thesis and providing valuable and constructive comments, which helped in preparing the thesis into its current form. I am also grateful to Professor Jean-Francois Nezanand for his kind acceptance to act as the opponent at my doctoral defense.

The research work presented in this thesis including the collection of papers is a result of several peoples' ideas and efforts. I am especially grateful to all my co-authors: Fareed Jokhio, Habtegebreil Haile, Adnan Ashraf, Lionel Morel, Sébastien Lafond, Johan Lilius. Also, I would like to thank everyone for the efforts related to the various open source projects such as FFMPEG and Orcc that have been essential for this work to be possible.

I would like to extend my gratitude to my teachers at the Department of Information Technologies, especially Mats Aspnäs, Jerker Björkqvist, Hannu Toivonen, Nybom, Kristian and Ivan Porres. I also wish to thank my colleagues in the Embedded Systems Laboratory, particularly Fareed Jokhio, Johan Ersfolk, Wictor Lund, Sudeep Kanur and Georgios Georgakarakos.

Furthermore, I want to acknowledge the support of the administrative and technical personnel at the Department of Information Technologies. Among others, I would like thank: Christel Engblom, Tove Österroos, Karl Rönnholm, Tomi M äntylä, Nina Rytkönen, Pia Kallio, Joachim Storrank and Niklas Grönblom.

I am highly grateful and honored to receive generous scholarships and grants from Turku Centre for Computer Science (TUCS), Åbo Akademi University, European digital innovation and entrepreneurial (EIT Digital), and

Ulla Tuominen Foundation. Without such support, this work would not have been possible.

Last but not least, I thank my family for their endless love and support throughout my long journey leading to now. I wish to express my deepest gratitude to my late mother and father for their selfless love and profound dedication in creating the ideal environment. I would like to also thank my sisters and brother for their constant encouragement and support. I would also like to thank Faezeh Siavashi which I consider my family for her love and understanding and filling my days in Finland with joy.

Tewodros Deneke
Åbo, February 2016

List of Original Publications

1. Tewodros Deneke, Sébastien Lafond, Johan Lilius. Analysis and Transcoding Time Prediction of Online Videos. In *Proceedings of the IEEE International Symposium on Multimedia, 2015 IEEE International Conference*, pages 319–322. Miami, Florida.

2. Tewodors Deneke, Habtegebreil Haile, Sébastien Lafond, Johan Lilius. Video Transcoding Time Prediction for Proactive Load Balancing. In *Proceedings of the International Conference on Multimedia and Expo , 2014 IEEE International Conference*, pages 1–6. Chengdu,China.

3. Tewodros Deneke, Lionel Morel, Sébastien Lafond, Johan Lilius. Integration of Dataflow Components Within a Legacy Video Transcoding Framework. In *Proceedings of the International Workshop on Signal Processing Systems, 2015 IEEE International Conference*, pages 1–6. Hangzhou,China.

4. Fareed Ahmed Jokhio, Tewodros Deneke, Sébastien Lafond, Johan Lilius. Bit Rate Reduction Video Transcoding with Distributed Computing. In *Proceedings of the 20th International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2012*, pages 206–212. Munich, Germany.

5. Tewodros Deneke, Sébastien Lafond, Johan Lilius. Proactive Management of Video Transcoding Services. Submitted to *ACM Trans. Multimedia Comput. Commun. Appl., 2016*, pages 1–19. NY, USA

Contents

Part I

Research Summary

Chapter 1

Introduction

Creation of multimedia content and the ability to exchange it seamlessly across space and time has brought convenience to our lives and altered our ways of communication. Currently, we can create rich multimedia content and share it globally in an instant. This development can be attributed to advances in information technology, computational capability and communication networks.

Multimedia systems are inherently composed of components which are directly related to processes of content creation, compression, storage and distribution. Multimedia content creation involves capturing and digitizing of media such as video and audio. Advances in information technology have provided us with inexpensive and easy to use capturing and rendering devices which are suitable for various scenarios. With the availability of such devices, we are now capturing enormous amounts of multimedia content which require significant amount of storage. Storage and compression of multimedia involve the process of saving and archiving media content for later use. State-of-the-art compression algorithms and standards have been developed to minimize necessities for storage and distribution. Multimedia distribution involves the process of moving media content via cables, optical networks, wireless networks, satellites and any combination of them. Recent advances in digital communication networks and protocols have brought the possibility of instant access and real-time interactivity to media content.

Even though technological advances have brought us the convenience and easy of use in capturing, processing and distributing multimedia content, they have also brought an ever-growing variety of capturing devices, rendering devices, compression standards and transport protocols. This means content producers and video on demand service providers need a way of adapting their content to fit various devices and standards so that end users doesn't have to care how the content is delivered to their devices.

Transcoding is currently the principal method used for changing one encoded media format into another to adopt the media content to various standards, device resolutions, network protocols and bandwidth. However, transcoding is a computationally intensive process and requires a cluster of servers to operate. Infrastructure as a service (IaaS) clouds such as Amazon Elastic Compute Cloud (EC2), Google Compute Engine and Azure Compute provide virtual machines (VMs) based on pay as you go pricing model. Transcoding services such as Zencoder and Amazon Elastic Transcoder have been developed on top of these public IaaS clouds and other private ones [1, 2, 3]. These transcoding services are then presented to consumers as software as a service (SaaS) using pay as you go model. Managing and effective utilization of IaaS resources is, therefore, vital for transcoding services as it reflects on their overall running cost and performance and is the main topic of this thesis.

1.1 State of the Art

One form of multimedia which has recently gained a lot of popularity is video. The consumption of video by individuals has become more heterogeneous considering content requested, network connections, and display devices. Video transcoding together with adaptive streaming solutions aims to address this growing heterogeneity by offering users with multiple versions of a given video tailored to their various devices and network connections. This means, each version of a given video is encoded at a different bitrate, resolution and with a different codec so that the user is served with the most suitable format for his choice of device and network connection. Currently, video transcoding is being utilized for such purposes as a bit-rate reduction to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for ensuring high quality of service (QoS) [4, 5, 6, 7, 8].

To design an efficient and well-founded video transcoding system, one needs to understand the current state of consumption of video content. Lately there has been significant research on understanding the workloads of new generation video services. These research especially focus on the social aspect of videos and traffic characterization such as popularity, active lifespan, user access pattern, growth pattern, request patterns and their implication on the design of large-scale video-on-demand services such as caching and networking [9, 10, 11, 12, 13]. On the other hand collecting statistics of lower level video characteristics such as video length, size, bitrate, framerate, codec type and resolution besides the social aspect and traffic characterization of such services is becoming necessary as it can guide the efficient management of processing capacity of large-scale video processing services [14, 15].

Because video transcoding is a computationally intensive process and requires expert knowledge, it can not usually be done on the consumer side. Transcoding video content at different screen sizes, bitrates and quality levels requires not only time, expertise and equipment but also storage capacity and extensive planning in the design and automation of video transcoding workflows. Therefore, content providers, such as YouTube, rely on private and public cloud computing services for large scale video transcoding. As a result, cloud-based transcoding services have emerged to automate and virtualize the complex and cumbersome process of video transcoding for content providers and individuals [2, 1, 16].

Cloud computing is a form of on-demand computing and a way to use dynamically scalable and configurable virtual shared pool of resources as a service through an on-demand request over the internet. The main characteristics of the cloud include high scalability and availability. Scalability means that the cloud infrastructure can be expanded to large scale with several resources. Availability means that the services are available even when quite a number of nodes fail. In addition to such features, the cloud brings a new kind of business model to the IT industry where users pay for only the time they require the service (resource). Services such as video transcoding can use cloud resources and be presented as a pay as you go cloud services [17].

Managing cloud computing resources efficiently is necessary for service providers as it directly affects the cost of running the service. To proactively manage a cloud based video transcoding service, one needs to know the computational load it will encounter in the future. This load among other things depend on the amount of incoming transcoding tasks and their sizes. Previous research in the area has proposed algorithms that provide mechanisms for allocation and deallocation of resources based on a regression model that tracks and predicts aggregated target transcoding rate of videos in a system [18, 19].

In addition to high-level optimization and management of transcoding services, it is also important to explore the possibility of more fine-grained parallelism inherent in the transcoding operation itself. The need for parallelization is becoming more apparent as computer architectures are moving towards multi and many core concepts rather than pushing the speed (frequency) of a single core. New parallel programming models and tools have been developed to utilize this opportunity. Parallel computing concepts and methodologies have been around for decades and have evolved depending on available hardware platforms. However, the foundation is still the same. In general, parallel computing is mostly based on data parallelism and task parallelism concepts. In data parallelism, a set of program instances work on a different portion of an input data in parallel. On the other hand in task parallelism, a program is divided into tasks which can operate in parallel and pipelined manner. Currently, several parallel programming models exist as an abstraction over hardware and memory architectures. Among others, the most notable one is the threads model. Threads are the smallest and most basic unit of execution consisting of a set of instructions that can be managed by a runtime (operating) system's scheduler. Using threads programmers express explicitly the parallelism that exists in a given program on a shared memory. The programmer also needs to explicitly manage the synchronization among threads working on a shared memory to avoid any resulting non-determinism. This is, of course, challenging, time-consuming and error-prone process as threading becomes essentially a way to only prune non-determinism after its introduction. More abstract programming models have been developed to enable programmers to express parallelism implicitly. Dataflow is one model of computation that can be used to express program parallelism implicitly. Among others, the main advantages of using dataflow as the main programming model include ease of use, flexibility, automatic analyzability, automatic parallelizability, visual presentability and above all side-effect-freeness [20, 21, 22].

1.2 Research Questions

Video transcoding services are becoming important as the variety of video coding formats, end user devices and delivery networks are growing. In this thesis work, the primary focus is on the research issues related to 1) proactive management of transcoding service resources and 2) parallelization of the transcoding operation. Proactive management of transcoding service resources leads to an optimal provisioning and utilization of computing resources for a given quality of service (e.g. transcoding task waiting time or delay). Parallelization of the transcoding operation enables efficient utilization of the state of the art parallel hardware architectures that are used to host the transcoding services.

To be able to proactively manage resources needed in a video transcoding system one needs to first answer the question **what is the current state of video consumption from such perspectives as device resolution and bandwidth?** Analysis and understanding of the usage (consumption) of video from various perspectives such as popularity of codecs, popularity of device resolutions and bandwidth requirements is necessary to understand and simulate real world workload patterns in video transcoding services. Transcoding services require various types of resources which include computing, storage and bandwidth. Proper management and efficient utilization of these resources will lead to lower cost and better quality of service. This means, one needs to make sure that just the right amount of resources are being provisioned for the required quality of service at the right time. And since being proactive requires ahead of time knowledge of the system load, it thus becomes crucial to answer the question **Is it possible to accurately anticipate the resource requirements of a transcoding service ahead of time?** This means predicting video transcoding workload. Then the next research question is **How to design novel resource management algorithms exploiting the transcoding workload predictions?** Finally, to efficiently utilize modern multi-core heterogeneous computing platforms used to build current computing infrastructures, the question of **how is it possible to parallelize the transcoding operation itself?** needed to be addressed. The contributions of this thesis follow directly from the questions paused in this section and are presented in the next section.

1.3 Research Contributions

In this thesis, approaches for proactive management of video transcoding services and parallelization of video transcoding operation have been developed. To understand and model real world video transcoding workloads, we mine the web for video resources and analyze the state of current online video [23, 24]. Based on the insight gained from the data we propose and show how a video transcoding prediction model can be built [23, 24]. Further more we show how the transcoding time prediction models can be used in resource management algorithms such as load balancing, provisioning and admission control [25].To further ensure efficient utilization of resources we propose course grained parallelization approaches for video transcoding [26, 27]. We also show how dataflow programming paradigm can be utilized to facilitate fine-grained parallelism in video transcoding applications [28]. A brief overview of the main contributions is presented in the following sections and in part II the contributions are presented in detail as part of the original Publications.

1.3.1 Understanding State of Online Videos

As previously mentioned, proactive management of video transcoding services requires understanding the state of online videos in terms of their fundamental characteristics such as bitrate, framerate, coding standard and resolution. Knowledge of these basic video characteristics is an important first step in modeling and characterizing expected workload of transcoding services. In this thesis, the first contribution is thus a video characteristics dataset and descriptive statistics that summarizes the current state of online videos [23, 24]. The dataset was built using data collected from a large video-on-demand system, YouTube[1]. The dataset contains a million randomly sampled video instances listing ten fundamental video characteristics. We report our analysis of the dataset which provides insightful statistics on fundamental online video characteristics that can be further exploited to optimize or model components of a multimedia processing systems. In comparison to other benchmarking video datasets such as HD-VideoBench [29], Mediabench [30], and Berkeley Multimedia workload [31], which are very brief for a realistic workload on large scale systems and are only meant for accessing the performance of video codecs; The dataset presented in this thesis is randomly sampled from a real world UGC (user generated content) system and covers wider range of video formats.

1.3.2 Transcoding Time Prediction

Prediction of video transcoding time is useful for several reasons. For example, resource management algorithms deployed on a large scale transcoding service such as [2] can utilize this prediction to increase system utilization through proper load balancing and provisioning. It can also enable them to provide a more detailed service level agreement (SLA) which includes processing time and cost estimates. Thus as the second contribution of this thesis, we present an approach for the prediction of video transcoding time given transcoding parameters and an input video [23, 24]. Our prediction approach treats video transcoding as a random variable and it is statistically predicted from past observations. The method predicts video transcoding time as a function of several parameters of the input and output video streams. In comparison to previous works [14, 15] which have proposed transcoding or decoding time prediction models for specific coding standards, our approach [23, 24, 25] accounts for variability of video coding algorithms and various fundamental video characteristics such as bitrate and resolution. Its modelling is also driven by an insight obtained from the online video characteristics dataset presented in the last section.

1.3.3 Proactive Resource Management

Efficient utilization of cloud resources such as VMs requires effective ways of management which include proper VM provisioning, load balancing and admission control. As our next contribution of this thesis, we propose new proactive resource management algorithms that utilize our transcoding time predictions presented in the previous section [25, 32]. This contribution addresses the challenges exposed when managing computing resources in modern online video transcoding services. We design a proactive computing resource management scheme to aid throughput and quality of service in

[1]https://www.youtube.com/

terms of waiting time of jobs. Our method explores the opportunities provided by trends in transcoding workload of a given transcoding service. Due to the correlation between transcoding time and transcoding parameters, we can predict the transcoding time of a video given the fundamental characteristics of the input video and the transcoding parameters. The prediction is in turn used to correctly provision the right number of transcoding servers and load balance video transcoding requests across servers. The final result is thus better waiting time and the throughput of the system.

1.3.4 Parallelization

Besides proactive cloud resource management, one needs to exploit application level parallelism to further maximize utilization of individual running resource instances such as VMs. Therefore as the last contribution of this thesis we provide parallelization approaches for video transcoding. In [26] and [27] we present a group of pictures (GOP) level parallelization approach implemented based on message passing interface (MPI). Also, various video data segmentation methods are explored and presented. The use of such data parallelization approach enables transcoding service providers to process a given video on multiple VMs or using multiple processor cores that exist in VMs. In [28] we presented an approach allowing the integration of dataflow components within an imperative code. The approach makes use of a generic interface definition that allows seamless interaction between I/O components and data processing components. I/O components are mostly state operations and are best implemented in imperative languages. On the other hand, data processing components are mostly stateless dataflow operations and are best implemented in dataflow languages. The advantage of the approach is the ease of development by allowing each language to be used on those parts of the application that it is most appropriate for. The functionality of the approach is demonstrated by using the generic interface to add a new dataflow based components (MPEG and HEVC decoders) into an existing video transcoding framework written in an imperative language.

1.4 Research Methods

The research approach we followed in this thesis work is a mixture of design science, which is more concerned with building artifacts, and grounded theory which is concerned with creation of theory and models from gathered row data. We followed such an approach as we believe research work should serve both science and practice as well as due to the complementary nature of theory and practise [33, 34].

The research work presented in this thesis was carried as part of three Finnish research projects: Cloud Software, ParallaX, and EASYTRANS.

Cloud Software was a national Finnish research program, whose main aim was to significantly improve the competitive position of the Finnish software intensive industry in the global markets with a particular focus on cloud computing. ParallaX was a project concerned with the efficient use of many-core processors. The goal of the project was to develop methods for the development of efficient applications with low energy consumption. EasyTrans was a commercialization project having as its primary purpose the development of an efficient media transcoding platform.

In our research process, we followed the approach presented by [33, 35, 36], where one continually builds and evaluates. In the build phase we diagnose, plan and implement an idea. In the evaluate phase we assess and learn to improve on the next cycle until a satisfactory (research saturation) is reached.

Our research outputs are then classified according to the research output types presented in [33] which can be constructs, models, methods, instantiations and proofs. In this thesis our main outcomes are in the form of data collection methodologies, transcoding prediction models, proactive resource management algorithms, parallelization methods, and their instantiations (implementation).

1.5 Thesis Organization

The thesis consists of two parts. Part I provides a research summary, while Part II presents the original publications. Part I consists of five chapters. Chapter 1 starts by providing the motivation for this thesis followed by a brief discussion on the research contributions and methods used. Chapter 2 provides background material and discusses important related works. Chapter 3 presents a summary of the main contributions while focusing on the problems that they address. Chapter 4 provides a description and organization of the original publications and provides a mapping between the publications and the problems addressed. Finally, we present our conclusions and some future directions in the last chapter.

Chapter 2

Background and Related Work

In this chapter, a brief overview of the background concepts and technologies related to this thesis are presented. These main concepts include video transcoding, cloud computing and machine learning. Finally various specific concepts related to resource management are presented.

2.1 Video Transcoding

As mentioned earlier, digital video processing, transmission and storage technologies have progressed a lot in the past decades. Specifically, the progress in video compression technologies is driven by practical applications that play a key role in bridging the gap between the massive amount of data needed for video and the limited amount of storage, bandwidth and hardware capabilities. These practical applications include techniques such as motion compensation and residual coding which are useful for removing redundant information and reduce final coded video size.

In order to understand video compression we first need to understand how video is represented digitally. Fundamentally video can be seen as a discrete representation of the real world sampled in the spatial and temporal domain. In the temporal domain the real world is often sampled at 25-30 frames (images) per second or more. Each of these frames is in turn consist of spatial samples of the world in terms of pixels. Each pixel is then represented and stored using a number of bits. If we for example consider a typical high definition video of 1920x1080 pixels resolution, where each pixel is represented by 24 bits, then it can be calculated that an hour long video requires about 625GB of storage. This imply imply an enormous amount of storage requirement and certainly needs a huge bandwidth to be transmitted. The solution for this is video compression.

Fundamentally video compression can be thought of as a way of reducing the number of bits needed to represent a video. A software, a hardware or a specification that is used to compress (encode) and decompress (decode) a video is called a codec. An encoder essentially represents an original video with a model (coded representation) having as few bits as possible and with as high quality as possible. A typical modern video encoder therefore consists of three main components which include a temporal model, spatial model and statistical model that correspondingly exploit the temporal, spatial and statistical redundancies in videos to reduce the amount of bits needed to represent it.

The temporal model reduces redundancy by predicting the current frame from past and future ones while compensating for motion in the scene. This means a video encoded with most modern codecs contain different types of frames such as I (intra), P (predicted), and B (bidirectional predicted). An I frame is an independent reference frame that does not require any other frame while processing it (e.g. decoding it). On the other hand, both P and B frames require data from other frames while processing. This is because video compression algorithms only try to encode the difference between consecutive frames which apparently share certain similar content that can be referenced and reused allowing for more compression.

The spatial model further reduces the redundancies that exist in the residual (i.e. the difference between actual and predicted frame) by exploiting the similarities between neighboring pixels inside a frame. It achieves this by transforming the residual into a set of representative transform coefficients in another suitable domain and quantizing them so as to remove the least significant ones.

Finally the statistical model assigns shorter codes to represent more frequently occurring parameters from the spatial and temporal models to output the final compressed video. A video decoder does the reverse of the encoder to reconstruct the video.

In general video compression is a trade-off between quality and the amount of bits used to represent a video. In practical situations, this means that video content providers or video on demand (VOD) sites can decide on the most suitable encoding parameters that will result in a certain amount of video bitrate (size) that is tailored for a resource a user might have. These resources include bandwidth, device display resolution, and etc..

The difference in device resources, network bandwidth and video representation types results in the need for a mechanism enabling video content adoption. This mechanism, called transcoding is currently being used for such purposes as bitrate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for insuring high quality of service (QoS) [37, 38].

As can be noted from Figure 2.1 a generic video transcoder contains five main parts, each having a set of components depicted as blocks. The five

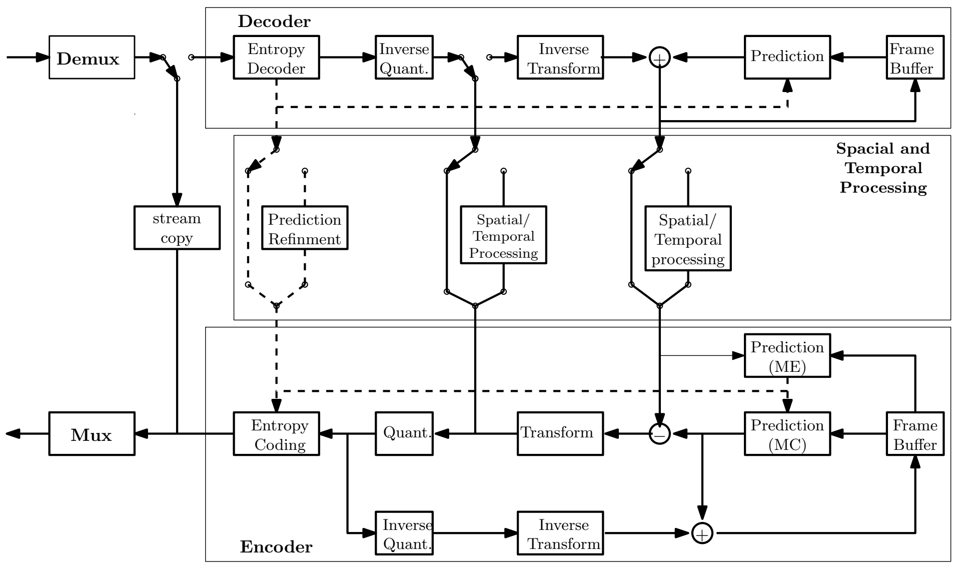


Figure 2.1: A complete overview of video transcoding. It shows the main blocks a transcoder constitutes and the different options of doing transcoding depending on the required type of conversion operation, acceptable quality and complexity. Note that not all the blocks need to be executed all the time. The transcoder tends to use information from the input video to simplify and skip blocks of operations.

main parts include a demuxer, a decoder, spatial and temporal processors, an encoder and a muxer. The demuxer is used to read interleaved streams (e.g. one audio, one video and a subtitle stream) from a network or a file. Usually a set of streams is encapsulated in a container format such as MP4. Packets which are read by the demuxer and contain compressed audio/video frames are then passed to the appropriate decoder to be decompressed. The decompressed audio/video frames are then spatially or temporally processed to adapt the video/audio to a particular framerate and/or resolution. Spatially and temporally processed uncompressed frames are then passed on to the encoder to be compressed via removing temporal, spatial and statistical redundancy that exists inside and among uncompressed frames.

Currently existing transcoders such as FFmpeg[1] constitute libraries corresponding to each block. For example, FFmpeg contains multiple codec implementations including H264, HEVC, MPEG4. A video codec is a hardware or software implementation of video compression and decompression algorithms. A variety of video coding standards have thus been established through the years. These standards are broadly categorized into three groups: a) the H.26x video standard family developed by the ITU (Inter-

[1] urlhttps://ffmpeg.org/

national Telecommunication Union), b) the MPEG standards developed by the ISO (International Organization for Standardization), and c) other codec standards such as On2 (vp8) developed by large corporations like Google and Microsoft. All these standards define only a compressed bit stream so that any standards based decoder can decode the bit stream. However, the compression algorithms followed in the encoders are completely dependent on the proprietary technology implemented by the encoder manufacturer. Each new standard from these groups is initiated with an objective to support an application with a new research and development technology.

Similarly, FFmpeg contains multiple implementations of muxers and demuxers that can be used to read, write and transmit multiple video and audio streams in one container. Some of these formats include MP4, MKV, FLV, MP3, WAV and DASH. Among these formats DASH (Dynamic adaptive streaming over HTTP) is the most notable one and is used for adaptive streaming of video content. DASH is an adaptive streaming technique where videos are segmented into multiple small chunks and are encoded in several formats which are presented to a client. The client then decides which format of a particular segment to consume depending on its available resources.

2.2 Cloud Computing

As the Information and Communication Technology (ICT) advances we begin to witness computing becoming the 5^{th} utility following water, electricity, gas, and telephony. Similar to the other utilities, computing has become essential for our society which requires access to virtually unlimited, uninterrupted and reliable access to computing resources. Computing as a utility relies on an on-demand service provisioning model where we won't any longer compute on our local computers but on a rented, centralized and remote facilities provided by a third-party who makes computing resources available as needed charging only for a specific usage on pay as you go basis. The idea of utility computing has been around since 1960s, where John McCarthy, Leonard Kleinrock and others predicted that computation may someday become a public utility [39, 40] [41, 42]. By 1990s Grid computing was proposed and several instances were implemented including TeraGrid and Open Science Grid [43, 44]. Since then Grids have provided on-demand computing resources mostly for scientific institutions. However, they lacked a reliable and easy to use business model required for public and commercial susses. Also, Grids have elaborate overall architecture and resource management system due to their support for a heterogeneous pool of resources across different domains [45, 40]. Cloud Computing in its modern sense has emerged as a popular paradigm during the 2000s. In some sense, Cloud Computing has evolved from Grid Computing where the focus is shifted from purely infrastructure building and application oriented towards an economics of scale, abstraction and service oriented [40, 39].

As an emerging paradigm, Cloud computing is defined in a variety of ways [46, 39, 47, 40]. Among others [40] defined cloud as:

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

The National Institute of Standards and Technology (NIST) defines cloud computing as [47]:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

Where the five essential characteristics include on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. The service models include Infrastructure as a service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). And the deployment models include private cloud, community cloud, public cloud, and hybrid cloud [47].

From the previous Cloud Computing definitions, we can note the main distinguishing features as its: 1) scalability where users can request seemingly unlimited resources, 2) abstract interfaces which are based on WEB 2.0 protocols that enable natural interaction for users, 3) economics of scale where a pool of resources are utilized more efficiently by many users as per demand leading to cost reduction, 4) dynamic configureability where users are provided with a simple measuring unit that they can utilize to easily and dynamically reconfigure required Cloud resources.

In recent years Cloud Computing has become attractive for the IT industry. This is in particular due to its pay-as-you-go business model which allows small start-ups or private users to gain access to high-quality IT infrastructure for a small cost. With such a business model, start-ups and private users will avoid investing in their own infrastructure before they prove their business model while cloud providers will benefit from the economics of scale.

Among other applications, video transcoding is one application that can exploit the benefits of Cloud [1, 2, 3]. Video transcoding is a computationally expensive process and requires large-scale computing infrastructure.

At the same time, media content producers and other entities that require transcoding services would be better off concentrating on creation of original content where their expertise lie than building and maintaining their own computing infrastructure. Just like these media content producing companies don't own their electric power plants, they now do not need to own datacenters as it has become a utility that can be bought readily online when needed. Currently, there are multiple cloud-based transcoding service providers. Transcoding services utilize the cloud infrastructure (IaaS) to provide their services as Platform as a service (PaaS) or Software as a Service (SaaS). Various research studies are being carried out in the area of large-scale video transcoding in the cloud which will allow efficient management of resources and better usability[48, 49, 50].

2.3 Machine Learning

Advances in computer technology have enabled us to acquire, store and analyze large amounts of data. For example, imagine the many Cloud providers that have several data centers across the globe with thousands of servers hosting millions of virtual machines that are utilized by millions of users. These Cloud providers record details of the usage of their virtual machines including CPU utilization, disk read-write operations, network input-output packet flow, and so forth. This typically amounts to gigabytes of data every day. This data will become useful when it is analyzed and turned in to information from which one can make predictions, for example, future usage pattern of users to enable efficient resource utilization.

Even though we do not know exactly how a Cloud user utilizes virtual machines in the cloud we are aware that their usage pattern is not random. This means that even if we may not be able to identify the process entirely, we can still construct a good and useful approximation. This is the niche problem that machine learning tries to solve. Machine learning can be seen in general as the use of a set of observations to uncover an underlying process that produces the observations. This is especially useful if the problem is less specified and one can not analytically derive the underlying process (function) and needs to rely on data obtained from process to approximate the target function that produces it. The basic premise of machine learning is, therefore, the use of a set of observations to uncover an underlying process.

Machine learning is broadly divided into three types which include supervised, reinforcement, and unsupervised learning. In supervised learning, we generally have a training data that contains explicit examples of which the correct output for given input is provided. In reinforcement learning, the training data does not contain the correct output for each input. Instead, for each input, we have some output and an associated grade for that output which provides a measure of how good the output is. Finally, in unsupervised learning, the training data only contains input examples. It is mostly used to spontaneously find patterns and structure in the input.

As an academic discipline, Machine learning grew out of the quest for artificial intelligence (AI). Already in the 1940s and 1950s, researchers were interested in building general purpose learning systems that start with little or no initial structure or task specific knowledge. They attempted to approach the problem through the application of symbolic logic which they used to model nervous system activities and termed it as neural networks and showed how it could learn [51, 52, 53]. After its first establishment as a separate field, AI has shown real promise and enthusiasm. Several applications have been developed [54, 55]. However, challenges such as limited computational power, combinatorial explosion and some fundamental limitation in the underlying structure of the smallest units of learning (perceptions) has made the advances in AI to slow down [56, 57] and led to the popularity of expert systems until the reinvention of backpropagation in the 1960s [52, 58].

Machine learning was reorganized as a separate field and started to advance in the 1990s. As a separate field, it focused its goal from general purpose learning system to tackling solvable practical learning problems. It also shifted its focus away from the symbolic approaches it had inherited from AI towards methods and models borrowed from statistics and probability theory. In addition to that, it also exploited the benefits gained from the increased availability of digitized information usable for training and the availability of better computing power [58].

One of the contributions of this thesis work is the prediction of video transcoding workload. To predict the transcoding workload of a video two of the most widely used supervised machine learning algorithms were used, the SVR and Neural Net. The general idea behind any regression problem in machine learning can be summarized as given a set of t observations with n features (bitrate, framerate, codec, etc.) each and a target variable (transcoding time) y as $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_t, y_t)\}$ where $\mathbf{x} \in \Re^n$, $y \in \Re$ the objective is to find a function (model)

$$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b \text{ with } \omega \in \Re^n, b \in \Re \tag{2.1}$$

with the best fit.

Neural nets The idea of neural networks was first inspired by the nervous system of human beings which consists of many simple processing units called neurons. Each neuron receives some input signals from outside or from other neurons and processes them with an activation function to produce its output and sends it to other neurons. These neurons can be understood as a mathematical function that take n element input vector and scale each data element x_i, by a weight w_j. The scaled data is offset by some bias b and

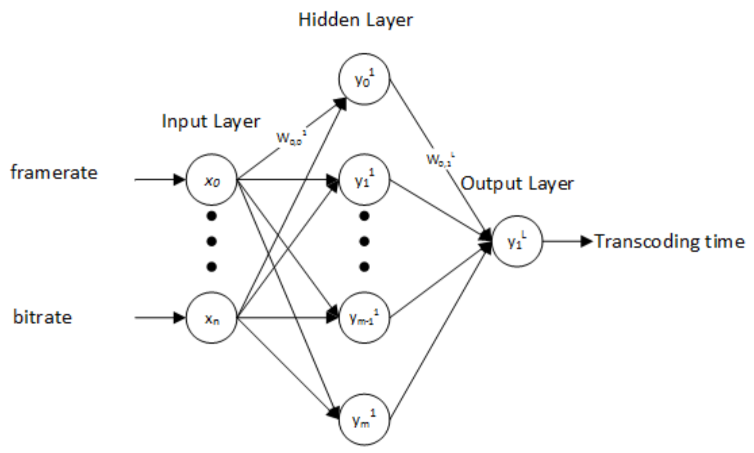


Figure 2.2: Multilayer Neural network as applied to video transcoding prediction.

put through a differentiable activation function f. The output of a neuron can be analytically viewed as in Equation 2.1. The impact of Each input is weighted differently from other inputs; Thus the neuron can interpret the data differently depending on the weight and bias. Consequently the more is the weight, the stronger would the connection be allowing that data point to influence the output more and vice versa. The activation function f can be linear or non-linear. Non-linear activation functions are useful in mapping non-linear relationships. One such function is called sigmoid which is represented as:

$$\frac{1}{1 + exp(-f)} \tag{2.2}$$

A network of these neurons forms a feedforward multilayer neural networks as shown in Figure 2.2. These networks are made of layers of neurons. The first layer is the layer connected to the input data. After that there could be one or more middle layers called hidden layers. The last layer is the output layer which shows the results. One of the learning methods in multilayer perception Neural Networks is the error back propagation in which the network learns the pattern in a data set and justifies the weight of the connections in the reverse direction with respect to the gradient vector of an error function. The error function is usually a regularized sum of squared error. The back propagation method picks a training vector from the training data set and moves it from the input layer toward the output layer. In the output layer, the error is calculated and propagated backward, so the weight of the connections will be corrected. This will usually go on until the error reaches a predefined value. It's proved that we can approximate any continuous function with a three layer feedback network with any precision [59, 60] .

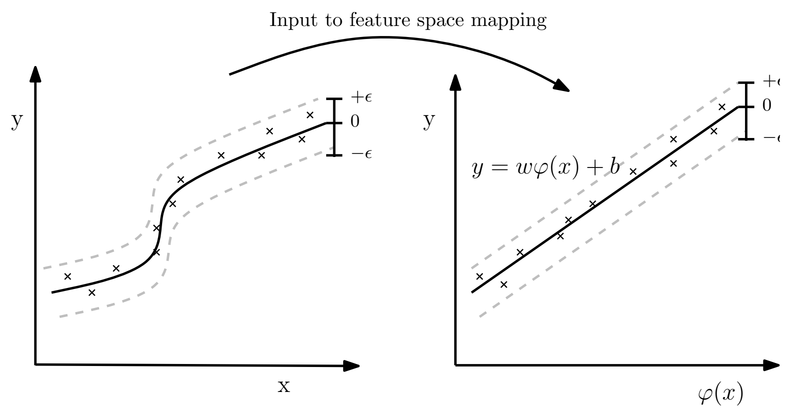


Figure 2.3: Support Vector Regression.

Support vector machines treats the regression problem as a convex optimization problem where the main idea is to minimize error for a given constraint in terms of error margin:

$$\text{minimize } \frac{1}{2}\|\omega\|^2 \tag{2.3}$$

$$\text{subject to } = \begin{cases} y_i - \|w, x_i\| - b \leq \epsilon \\ \|w, x_i\| + b - y_i \leq \epsilon \end{cases} \tag{2.4}$$

For non-linear problems, a non-linear SVR can be used. In non-linear SVR the training data is first mapped into higher dimensional feature space via a non-linear kernel function as shown in Figure 2.3. One such kernel is the radial basis function (RBF) which is represented as

$$exp(-\frac{1}{2\sigma^2}\|f\|^2)$$

2.4 Transcoding Workload Understanding

Currently, there are only a few research works in analysis and modelling of video transcoding workloads specifically, but there has been significant research on understanding the workloads of new generation video servers in general. These research especially focus on the social aspect of videos and traffic characterization such as popularity, active lifespan, user access pattern, growth pattern and request patterns [9, 10, 11, 12, 61, 13]. Recent works; however, are showing interest in understanding video transcoding workloads especially for determining optimal video representation sets and just-in-time transcoding bitrate selection [62, 63, 64].

Yu et al. [9] studied user behavior, content access pattern and their implications on the design large-scale video-on-demand systems. Based on measurements collected from a large video on demand (VOD) system they were able to model the arrival pattern of users and provide an explanation on the various factors that contribute towards the popularity of videos in VOD sites. Such type of models and understanding are useful when modeling, simulation and further understanding of large-scale VOD systems. Possible improvements on UGC design were proposed by Cha et al. [10] after analyzing traces containing various video information from YouTube and Daum, a popular UGC in Korea. The main improvements proposed include better caching schemes and a peer-assisted video delivery system that save storage and bandwidth. Analysis of the traces also provided insights such as the relationship between video age and requests made. After tracking YouTube transactions from a network edge, Gill et al. [11] were able to analyze YouTube traffic characteristics and discuss the implications of their observation on key concepts such as caching. In their work Gill et al. [11] collected YouTube traffic locally in a campus and examined over time the most popular videos on the site. Based on the collected data they were able to draw insights such as the need for caching and relation between local and global popularity. The caching problem in YouTube have been further studied by Zink et al. [12]. In their work, they collected and analyzed Youtube traffic from a network edge. Taking their analysis result into account, which showed that users usually tend to view a video more than once, they proposed and analyzed the benefits of client side caching mechanism. Furthermore, they showed the benefits of P2P based caching mechanism due to the observation that similar videos are accessed from the same locality. The social networking among videos was studied in the works of Halvey and Keane [61] and [13]. Mislove et al. [13] crawled social networking sites for their user graphs and confirmed the power-law, small-world, and scale free properties of online social networks. They also observed that the graphs contain a densely connected core of high-degree nodes which links small groups of strongly clustered low-degree nodes. Such findings about structural properties of social networks can lead to better information dissemination and search algorithms.

After noting the drawback of transcoding video segments into all possible bitrates on wasting computing and storage resources, Krishnappa et al. [64] proposed several just-in-time transcoding polices. In order to materialize their proposed just-in-time transcoding mechanism they first constructed a Markov model for accurately predicting the bitrate of the next video segment that will be requested by a user. Their prediction model is built based on a dataset collected from Akamai's video content delivery network (CDN) containing users video access pattern. In [63] datasets characterizing the popular live streaming provider (Twitch) and its computing resource needs for transcoding videos are presented. Based on the dataset they proposed solutions for the optimal number of video representations required to maximize the average user quality of viewing experience and the corresponding encoding parameters required for creating the representations such as resolution and bit-rate. The optimization problem and its solution is presented in detail in [62].

Rather than only focusing on the social aspect and traffic characterization of such services, our works [23, 24] in this area focus on collecting detailed statistics of video characteristics such as video length, size, bitrate, frame rate, codec type, resolution, etc. This contribution will thus fill some of the gap in video characterization from the view of computational workload modeling of large-scale video processing systems.

2.5 Transcoding Workload Modeling and Prediction

Several workload prediction models for video processing applications have been proposed in the literature. The existing models can be classified into two categories: models based on history and models relaying on information extracted from the video bitstream. In the history based methods, the workload of the current work item (e.g. a frame) is predicted as a weighted average of previous work items [65, 66, 14, 18]. However, due to the large variability in video processing workload of consecutive work items, the history-based models often suffer in terms of accuracy. Consequently, research in this area focus on improving prediction accuracy. On the other hand, models based on information extracted from video bitstream predict future workload based on models constructed from predictive features obtainable from the bitstream [67, 68, 69, 15]. These models often tend to be more accurate but incur more overhead due to the time required for video bitstream parsing. Research in this area thus mostly focus on finding less expensive yet predictive features.

In Choi et al. [65] the decoding time of a current frame is predicted based on a moving average over past frames of a similar type. This means that decoding time prediction of I, P and B are done separately. The resulting prediction is then used for scaling the processor voltage and frequency in order to provision the proper amount of computing power required to decode a frame.

Bavier et al. [66] proposed a model which can predict video decoding workload at frame and network packet level. They utilized linear regression analysis to gain insight on the relationship between MPEG bitstream components and decoding time of a frame or a packet. Their frame level predictor uses running average of past frames of similar type while taking into account the byte length of the frames. Similarly their packet level predictor uses a running average of past frames of similar type taking into account the number of blocks of the packets. Their prediction approach is designed to be computationally cheap and usable for real-time multimedia application scheduling.

Guo and Bhuyan [14] determined the necessity of predicting the CPU load of transcoding tasks in order to schedule them on a cluster of computing nodes. They, therefore, proposed an online prediction algorithm that can dynamically predict the processing time of video segments (GOPs). Their predictor is a linear model where its slope is incrementally approximated according to the difference between accumulated regional and global means of GOP transcoding times and sizes.

Huang et al. [68] proposed a workload prediction technique for video decoding which is based on an offline bitstream analysis of a video. The predictions are then used to insert metadata information, a sequence of frequency values, with which the processor needs to run while decoding various segment of the video. The ultimate goal of their work is saving energy while decoding video streams. In a later work [69] they proposed a new workload-scalable transcoding scheme which converts a pre-recorded video bitstream into a new video bitstream that satisfies a given playback device workload constraint while keeping the transcoding distortion minimal as measured in terms of their proposed compressed domain distortion measure, a function of frames per second and bits per frame.

Roitzsch and Pohlack [15] presented the design and implementation of a per-frame decoding time prediction method for MPEG based video decoders. In order to find useful prediction metrics, they divided the decoding process into logical steps and established metrics from the video bitstream that are useful to get reasonable execution time estimates for each step. They then modeled the prediction problem as linear least square problem and solved the model coefficients using a training dataset collected over test video sequences.

Most of the related works in the area of multimedia workload modelling and prediction focus on one coding algorithm and use only a brief set of videos to train their prediction models. The proposed transcoding workload modeling and prediction approach in this thesis is designed to work across multiple coding algorithms [23, 24]. It is also modeled based on the video dataset presented in the previous section having a realistic distribution of online video characteristics.

The effectiveness the transcoding workload prediction method is shown via comparing the resulting transcoding time predictions with the actual transcoding times on unseen video streams.

2.6 Proactive Transcoding Resource Management

Most of the research work in proactive management of computing resources for video transcoding applications and services focus on load balancing, provisioning and task migration. These works often follow from workload modelling and prediction works presented in subsection 2.5.

Choi et al. [65] used a moving average based frame decoding workload prediction for scaling the processor voltage and frequency so that they provision the proper amount of computing power required to decode a frame. In [18] prediction-based dynamic resource allocation algorithm to scale video transcoding service on a given Infrastructure as a Service cloud were discussed. The proposed algorithm provides mechanisms for allocation and deallocation of virtual machines based on a regression model that tracks and predicts the aggregate target transcoding rate required by the service. In Huang et al. [68] and Huang et al. [69] the authors applied frequency scaling for proper provisioning of computing resources for decoding and transcoding applications. The frequency value at which the processor should run at specific times during the video playback or transcoding is inserted into the video bitstream.

In [14] the authors proposed a load balancing scheme based on prediction of video transcoding workload at GOP level. Their main aim was to minimize the total processing time while maintaining the order of media units for each outgoing stream. In their work, they designed and evaluated algorithms such as First Fit (FF) and Adoptive Load sharing (ALS). Kuang et al. [70] proposed and evaluated a power-efficient and traffic aware transcoding system for multicore servers. The approach manages computing resources by adjusting processor operating levels that match the incoming traffic rate. More specifically their approach is capable of configuring the number of active cores and core frequency on-the-fly according to varying traffic rate.

In this thesis work, transcoding workload prediction models that rely on information extracted from the bitstream are proposed and used. The main reason for the use of such approaches is the need for capturing the high variability in terms of transcoding workload among a sequence of frames in a video stream. Based on these prediction models it is then shown how it is possible to manage computing resources efficiently. The idea is demonstrated through evaluation of proactive provisioning and load balancing approaches.

The effectiveness of the proposed proactive load balancing approach for transcoding jobs across virtual machines is evaluated by comparing it with classical load balancing methods in terms of total system utilization and quality of service. Also, a proactive provisioning method that enables provisioning of the right amount of transcoding servers for a given quality of service was presented and evaluated.

2.7 Transcoding Parallelization

Most of the research work around parallelization of video processing applications including video decoding, encoding and transcoding can be broadly divided into three main areas: problem decomposition, process interaction and resource management mechanisms. Video processing applications are among the most compute intensive applications in computer systems, and various ways of partitioning these applications have been employed to be able to deploy them on multi-processor architectures and distributed systems efficiently. The most commonly used problem decomposition approaches include task, data and implicit parallelism [71, 72, 73, 74, 75]. Research related to process interaction, on the other hand deal with the mechanisms by which parallel processes that make up the applications communicate with each other. These interaction mechanisms include message passing, shared memory and implicit interaction approaches [76, 77, 78]. Resource management research in the context of video transcoding application parallelization include runtime and compiler works that try to map applications efficiently; that are, decomposed into parallel entities and their interaction is expressed in a certain way into various physical hardware architectures. Note that quite a few of the research works in these categories are interrelated and are often approached together.

As the computational complexity of video codecs grows more parallelization approaches are being proposed and integrated into newer coding standards. In [71] a data parallel problem decomposition approach for H264 on a 64 core shared memory architecture is presented. The approach shows how macroblocks (MBs) can be processed in parallel both at intra-frame and inter-frame levels. They also presented a scheduling approach that avoids the latency and large memory requirements associated with processing a large number of MBs spanning over multiple frames. In [72] a more detailed discussion and comparison of data parallel problem decomposition approaches for H264 video codec are presented. Chi et al. [73] have nicely presented efficient implementations of the most promising parallelization proposals for HEVC which include tiles and wavefront parallel processing. HEVC [79] is currently the newest video coding algorithm where parallelism was a forethought in contrast to previous video coding algorithms. This fact further shows the relative importance of parallelization in the recent years.

In [74] and [75] an implicit problem decomposition approach for MPEG type video coding standards is presented. Janneck et al. [74] noted the current C/C++ based monolithic video coding standard specifications hide the inherent data flow structure found in video coding algorithms. Their observation led them to propose the Reconfigurable Video Coding (RVC) standard which attempts to address this issue by building a framework that supports the construction of video standards as libraries of coding tools.

These libraries can be incrementally updated and extended, and the tools can be aggregated to form complete codecs using a streaming (or dataflow) programming model, which preserves the inherent parallelism of the coding algorithm. Along with that they provided a tool-chain to develop and compile video coding algorithms on various heterogeneous and multi-core architectures.

While problem decompositions is usually an important first dimension in parallelization of video coding applications, an equally important next dimension is the interaction mechanism among program partitions. Barbosa et al. [76] proposed a simple shared-memory parallelization approach for MPEG video encoding based on simultaneous execution of different coarse-grained and hierarchical tasks including reading, writing and encoding. To facilitate interaction among task partitions they used a fork-join and a bag of tasks synchronization approaches. In [77] a new adaptive slice-size selection technique for efficient slice-level parallelism of H.264/AVC encoding on a multi-core processor is proposed. A fast MB mode selection method as a pre-processing step was also used to decrease the overhead of fine-grained MB based parallelism. Sambe et al. [78] proposed a distributed video transcoding system that can simultaneously transcode an MPEG- 2 video file into various video coding formats with different rates. The transcoder divides the MPEG-2 file into small segments along the time axis and transcodes them in parallel. They also propose efficient video segment handling methods that minimize the inter-processor communication overhead and eliminate temporal discontinuities from the re-encoded video. Tian et al. [80] proposed a distributed video transcoding approach similar to [78] while improving on the segmentation approach to obtain a better total transcoding time. On the other hand in [81] a similar distributed video transcoding approach was proposed for active routers. Lao et al. [82] proposed a map/reduce based cloud transcoder. They also proposed a heuristic load balancing algorithm which minimize the total transcoding time of videos under the assumption that transcoding complexity of segments is known in advance somehow. In [74] and [75] similarly to the implicit nature of the problem decomposition, the interaction among tasks has been made implicit to the programmer. An ever increase in complexity of video coding means an ever growing necessity for automating the parallelization of video coding algorithms.

Parallelization is done on various level of granularity. In [26] and [27] a distributed video transcoding approach where multiple processing nodes transcode GOP level video segments in parallel has been proposed. Transcoding processes executing on different nodes interact with each other using a message passing interface. This thesis work mostly relates to [78] and Tian et al. [80] and focus on coarse-grained parallelism to minimize communication overhead across nodes of a typical distributed system. In contrast to others; in this thesis work, multiple types of segmentations strategies have been considered to reduce the total transcoding time of videos. In [28] we have proposed a new approach that allows the integration of dataflow components written in dataflow language within an existing transcoding framework written in an imperative language. The approach makes use of a generic interface definition that allows seamless interaction between an existing code written in an imperative language with highly parallel data flow components written in a dataflow language. The advantage of the approach is the ease of development that allows the use of different languages for different parts of an application.

Chapter 3

Contributions of the Thesis

In this thesis work, the problem of efficient computing resource utilization for large scale distributed video processing is addressed based on proactive management and parallelization approaches. Solution to the problem is mainly sought in the context of a video transcoding application.

In order to understand the state of online video being produced and consumed today, we mined YouTube and performed a descriptive statistics on various important video characteristics. Based on the insight gained from the dataset mined from YouTube and our domain knowledge we have selected a set of features (metrics) that are useful in predicting the transcoding workload of videos. We showed how a transcoding workload prediction model can be built based on our dataset and some of the widely used supervised machine learning algorithms [24, 23]. Furthermore, we show how the workload prediction model can be used to load balance transcoding jobs across servers proactively [23]. We also present analysis and implementation of a course level parallelization approach for video transcoding [27, 26]. In addition, we introduces a new approach that allows the integration of dataflow components within imperative code. The approach makes the development of fine-grained parallelism of video coding components easy and intuitive [28].

In the following sections, we provide a summary of the main contribution of this thesis along with a brief overview of some of the most important challenges they address. The details of our contributions are presented in the original publications found in part II.

3.1 Analysis and Characterization of Online Video

To obtain enough information on the characteristics of real world online videos needed to model video processing workloads, researchers need datasets obtained from real world video on demand services. As our first contribution of this thesis, we built a video characteristics dataset, using data collected from a large video-on-demand system, YouTube. The dataset contains a million randomly sampled video instances listing 10 fundamental video characteristics [83, 23]. The data was collected based on an unbiased sampling method, the random prefix sampling [84]. Our analysis of the dataset provides insightful statistics on fundamental video characteristics and can be further exploited to optimize or model components of a multimedia processing systems.

3.1.1 Crawling YouTube and Collecting Data

We collected our dataset systematically from YouTube, the largest user generated content (UGC) video website. By 2011 YouTube is estimated to have over half a billion videos and accounts for a significant percentage of the world's bandwidth utilization [84]. YouTube organizes its videos as a directed graph, where each video is considered as a node and edges represent links to a set of related videos.

There are several known ways of crawling YouTube's video graph [84, 85]. In this thesis, we used the random prefix sampling method from [84]. Random prefix sampling over YouTube video ID space is done through utilizing a unique property of the YouTube's API which allows searching videos using a randomly generated prefix of a possible YouTube video ID. YouTube video ID consists of 11 characters drawn from a set $S = \{0 - 9, A - Z, a - z,, -\}$. Providing a search string of "v=abc..." where "abc..." is a randomly generated string of length less than or equal to eleven returns a list of video IDs which start with this string. This fact enables one to avoid the impossible task of uniform random sampling over the extremely large id space of the order 64^{11}. Sampling YouTube's video graph through random prefix sampling rather than breadth-first search or other graph based sampling techniques produces a more unbiased sample. In our experiment, we used a randomly generated four character prefix to sample the YouTube's video graph. Experiments show that a prefix larger than four often returns an empty list indicating the fact that YouTube's video ID space is randomly generated.

In order to collect our online video characteristics dataset we implemented a small wrapper tool in Java over other well known open source tools, ffprobe [86] and youtube-dl [87] . Our tool first generates a four character random prefix from YouTube's video ID space and searches YouTube through its API. The API will return a list of valid video IDs starting with the random prefix. A video ID is then selected randomly from the returned list. Then, a set of direct links for all stored file formats of the video with that ID is fetched by the open source tool, youtube-dl. Each of these video links are then probed by ffprobe, a video analysis tool that can be used to collect video characteristics such as bitrate, framerate, resolution, container format, codec, duration and frame types.

The dataset contains ten columns of fundamental video characteristics; Duration, video codec, framerate, estimated framerate, total bitrate (audio plus video bitrate), video bitrate, resolution ,category, direct video link and video ID. This dataset can be used to gain insight in characteristics of videos on UGC.

3.1.2 Analysis of online Video Characteristics

A video has several fundamental characteristics that define its behavior in terms of quality, storage, bandwidth requirement and processing time. Some of these fundamental video characteristics include its duration, video codec, framerate, bitrate, resolution and category. A descriptive analysis and more detailed analysis of these characteristics as obtained from our dataset are presented in [24, 23] and a brief overview of some of these properties is presented below. Such statistics is useful in modeling different parts of a video processing service.

Bitrate One of the most important characteristics of a video file is defined by its bitrate. It indicates the number of bits being processed per unit time. Video-on-demand sites such as YouTube should select a set of optimal bitrates for their videos considering quality of service and bandwidth availability. Larger bitrates enable higher quality but will require larger bandwidth and processing power.

Framerate Another important video characteristic is the framerate. It indicates the number of frames (pictures) produced by a given video source per unit time. The human vision system perceives a sequence of pictures as a video if it has a framerate of 25 frames per second (fps) or more. Video-on-demand sites select a set of optimal frame rate for their videos considering quality of service, bandwidth and processing power availability.

Resolution of a video is defined by the number of pixels in its two dimensions. Video-on-demand sites should select a set of optimal resolutions for their videos considering quality of service, bandwidth and more importantly the types of displays owned by their users.

Codec A video codec is a hardware or software implementation of video compression and decompression algorithms. A variety of video coding standards have thus been established through the years. Each new standard is developed with an objective to support an application with a new research and development technology. Thus, earlier standards have very limited scope, whereas the later standards tend to include more advanced algorithms and techniques. This makes a codec one of the most important characteristics in determining the workload of a video transcoding task in terms of processing time.

Frame Types Depending on the type of codec used and its parameters, encoded videos contain different types of frames such as I (Intra), P (Predicted), and B (Bidirectional predicted). The proportion of each frame types in a video is an important characteristic as it correlates well with video size and processing time.

Formats YouTube stores its videos in several formats. These formats enable YouTube to provide its services to wide range of devices having different resolution, supported codec and connected through various types of networks.

3.2 Prediction of Video Transcoding Time

Prediction of video transcoding time is important for several reasons. For example, resource management algorithms deployed on a large scale transcoding service such as [88] can utilize this prediction to increase system utilization through proper load balancing and provisioning. Previous works such as [14] has proposed to model the prediction problem by characterizing a video using simple and few features (e.g. size alone). However, such models did not account for variability of video coding algorithms and other factors that make prediction more complex.

In this thesis, as our second contribution, we propose the use of more video characteristics and machine learning for better accuracy and generality over a range of coding algorithms. Machine learning techniques are often used as decision-making mechanisms for a variety of systems. Basically, machine learning allows computers to evolve behaviours based on empirical data, in our case, this is a collection of samples with important video characteristics, transcoding parameter sets and measured transcoding times. This means video transcoding time is treated as a random variable and is statistically predicted from past observations. More specifically our proposed method predicts the transcoding time as a function of several parameters of the input and output video stream. The details of the approach and experiments on its prediction accuracy are presented in [24, 23, 25] and a brief overview is presented in the following subsections.

3.2.1 Transcoding Time Prediction Model

The prediction model used in this thesis is depicted in Figure 3.1. We refer this model as a *transcoding time predictor*. It takes as input a video characterization $C = \{c_1, c_2, ..., c_n\}$ and transcoding parameter sequences $P = \{p_1, p_2, ..., p_n\}$, and it outputs the predicted transcoding time.

A general formulation of the transcoding time prediction problem is to construct a function that takes as input easily extractable characteristics of a video together with input parameters that specify the characteristics of the output video and generate an estimated transcoding time of the video on a given platform.

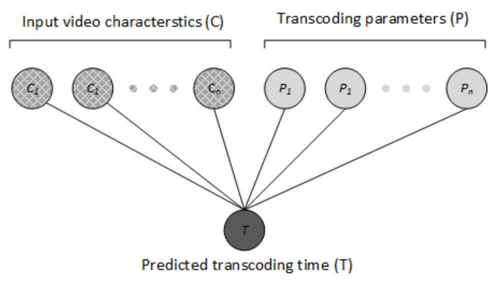

Input video characterstics (C)
Transcoding parameters (P)
C_1
C_1
C_n
P_1
P_1
P_n
T
Predicted transcoding time (T)

Figure 3.1: Transcoding time prediction.

3.2.2 Input Video Characterization

We have seen from the Section 3.1 that online videos can be characterized by a number of basic characteristics such as bitrate, framerate, codec and resolution. We have also shown their distribution as obtained from random sampling of YouTube videos in [23, 24]. To characterize an input video we collect all features listed in Table 3.1. This list of video characteristic features are selected through expert analysis of the problem and the insight gained from the dataset presented in Section 3.1 and [23, 24].

Characteristics	Description
Codec	Coding standard used
Resolution(W,H)	Width and height in pixels
Bitrate	Bits processed per second
Framerate	Frames per second
Frmaes (I,P,B, Total)	Number of frames per type
Size (I,P,B, Total)	Size in byte per type

Table 3.1: Video characteristics features.

Figure 3.2 shows the effect of some of the input and output video characteristics listed in Table 3.1 and Table 3.2 on the transcoding time of 20 second video segments from randomly sampled online YouTube videos from our dataset. The figure shows the relationship between the dependent and independent variables under a controlled experiment where only one independent variable (i.e. predictive feature) is monitored while the rest are kept constant. Even though the plots show the correlation between our independent variables and transcoding time, the relationship is often non-linear. This means we either need to pre-process (be able to apply a proper transformation) of our predictive features to achieve a linear relationship with our target variable or use a non-linear learning algorithm to build our prediction models.

In this thesis we choose the later as 1) our primary aim is to predict rather than explain and 2) as the number of our predictive features are considerable due to the need to support multiple video coding algorithms [89, 90].

3.2.3 Building and Using Prediction Model.

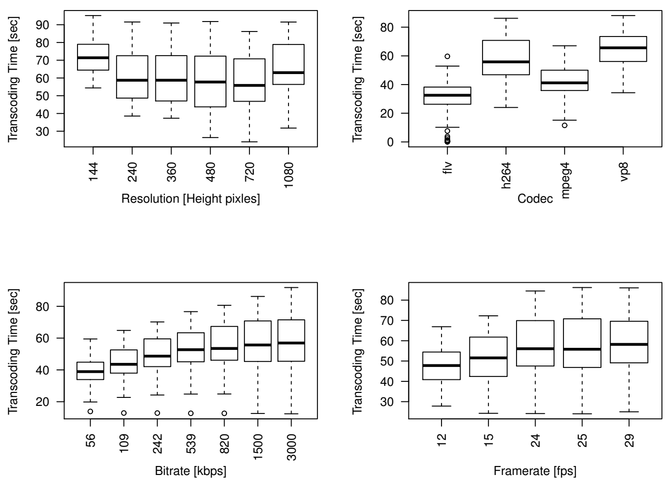
Collecting Transcoding Time Table 3.2 gives the list of the transcoding parameters that are used to create our transcoding search space. When considering all the possible combination of all transcoding options, the transcoding space can lead to a large set of possibilities. We reduced the search space size using expert pruning based on the insight gained from the online video characterization results presented in Section 3.1 and [23, 24], but we still have 840 of combinations to consider, see Table 3.2. We then collected the transcoding time obtained for each transcoding sequence on a set of 80 randomly selected 20 second YouTube videos. The resulting training data contains 67200 transcoding measurements (instances). All possible combinations of video transcoding parameters from Table 3.2 have been applied to each video when collecting the training data.

Parameter	Value
Codec	H264, Mpeg4, Vp8, H263
Resolution	144p, 240p, 360p, 480p, 720p, 1080p
Bitrate	56k,109k, 242k, 539k, 820k, 3000k, 5000k
Framerate	12, 15, 24, 25, 29.97

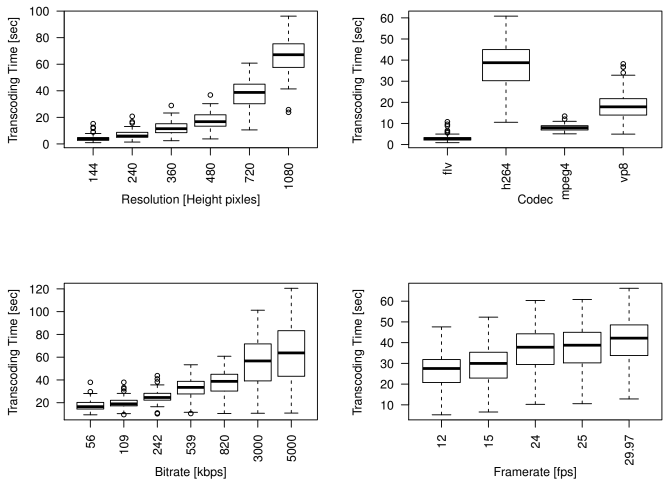
Table 3.2: Transcoding parameter space.

The process of collecting transcoding time for each sequence is shown in Figure 3.3. The characterization of each online video along with a transcoding parameter and the corresponding measured transcoding time makes up an instance of our training data. We used Ffmpeg and Ffprobe to collect video transcoding time and video characteristics of each video in our training set.

Constructing the Model Once the training data is constructed, it can be fed to a learning algorithm that will automatically learn a prediction model as shown in Figure 3.4. We used support vector regression (SVR), linear regression (LR) and multilayer perceptron (MLP) to construct our prediction models. Details of the setups used for these learning algorithms can be found in [23, 24].



(a) Effect of input video characteristics on transcoding time of a set of 80 Youtube videos when transcoding using a fixed transcoding parameter set. Transcoding parameters are fixed to 3000 kbps bitrate, 25 fps framerate, 720p resolution and h264 codec. In this case the transcoding parameters are fixed to show the effect of each input video characteristics on the total transcoding time.



(b) Effect of transcoding parameters on transcoding time of a set of 80 YouTube videos. The effect of each transcoding parameter is shown while keeping the other parameters fixed. Each box plot corresponds to transcoding time values of the 80 randomly selected YouTube videos. The set videos is fixed and all transcoding parameters except the controlled parameters is fixed.

Figure 3.2: Effect of input video characteristics and transcoding parameters.

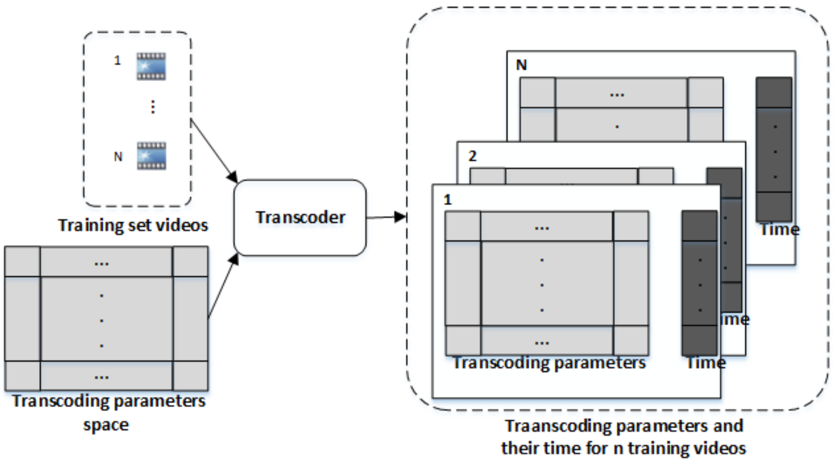

1
N
Training set videos
Transcoder
Transcoding parameters space
N
2
1
Time
Transcoding parameters
Traanscoding parameters and their time for n training videos

Figure 3.3: Collecting transcoding time.

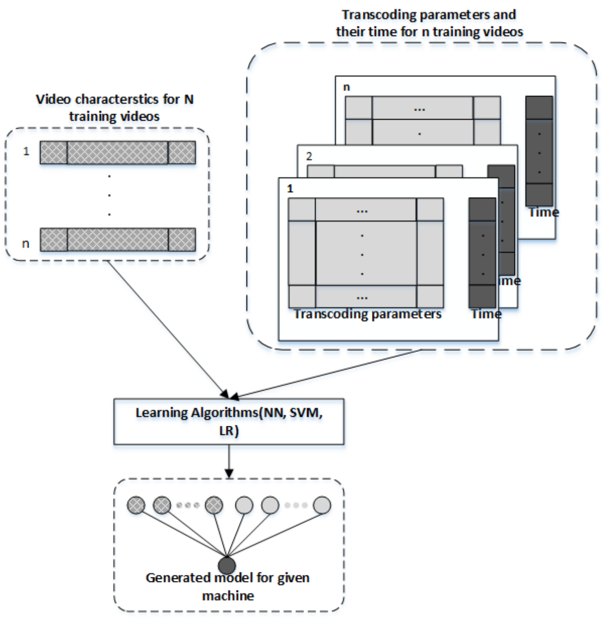

Transcoding parameters and their time for n training videos
Video characterstics for N training videos
1
n
Transcoding parameters
Time
Learning Algorithms(NN, SVM, LR)
Generated model for given machine

Figure 3.4: Constructing the model.

Model on Unseen Video Figure 3.5 illustrates how the model obtained from the learning algorithms can be used on an unseen video to get an estimation of the transcoding time of a video, which can be exploited by a load balancing algorithm on a large scale distributed video transcoding service.

We train and validate our models using 2/3 of the training data we have collected and the rest 1/3 portion is left for testing and evaluation of the models.

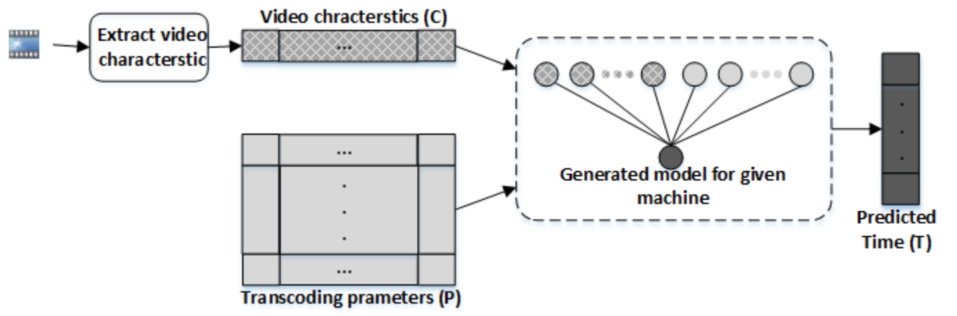

Extract video characterstic
Video chracterstics (C)
Generated model for given machine
Transcoding prameters (P)
Predicted Time (T)

Figure 3.5: Using model on unseen video.

3.3 Proactive Management of Transcoding Services

Our third contribution in this thesis is a proactive management approach for video transcoding in the cloud [25]. The proposed proactive resource management approach relies on the transcoding workload prediction approach presented in Section 3.2.

Figure 3.6 presents the system architecture of our proposed approach. It mainly consists of a predictor, a manager and transcoding servers. When transcoding requests are made to the system, they will be routed to the predictor which will predict the workload of the request before the actual transcoding. The requests, augmented with their workload prediction are then sent to the manager which consists of various sub-components that are important in decision making regarding efficient system utilization.

The main components that constitute the manager include the admission controller, the load balancer (dispatcher) and the provisioner. The admission controller is responsible for making the decision whether there is enough transcoding capacity at the moment to accept the request for a certain SLA (e.g. estimated response time). The load balancer is responsible for deciding which transcoding server to use for the request such that the waiting and time total utilization of the system is optimized. The provisioner is responsible for allocating and deallocating transcoding resources according to the total predicted workload of the system. Once the request is accepted it is routed to a selected transcoding node where it is transcoded, and all the information required by the predictor including the transcoding time are logged for retraining and learning a better prediction model.

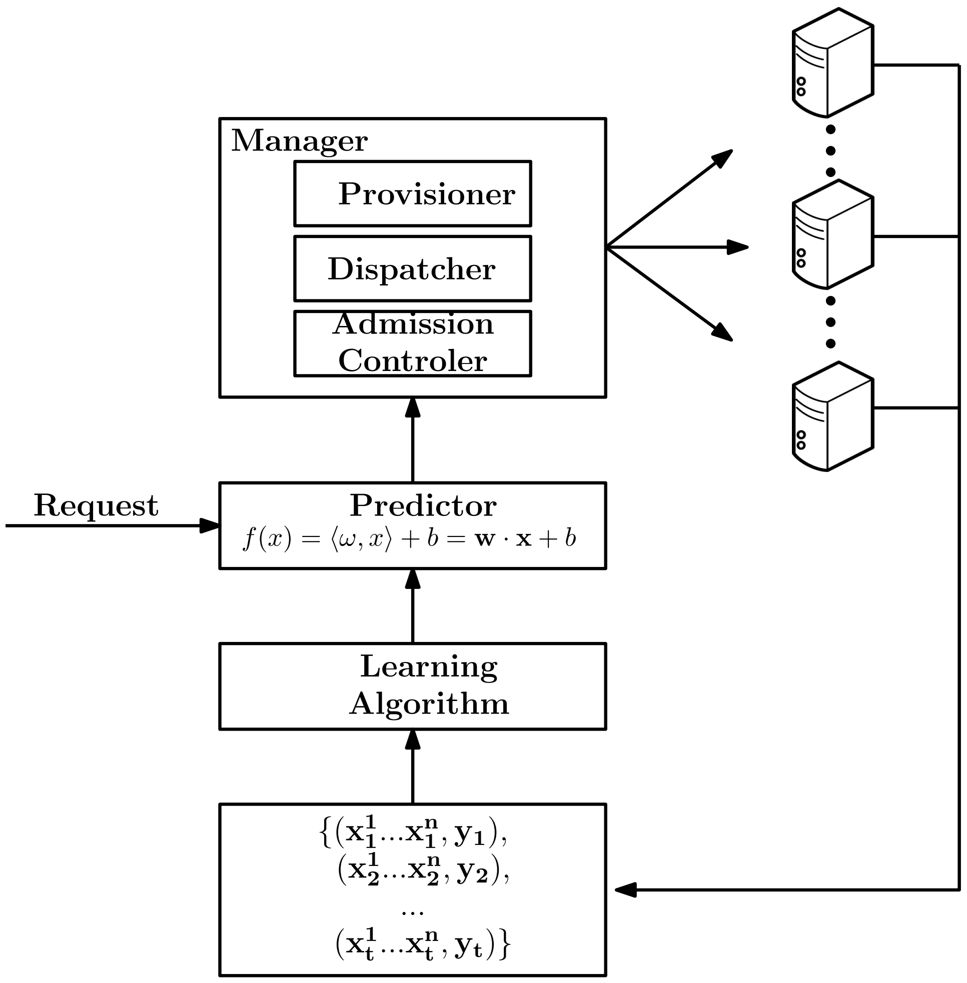

Manager
Provisioner
Dispatcher
Admission Controler
Request
Predictor
$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b$
Learning Algorithm
$\{(\mathbf{x_1^1 ... x_1^n, y_1}),$
$(\mathbf{x_2^1 ... x_2^n, y_2}),$
$...$
$(\mathbf{x_t^1 ... x_t^n, y_t})\}$

Figure 3.6: System Architecture.

In this thesis we particularly focus on proactive provisioning and load balancing and the following sections presents these topics briefly. Detail of our work on proactive provisioning and load balancing approaches can be found in [25, 32].

3.3.1 Proactive Load Balancing of Transcoding Jobs

Without a workload prediction mechanism, current large-scale video processing platforms use one of the two basic and widely used algorithms; queue length based load balancing where the load is being balanced based on queue length associated with each server or round robin approach where transcoding requests are routed to transcoding servers in a round robin fashion.

As can be noted from Algorithm 1 our load balancing approach uses

```
1: $server \leftarrow servers(0)$
2: for all $req$ in $requests$ do
3:     for all $s$ in $servers$ do
4:         if $predLoad(s) < predLoad(server)$ then
5:             $server \leftarrow s$
6:         end if
7:     end for
8:     $send(req, server)$
9:     $load(req) \leftarrow predictTranscodingTime(req, algo)$
10:    $predLoad(server) \leftarrow predLoad(server) + load(req)$
11: end for
```

Algorithm 1: Proactive load balancing algorithm

the transcoding time prediction as a load measure rather than queue length which often is not indicative of the actual workload. In a nutshell, what our algorithm does is to look for a server with the least total predicted load and send the current transcoding request to it.

3.3.2 Proactive Provisioning of Transcoding Servers

Due to the variability of individual video transcoding jobs and the variable transcoding requests made towards transcoding services, it is important to have a mechanism for proactive provisioning of transcoding servers. Without such mechanism, video transcoding services would have to resort to a fixed computing capacity which leads to either over-provisioning or under-provisioning of computing resources for a required level of Quality of Service.

In this thesis, we contribute towards a proactive provisioning approach based on ahead of time transcoding workload prediction. Our approach calculates the right required amount of transcoding servers based on the formula:

$$\left(\left\lceil \frac{predLoad(servers)}{\alpha * slaWaitingTime} \right\rceil - 1\right) * len(servers) \tag{3.1}$$

where $predLoad(servers)$ is the average predicted load of servers, $slaWaitingTime$ is the required system-wide average waiting time for transcoding jobs and α is an adjusting parameter that can be used to compensate prediction errors. The average predicted load of servers for a given period is calculated by summing up workload predictions of all transcoding tasks assigned to servers and dividing that by the number of servers at the same period. Averages from past and current period are then smoothed using exponential moving average technique and are used by simple regression model to predict the average predicted load of the next period. The average predicted load is used in our formula for calculating the number of VMs.

The formula calculates the number of servers to add or remove in order for the average load on each server get as close as possible to the waiting time specified by the SLA of the service. This ensures provisioning the right amount of servers for the Quality of Services agreed. Note that a negative result from the formula indicates the number of servers that needed to be removed while a positive number indicates the number of servers that needed to be added.

Since most cloud providers that host transcoding services charge for their virtual machines on an hourly basis, the provisioning algorithm checks the servers renting time before removal. Servers are removed only if they are near to the end of their renting time and has no job under processing. Servers that have running jobs and are near to the end of their renting time are flagged. A flagged server will not receive any more jobs and until it is removed later or become unflagged due to system load increase in which case it's renting time will is renewed. Similarly when the system needs to add servers, it will start first by unflagging flagged servers. This allows the servers to receive new jobs and their renting time be renewed. Therefore the algorithm provisions new servers only when there are no flagged servers at disposal.

3.4 Parallelization of Transcoding

Beside properly managing computing resources of a given distributed video processing platform, one needs to parallelize the applications that run on top of it to further ensure efficient utilization of modern multi-core architectures used to build the infrastructure. Therefore, as our forth contribution of this thesis, we present a parallelization approach for a video transcoding application. The approach shows how a high performance distributed video transcoder can be built using multiple processing units and a Message Passing Interface programming model. The problem decomposition and process interaction approaches used are briefly presented in the following subsections and the details of the approach are presented in [26].

3.4.1 Problem Decomposition

A video stream consists of several independent units called as video sequences where every sequence has its own headers. The video sequence consists of several groups of pictures (GOP). The group of pictures consists of frames. There are different types of frames; I (Intra) frame, P (Predictive) frame and B (Bidirectional) frame. The frame is further divided into slices; each slice consists of macro blocks and every macro block consists of blocks. Figure 3.7 shows the video stream structure down to the MB level.

At each of these levels, there is a potential for data level parallelization of any video processing application including video transcoding. In our case,

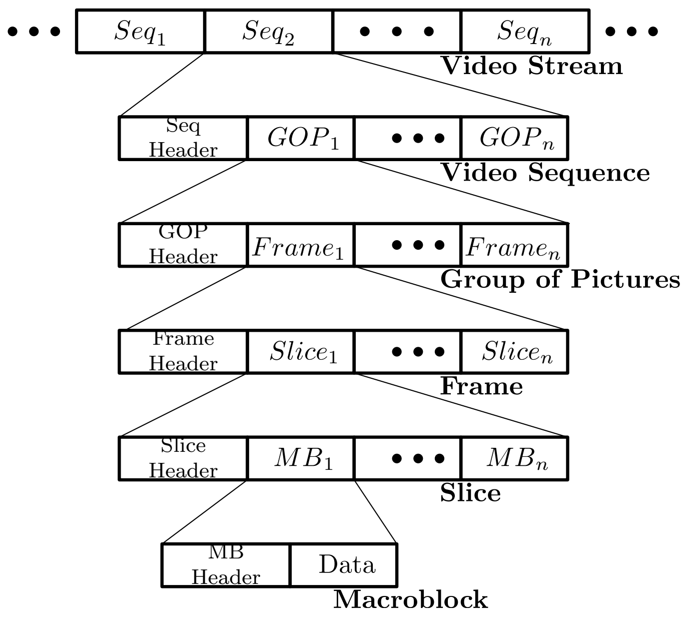


Figure 3.7: Video Stream Structure.

we use GOP level data parallelization where a video stream is segmented into parts containing one or more GOPs. Such course grain parallelism at GOP level is interesting because VODs often used pseudo streaming techniques such as DASH [91]. This makes parallel video processing at GOP even more attractive. Further fine-grained parallelism and approaches for implementing them are discussed in the next section as our next contribution of this thesis.

3.4.2 Process Interaction

Our parallel transcoder is implemented based on Message Passing Interface (MPI) and a master worker design pattern. The parallel transcoder consists of a manager node and a set of worker nodes. The manager node is responsible for splitting, merging and assigning video segments to worker nodes. A worker node runs a full video transcoder and its task is to transcode video segments sent to it by the manager node. The master and the worker processes interact with each other based on tagged messages.

3.5 Interfacing Dataflow with Imperative Code

Implementing fine-grained parallelism using such programming model is more complicated and error prone as the programmer needs to explicitly express the parallelism in the program including managing and synchronization of processes which is required to avoid non-determinism. Recently more abstract programming models have been developed to enable programmers to express program parallelism implicitly. Dataflow is one model of computation that can be used to implicitly express program parallelism. In this section, as our last contribution, we introduce a new approach allowing the integration of dataflow components within a imperative code. The approach makes use of a generic interface definition that allows seamless interaction between I/O components and data processing components. I/O components are mostly state operations and are best implemented in imperative languages. On the other hand, data processing components are mostly stateless dataflow operations and are best implemented in dataflow languages.

Ideally we would need to find the most transparent way to accomplish the interfacing. It should be done in such a way that enables code in both languages to remain natural. There are two possible approaches to interface imperative code with dataflow code. One possible approach, adopted by current RVC-CAL application developments, is based on the interface driven by the dataflow code. The I/O imperative functions are called from the dataflow program using CAL procedures which are ad-hoc mechanisms put in place to accommodate imperative code into dataflow components.

The approach proposed in this thesis is also based on the use of an imperative code for I/O. However, in this approach it is the imperative code that calls the dataflow code with an input data to be processed. The main advantage of this approach is the ease of development. Each language is used to implement those parts of the code for which the language is most appropriate for, without the need to accommodate the languages to each other (as in RVC-CAL procedures). This enables independent prototyping and reuse of code already written. It also permits existing libraries to access dataflow components and helps adoption of dataflow programming.

3.5.1 Interface Definition

The interface is designed to be generic enough such that any new dataflow component, like a decoder, encoder or filter, can be added to any existing or legacy video processing library written in an imperative language. The proposed interface consists of three functions and a data structure which are explained and implemented as follows.

init_component

This interfacing function is used to launch the dataflow component. Launching a dataflow component can be conceived as starting a conveyor belt system in a factory. Once a conveyor belt along with the processing units connected to it are started, a factory is ready to receive a stream of items to be processed. In our case, the initialization function starts the dataflow runtime system. The RVC-CAL runtime mainly consists of mapping, scheduling and other utility routines. The runtime takes in a set of actors, their network and user-supplied parameters such as an input data and maps, schedules and executes actors on a number of processing units. Mapping is either done by simply assigning actors to available processing units in a round-robin manner or via more complex post-profiling weight-based methods [92]. Once actors are mapped to a processing unit, they are scheduled using round-robin or more advanced data-driven methods [93].

```
Data: context
Result: success
1 set_component_context($context$);
2 success = thread_create($launcher, launch, context, tid$);
```

Algorithm 2: Dataflow component initialization

The pseudocode in Algorithm 2 shows the implementation of the *init_ component* function. *Context* is any information that is needed to start the component. It contains the runtime options of the dataflow component such as mapping policy, scheduling policy, the number of cores to be used and the dataflow network itself. Once the desired *context* of the component is set, the dataflow component is launched with a new thread which ensures the *init_ component* function returns control to the imperative code immediately. This allows the caller to continue its execution by sending row data and receiving processed data from the dataflow component.

process

This function is responsible for feeding the already initialized dataflow component with input data and grabbing the output data if available. Every call to this function from the imperative transcoder might fill the input FIFO of the dataflow component or get tokens from the output FIFO of the component, or both.

As shown in Algorithm 3, the *process* function takes in the *context* of the dataflow component which contains the network information and the data to be processed in *ipkt*. It then returns any result that might be available from the dataflow component in *opkt*. Note that this function also returns the size of the data consumed by the component through the variable *sent*. Any unconsumed data on the input FIFO of the dataflow component should be re-supplied to this function. *got_ result* is used to tell a calling function if the dataflow component resulted in a valid output via *opkt*.

```
Data: context, ipkt
Result: sent, opkt, got_result
1  tosend=ipkt.size;
2  sent = 0;
   /* send input to dataflow                                   */
3  sent += send(context,ipkt, tosend);
4  last_processed=processed;
   /* recive processed data                                    */
5  processed += recv(context,opkt);
6  if $last\_processed < processed$ then
       /* we have got data                                     */
7      got_result = 1;
8  else
9      got_result = 0;
10 end
11 ipkt.size -= sent;
12 ipkt.data += sent;
```

Algorithm 3: Dataflow processing

close_component

This interface function is used to end the already running dataflow component. More specifically it ends the runtime of the component by joining all created threads that were responsible for executing the component's actors.

```
Data: context
Result: success
1  thread_join(launcher);
2  success = free_component_context(context);
```

Algorithm 4: Dataflow component termination

Algorithm 4 shows the *close_ component* function.

Component Structure

This structure definition enables the use of multiple dataflow components and ensures the generic nature of the interface.

As can be noted from Algorithm 5, the component definition allows multiple dataflow components to be identified by a name or id.

```
typedef struct component {
    const char name;
    enum type component_type;
    enum id component_id;
    struct component *next;
    int (*init_component)(context *);
    int (*process)(context *, opkt*, ipkt*, *got_result);
    int (*close_component)(context *);
}
```

Algorithm 5: Component Structure

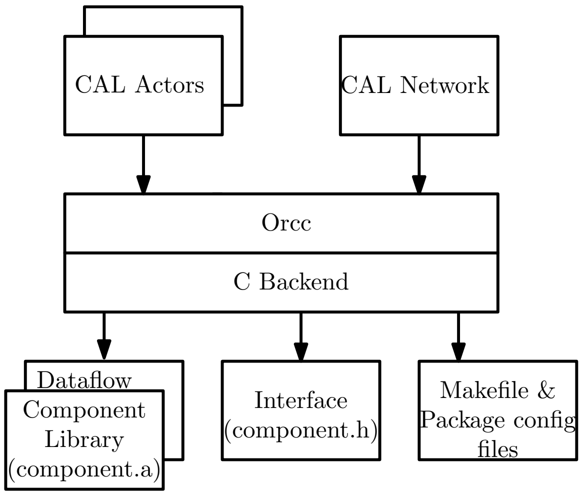

CAL Actors
CAL Network
Orcc
C Backend
Dataflow Component Library (component.a)
Interface (component.h)
Makefile & Package config files

Figure 3.8: Generation of the dataflow component library and its interface.

It also contains pointer to the three functions that are used to initiate, use and close a given dataflow component.

3.5.2 Generating Interface and Dataflow Component

To have an automated workflow for integrating dataflow components into a transcoding framework, we propose generating the interface automatically from the ORCC backend.

As shown in figure 3.8 we have modified the ORCC C backend to generate the dataflow components as a library along with a header file instead of stand alone executable. In addition, we have added generation of package configuration files so that the library can be installed and be used easily.

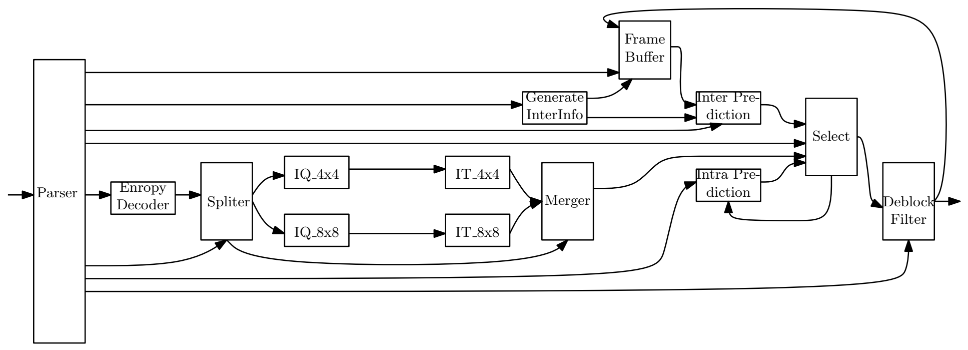


Figure 3.9: Dataflow Decoder Implementation from ORCC.

3.5.3 Using the Interface

To demonstrate the functionality of the approach we have generated MPEG and HEVC video decoders form the corresponding dataflow descriptions, written in RVC-CAL and obtained from [20], using our modified C backend. Figure 3.9 shows the structure of the HEVC dataflow decoder.

The dataflow decoder descriptions are then compiled into libraries with proper interfaces and are installed in our system. Following that we have included the header files of the generated dataflow components into an existing video transcoding framework written in a procedural language and linked to the installed component libraries during its compilation.

Algorithm 6 shows the use of a dataflow decoder component by the existing video transcoder FFmpeg written in a procedural language. Three main points can be noted from the pseudo-code. The first is the *register_ all* function that is used to register the available formats, codec and filters in a given system. This function is from FFmpeg libraries and we have also used it to register our new dataflow decoder component. The second point to note is the use of our interface which constitutes the three functions, *init_ component, process and close_ component.* This interface can be used to abstract the various types of dataflow components that can be implemented and integrated to FFmpeg or any other transcoder. Finally one can note that the FFmpeg libraries supply the I/O (read/write) functions which are capable of parsing almost any known video container format efficiently. In addition to the I/O functions, data processing functionalities such as video scalers and encoders that are yet to be implemented by dataflow approach can also be used.

Using our interface we were, therefore, able to provide FFmpeg, an existing video transcoding framework written in an imperative language with dataflow components that implement fine-grained parallelism. Note from Figure 3.9 that the dataflow decoder component provides a fine-grained parallelism by implementing functional blocks as separate actors.

```
Data: $v_s$ (source video), context
Result: $v_p$ (processed video)
register_all();
init_component(context) ;            // initialize component e.g.
  decoder/filter/encoder
while read(context, $v_s$, ipkt) do
    process(context, opkt, ipkt, got_result) ;       // e.g.
      decode/filter/encoder
    if got_result then
        rescale_frame (context, fpkt, opkt, got_result);
    end
    if got_result then
        encode_frame (context, opkt, fpkt, got_result);
    end
    if got_result then
        write(context, opkt);
    end
    ipkt.data += ret;
    ipkt.size -= ret;
end
flush();
close_component(context) ;           // close component e.g.
  decoder/filter/encoder
```

Algorithm 6: FFmpeg overview

Chapter 4

Overview of Original Publications

In this chapter we present a summary of our original publications, the contribution of the authors towards these publications, the relationship among the publications and with the research questions posed in Section 1.2.

4.1 Overview of Original Publications

4.1.1 Paper I: Analysis and Transcoding Time Prediction of Online Videos

Paper I presents an analysis of the fundamental characteristics of online videos and their application in multimedia processing workload modelling. Its main contribution is to provide insightful statistics and a dataset on fundamental characteristics of online videos that can be further exploited to optimize or model components of a multimedia processing systems. The usefulness of dataset along with its summary statistics is demonstrated through its use in modelling and prediction of video transcoding time. The outputs of this research thus include a dataset, a tool used to extract it and a method of modelling transcoding time based on the dataset.

Author's contribution: The main idea presented in this paper was developed by the under the guidance of Dr. Sébastien Lafond and Professor Johan Lilius. Tewodros Deneke is the primary author of this paper. The tools used to extract the dataset and build the transcoding prediction model are also developed by Tewodros Deneke. The dataset presented in the paper can be found in [83].

4.1.2 Paper II: Video Transcoding Time Prediction for Proactive Load Balancing

Paper II presents the details of our transcoding time prediction method and its use for proactive load balancing of transcoding jobs across VMs in the cloud. Its main contribution is thus a proactive load balancing approach that utilizes transcoding prediction models built based on past observations and generic learning algorithms. The results obtained in this paper are based on an extended version of CloudSim, an open source discrete event cloud simulator. In the paper, we also provided a comparison of our approach with existing basic load balancing algorithms.

Author's contribution: The main idea presented in this paper was developed by the author under the guidance of Dr. Sébastien Lafond and Professor Johan Lilius. Tewodros Deneke is the main author of this paper. The extensions to the cloud simulator are also developed by Tewodros Deneke. Co-author Habtegebreil Haile made the initial setup of the simulator and provided ideas on how the required extensions can be integrated.

4.1.3 Paper III: Integration of Dataflow Components Within a Legacy Video Transcoding Framework

Recently the RVC-CAL dataflow language has enabled video codecs to be specified in a more natural way than imperative languages by allowing implicit expression of parallelism and side-effect-freeness. The tools developed for RVCCAL have also enabled the automatic generation of parallel C code, among others, from dataflow specifications. The main contribution of paper III is thus the introduction of a new approach allowing the integration of dataflow components within an existing or legacy. The approach makes use of a generic interface definition that allows seamless interaction between I/O components and data processing components. I/O components are mostly state operations and are best implemented in imperative languages. On the other hand, data processing components are mostly stateless dataflow operations and are best implemented in dataflow languages. The advantage of the approach is the ease of development by allowing each language to be used on those parts of the application that it is most appropriate for. The functionality of the approach is demonstrated[1] through using the automatically generated generic interface to add new dataflow based MPEG and HEVC decoders into an existing video transcoding library FFmpeg.

Author's contribution: The main idea presented in this paper was developed by the co-authors Tewodros Deneke and Dr. Lionel Morel under the guidance of Dr. Sébastien Lafond and Professor Johan Lilius. Tewodros

[1]`https://github.com/tdeneke/ffmpeg-2.5.3`, `https://github.com/tdeneke/orcc` and `https://github.com/tdeneke/orc-apps`

Deneke is the main author of this paper. The compiler extensions and the generic interface were designed and developed by Tewodros Deneke.

4.1.4 Paper IV: Bit Rate Reduction Video Transcoding with Distributed Computing

Paper IV presents an approach to perform a distributed video bit rate reduction transcoding. The approach is based on video segmentation at group of pictures level and message passing programming model for inter-process interaction. The main contribution of the paper is thus to show how a high performance distributed video transcoder can be built using multiple processing units and a Message Passing Interface programming model. Performance and scalability results are presented based on a real implementation of the approach. Also, comparison of the different group of pictures level segmentation strategies and their effect on the performance of the approach is presented.

Author's contribution: The main idea presented in this paper was developed by the co-authors Fareed Jokhio and Tewodros Deneke under the guidance of Dr. Sébastien Lafond and Professor Johan Lilius. Fareed Jokhio is the main author of this paper. The paper is written jointly by co-authors Fareed Jokhio and Tewodros Deneke. The implementation is mostly done by Tewodros Deneke and the experiments are done by Fareed Jokhio.

4.1.5 Paper V: Proactive Management of Video Transcoding Services

Paper V ties together most of the contributions in this thesis. It presents a proactive transcoding service management approach based on transcoding task size prediction to optimize the usage of cloud platforms. The proposed approach uses machine learning based task size prediction to enable more efficient resource management in terms of auto-scaling and load balancing. It also allows for a clear definition of service level agreement (SLA) in terms of average waiting time of transcoding jobs. Simulation results show that our proactive transcoding service management methods based on transcoding task prediction enable a significantly better resource utilization for a given quality of service.

Author's contribution: The main idea presented in this paper was developed by the author under the guidance of Dr. Sébastien Lafond and Professor Johan Lilius. Tewodros Deneke is the main author of this paper. The tools and simulation programs[2] used in this work are all developed by Tewodros Deneke.

[2]`https://github.com/tdeneke/cloudsim/`

4.2 Discussion

In section 1.2, we have presented four research questions this thesis tries to address. The first research question is concerned with understanding the current state of videos in large scale video on demand systems. The question is raised due to the need to understand, model and optimize video processing workload on such systems. This question is addressed in Paper I. Paper I provide an insightful descriptive statistics on the various video characteristics based on a dataset mined from a large scale video on demand system, Youtube. More summary statistics and associated insights than covered in the first paper can be found in [24].

The second research question is concerned with the possibility of accurately predicting computational workload of video processing applications such as video transcoding. Parts of Paper I and V present how video transcoding time can be modeled and predicted based on historical data and generic learning algorithms.

The third research question addressed in this thesis is concerned with the design of novel resource management algorithms driven by the workload prediction mechanisms. Paper II addresses this research question by demonstrating how video transcoding workload can be efficiently distributed across VMs such that the total utilization of a distributed transcoding system can be maximized.

Our final research question of the thesis is concerned with the possibilities of parallelizing video processing applications to ensure efficient utilization of modern multi and many core architectures that make up the cloud and distributed computing infrastructures. To this end, Paper III provides one possible solution that allows fine-grained parallel implementations of video processing components in dataflow languages. Paper IV addresses the possibility of coarse-grained parallelization approaches for video transcoding applications.

Figure 4.1 illustrates how the papers found in part II correspond to the various parts of the system architecture. Papers I, II and V are related to understanding video processing workload and efficient management of computing resources based on workload predictions. Papers III and IV are related to efficient utilization of computing resources that make up large-scale cloud and distributed computing platforms. Paper I provide a summary and descriptive statistics that give insight on the current state of online videos and show how the insight can be used to model workloads of large-scale video processing platforms. An extended summary and explanation of how to model video transcoding workload can be found [24]. Paper II extends the work of Paper I and IV to provide a proactive load balancing and management approach for cloud based distributed transcoding services.

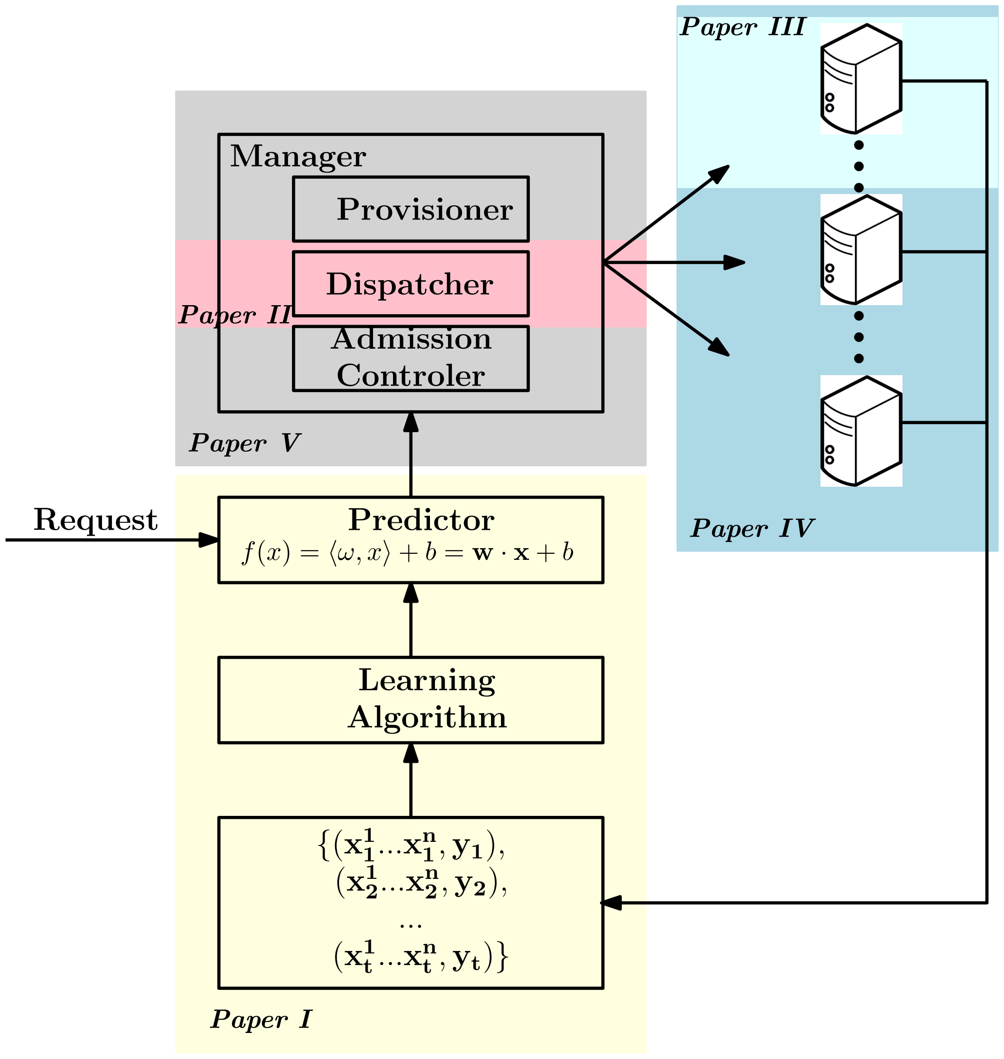

Paper III
Manager
Provisioner
Dispatcher
Paper II
Admission Controler
Paper V
Request
Predictor
$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b$
Paper IV
Learning Algorithm
$\{(\mathbf{x_1^1}...\mathbf{x_1^n}, \mathbf{y_1}),$
$(\mathbf{x_2^1}...\mathbf{x_2^n}, \mathbf{y_2}),$
$...$
$(\mathbf{x_t^1}...\mathbf{x_t^n}, \mathbf{y_t})\}$
Paper I

Figure 4.1: Illustration of how the papers correspond to the various parts of the system architecture

Paper IV provides a course grained parallelization approach for video transcoding while Paper III shows how to facilitate the use of dataflow programming approaches for fine grained parallelization of video processing approaches in the context of video transcoding. Paper V ties most of the contributions together and provide methods such as provisioning and load balancing to manage computing resources in video transcoding services proactively.

Chapter 5

Conclusions and Future Work

The primary focus of this thesis work was the different research issues related to proactive management of transcoding services and transcoding parallelization. To this end, we have systematically collected a dataset containing a set of relevant video characteristics and presented a descriptive statistics over them. The insight gained from these statistics enables proper design and test of large-scale video processing services such as video transcoding. The dataset was obtained through random sampling of the most popular video UGC site, YouTube. Such random sampling enabled us to construct an unbiased dataset. The data collection was done through extending open source tools and libraries such as Ffmpeg. Our benchmark results showed how the datasets can be used in constructing models to predict video processing workloads with a very good accuracy. A squared correlation (predicted vs measured) of 0.958 has been achieved by the best prediction model on 20-second video segments.

The next challenge we have addressed was the design of proactive computing resource management algorithms such as load balancing and provisioning. We designed a proactive load balancing scheme to improve throughput and quality of service of video transcoding services. Our method explored the opportunities provided by trends obtainable from transcoding workload logs of past requests. We were able to predict transcoding time of a video given its fundamental characteristics and the transcoding parameters used. Based on such prediction can we were able to design a proactive load balancing approach that can be used to properly load balance video transcoding requests across servers. In our experiment, we have used real-world data and designed the simulation scenario imitating real world Internet transactions. Our proactive load balancing algorithm showed a significant improvement over the traditional methods in terms of total system throughput and waiting time. More specifically up to 15% improvement has been achieved in terms of system throughput over the round robin and queue length approaches on a fixed server capacity. Also, a proactive provisioning method that enables provisioning of the right amount of transcoding servers for a given quality of service was presented and evaluated. Experiment has shown that using proactive provisioning rather than fixed worst case based provisioning saves up to (86%) in terms of VM hours over the course of a week. Also up to 8% in VM hours has been shown to be saved when augmenting proactive provisioning with proactive load balancing.

Another dimension towards efficient computing utilization besides efficient management is parallelization applications. In this thesis, work we explored parallelization opportunities at both coarse-grained and fine-grained level. First, we have proposed a scalable distributed MPI based transcoder implementation. In this implementation, a master node (workstation) partitions a given input video file into segments and distributes them among worker nodes. The actual transcoding is performed by worker machines in parallel to get more speedup of overall transcoding process. The worker machines send back the transcoded video to master for merging. We were able to see a considerable performance gain with MPI based transcoder as the number of worker machines or cores increases. It was observed that the segmentation with an equal number of Intra frames is more efficient than equal size segmentation. The unequal size segmentation is better for having short startup time. Then, in order to address the need for fine-grained parallelism, we have proposed the use of a generic interface for integrating dataflow components such as decoders, encoders and filters with existing transcoding libraries written in imperative languages. The interface enables seamless interaction between dataflow and existing imperative code allowing each programming approach to implement components for which it is appropriate for. We have also tested and shown the proper functionality of our approach. Scalability evaluations also show the gain that can be obtained from using dataflow components via the proposed interface. In the future we would like to further explore the effect of the dataflow component runtime and transcoding framework runtime on each other.

As a future work we encourage other researchers to use our dataset in discovering more interesting information about video characteristics on the web and test their own algorithms to build similar predictive models [83, 23]. It is also good to incorporate more video data sources which could help in building more insight, especially on niche sources. In this thesis we gave more attention towards prediction accuracy when modelling our workload prediction algorithms, in the future researchers could also look towards modelling for understanding. In this thesis load balancing has been used to highlight the use of data to drive management of system resources, however, other components such as resource provisioner could use similar approaches to be come proactive and effective. In the direction of multimedia application parallelization researchers could more deeply look at the benefits and drawbacks of the various programming models in order to tailor programming models for multimedia applications that makes it easy for programmers to utilize all system resources.

Bibliography

[1] Zencoder Inc. Zencoder cloud transcoder, August 2013. URL `http://zencoder.com/en/`.

[2] Amazon Inc. Amazon elastic transcoder, August 2013. URL `http://aws.amazon.com/elastictranscoder/`.

[3] Bitmovin Inc. Bitmovin cloud encoding, May 2016. URL `https://bitmovin.com/encoding/`.

[4] Shih fu Chang, Shih fu Chang, and Anthony Vetro. Video adaptation: Concepts, technologies, and open issues. *Proceedings of IEEE*, January 2005.

[5] Jun Xin, Chia-Wen Lin, and Ming-Ting Sun. Digital video transcoding. *Proceedings of the IEEE*, January 2005.

[6] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, October 2011.

[7] Thomas Stockhammer. Dynamic adaptive streaming over HTTP –: Standards and design principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[8] Laura Toni, Ramon Aparicio-Pardo, Gwendal Simon, Alberto Blanc, and Pascal Frossard. Optimal set of video representations in adaptive streaming. In *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, pages 271–282, New York, NY, USA, 2014. ACM.

[9] Hongliang Yu, Dongdong Zheng, Ben Y. Zhao, and Weimin Zheng. Understanding user behavior in large-scale video-on-demand systems. *SIGOPS Oper. Syst. Rev.*, April 2006.

[10] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. IMC '07, San Diego, California, USA.

[11] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. IMC '07, San Diego, California, USA.

[12] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. Watch global, cache local: YouTube network traffic at a campus network - measurements and implications. Technical report, 2008.

[13] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, page 29–42, New York, NY, USA, 2007. ACM.

[14] Jiani Guo and Laxmi Narayan Bhuyan. Load balancing in a cluster-based web server for multimedia applications. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1321–1334, November 2006.

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

[16] Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2012.

[17] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei"i Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[18] Fareed Ahmed Jokhio, Adnan Ashraf, Sebastien Lafond, Ivan Porres, and Johan Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *PDP, 2013*, 2013.

[19] Adnan Ashraf, Fareed Jokhio, Tewodros Deneke, Sebastien Lafond, Ivan Porres, and Johan Lilius. Stream-based admission control and scheduling for video transcoding in cloud computing. In Pavan Balaji, Dick Epema, and Thomas Fahringer, editors, *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 482–489. IEEE Computer Society, 2013.

[20] Orcc. Open RVC-CAL compiler, 2009. URL `http://orcc.sourceforge.net/`.

[21] Matthieu Wipliez. *Compilation infrastructure for dataow programs.* PhD thesis, INSA Rennes, September 2010.

[22] H Yviquel. *From Dataflow-based Video Coding Tools to Dedicated Embedded.* PhD thesis, UNIVERSITE DE RENNES 1, October 2013.

[23] T. Deneke, S. Lafond, and J. Lilius. Analysis and transcoding time prediction of online videos. In *2015 IEEE International Symposium on Multimedia (ISM)*, pages 319–322, Dec 2015.

[24] Tewodros Deneke, Sebastien Lafond, and Johan Lilius. Analysis and transcoding time prediction of online videos. Technical Report 1145, 2015.

[25] T. Deneke, H. Haile, S. Lafond, and J. Lilius. Video transcoding time prediction for proactive load balancing. In *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, July 2014.

[26] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius. Bit rate reduction video transcoding with distributed computing. In *PDP, 2012 20th Euromicro International Conference*, February 2012.

[27] Fareed A. Jokhio, Tewodros Deneke, Sébastien Lafond, and Johan Lilius. Analysis of video segmentation for spatial resolution reduction video transcoding. In *ISPACS, 2011 International Symposiuml*, December 2011.

[28] T. Deneke, L. Morel, S. Lafond, and J. Lilius. Integration of dataflow components within a legacy video transcoding framework. In *Signal Processing Systems (SiPS), 2015 IEEE Workshop on*, pages 1–6, Oct 2015.

[29] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. HD-VideoBench. A benchmark for evaluating high definition digital video applications. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 120–125, 2007. doi: 10.1109/IISWC.2007.4362188.

[30] MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE*, MICRO 30, page 330–335, Washington, DC, USA.

[31] Nathan T. Slingerland and Alan Jay Smith. Design and characterization of the berkeley multimedia workload. *Multimedia Syst.*, 8(4):315–327, July 2002.

[32] Tewodros Deneke, Sebastien Lafond, and Johan Lilius. Proactive management of video transcoding services. *ACM Trans. Multimedia Comput. Commun. Appl.*, August Submitted.

[33] P. Järvinen. *On research methods.* Opinpajan Kirja, 2001.

[34] A.L. Strauss and J.M. Corbin. *Basics of qualitative research: grounded theory procedures and techniques.* Sage Publications, 1990.

[35] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, December 1995.

[36] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004.

[37] Shih fu Chang, Shih fu Chang, Anthony Vetro, Anthony Vetro, and Senior Member. Video adaptation: Concepts, technologies, and open issues. In *Proc. IEEE*, pages 148–158, 2005.

[38] Jun Xin, Chia-Wen Lin, and Ming-Ting Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1):84–97, 2005.

[39] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud consumputing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009.

[40] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10, Nov 2008.

[41] Simson Garfinkel and Harold Abelson. *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at Mit.* MIT Press, Cambridge, MA, USA, 1999.

[42] L. Kleinrock. A vision for the Internet. *ST Journal for Research*, 2(1): 4–5, November 2005.

[43] Charlie Catlett. The philosophy of teragrid: Building an open, extensible, distributed terascale facility. In *Proceedings of the 2Nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 8–, Washington, DC, USA, 2002. IEEE Computer Society.

[44] Dennis Gannon2005, Beth Plale, Marcus Christie, Liang Fang, Yi Huang, Scott Jensen, Gopi Kandaswamy, Suresh Marru, Sangmi Lee

Pallickara, Satoshi Shirasuna, Yogesh Simmhan, Aleksander Slominski, and Yiming Sun. *Service-Oriented Computing - ICSOC 2005: Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005. Proceedings*, chapter Service Oriented Architectures for Science Gateways on Grid Systems, pages 21–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[45] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, February 2005.

[46] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[47] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[48] Adriana Garcia, Hari Kalva, and Borko Furht. A study of transcoding on cloud environments for video content delivery. In *Proceedings of the 2010 ACM Multimedia Workshop on Mobile Cloud Media Computing*, MCMC '10, pages 13–18, New York, NY, USA, 2010. ACM.

[49] Seungcheol Ko, Seongsoo Park, and Hwansoo Han. Design analysis for real-time video transcoding on cloud systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1610–1615, New York, NY, USA, 2013. ACM.

[50] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, pages 49–60, New York, NY, USA, 2015. ACM.

[51] Warren S. McCulloch and Walter Pitts. Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.

[52] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[53] Donald O. Hebb. Neurocomputing: Foundations of research. chapter The Organization of Behavior, pages 43–54. MIT Press, Cambridge, MA, USA, 1988.

[54] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, DTIC Document, 1971.

[55] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Magazine*, 27(4):12, 2006.

[56] . *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 1969.

[57] Science Research Council (Great Britain). *Artificial Intelligence: A Paper Symposium.* Science Research Council, 1973.

[58] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. Readings from the ai magazine. chapter Machine Learning: A Historical and Methodological Analysis, pages 400–408. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1988.

[59] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989.

[60] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[61] Martin J. Halvey and Mark T. Keane. Exploring social dynamics in online media sharing. WWW '07, Banff, Alberta, Canada.

[62] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, pages 49–60, New York, NY, USA, 2015. ACM.

[63] Laura Toni, Ramon Aparicio-Pardo, Gwendal Simon, Alberto Blanc, and Pascal Frossard. Optimal set of video representations in adaptive streaming. In *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, pages 271–282, New York, NY, USA, 2014. ACM.

[64] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K. Sitaraman. Optimizing the video transcoding workflow in content delivery networks. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, pages 37–48, New York, NY, USA, 2015. ACM.

[65] Kihwan Choi, K. Dantu, Wei-Chung Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a mpeg decoder. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 732–737, Nov 2002.

[66] Andy C. Bavier, A. Brady Montz, and Larry L. Peterson. Predicting mpeg execution times. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 131–140, New York, NY, USA, 1998. ACM.

[67] Marco Mattavelli and Sylvain Brunetton. *Real-time constraints and prediction of video decoding time for multimedia systems*, pages 425–438. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[68] Yicheng Huang, Samarjit Chakraborty, and Ye Wang. Using offline bitstream analysis for power-aware video decoding in portable devices. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, pages 299–302, New York, NY, USA, 2005. ACM.

[69] Yicheng Huang, An Vu Tran, and Ye Wang. A workload prediction model for decoding mpeg video and its application to workload-scalable transcoding. In *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, pages 952–961, New York, NY, USA, 2007. ACM.

[70] J. Kuang, D. Guo, and L. Bhuyan. Power optimization for multimedia transcoding on multicore servers. In *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, pages 1–2, Oct 2010.

[71] Arnaldo Azevedo, Ben Juurlink, Cor Meenderinck, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, Alex Ramirez, and Mateo Valero. *A Highly Scalable Parallel Implementation of H.264*, pages 111–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[72] Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez. *Parallel Scalability of H.264.*

[73] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl. Parallel scalability and efficiency of hevc parallelization approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1827–1838, Dec 2012.

[74] Jörn W. Janneck, Marco Mattavelli, Mickael Raulet, and Matthieu Wipliez. Reconfigurable video coding: A stream programming approach to the specification of new video coding standards. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, pages 223–234, New York, NY, USA, 2010. ACM.

[75] Ghislain Roquier, Matthieu Wipliez, Mickael Raulet, Jörn W. Janneck, Ian D. Miller, and David B. Parlour. Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 281 – 286, Washington, United States, October 2008.

[76] D. M. Barbosa, J. P. Kitajima, and W. Weira. Parallelizing mpeg video encoding using multiprocessors. In *Computer Graphics and Image Processing, 1999. Proceedings. XII Brazilian Symposium on*, pages 215–222, 1999.

[77] Bongsoo Jung and Byeungwoo Jeon. Adaptive slice-level parallelism for h.264/avc encoding using pre macroblock mode selection. *J. Vis. Comun. Image Represent.*, 19(8):558–572, December 2008. doi: 10.1016/j.jvcir.2008.09.004.

[78] Yasuo Sambe, Shintaro Watanabe, Dong Yu, Taichi Nakamura, and Naoki Wakamiya. High-speed distributed video transcoding for multiple rates and formats. *IEICE - Trans. Inf. Syst.*, E88-D(8):1923–1931, August 2005. ISSN 0916-8532. doi: 10.1093/ietisy/e88-d.8.1923. URL `http://dx.doi.org/10.1093/ietisy/e88-d.8.1923`.

[79] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, Dec 2012.

[80] Zhiqiang Tian, Jianru Xue, Wei Hu, Tao Xu, and Nanning Zheng. High performance cluster-based transcoder. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume 2, pages V2–48–V2–52, Oct 2010.

[81] Jiani Guo, Fang Chen, L. Bhuyan, and R. Kumar. A cluster-based active router architecture supporting video/audio stream transcoding service. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8 pp.–, April 2003.

[82] F. Lao, X. Zhang, and Z. Guo. Parallelizing video transcoding using map-reduce-based cloud computing. In *2012 IEEE International Symposium on Circuits and Systems*, pages 2905–2908, May 2012.

[83] T Deneke. Online video characteristics and transcoding time dataset data set, 2015. URL `https://archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset`.

[84] Jia Zhou, Yanhua Li, Vijay Kumar Adhikari, and Zhi-Li Zhang. Counting YouTube videos via random prefix sampling. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, page 371–380, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068851. URL `http://doi.acm.org/10.1145/2068816.2068851`.

[85] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and social network of YouTube videos. In *in Proc. of IEEE IWQoS*, 2008.

[86] Stefano Sabatini. *Ffprobe*. August 2013. URL `http://ffmpeg.org/ffprobe.html`.

[87] Ricardo Garcia Gonzalez. *youtube-dl*. September 2013. URL `http://rg3.github.io/youtube-dl/`.

[88] Amazon Inc. Amazon EC2 instance types, August 2013. URL `http://aws.amazon.com/ec2/instance-types/`.

[89] Galit Shmueli. To explain or to predict? *Statist. Sci.*, 25(3):289–310, 08 2010. doi: 10.1214/10-STS330.

[90] Leo Breiman. Statistical modeling: The two cultures. *Statistical Science*, 2001.

[91] I MPEG. Information technology-dynamic adaptive streaming over http (dash)-part 1: Media presentation description and segment formats. *ISO/IEC MPEG, Tech. Rep*, 2012.

[92] H. Yviquel, E. Casseau, M. Raulet, P. Jaaskelainen, and J. Takala. Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms. pages 732–737, Sept 2013.

[93] H. Yviquel, E. Casseau, M. Wipliez, and M. Raulet. Efficient multicore scheduling of dataflow process networks. In *SiPS*, pages 198–203, Oct 2011.

Part II

Original Publications

Paper I

Analysis and Transcoding Time Prediction of Online Videos

Tewodros Deneke, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the IEEE International Symposium on Multimedia, 2015 IEEE International Conference*, pages 319–322. Miami, Florida.

Analysis and Transcoding Time Prediction of Online Videos

Tewodors Deneke[1,2], Sébastien Lafond[1], Johan Lilius[1]
[1]TUCS - Turku Centre for Computer Science, Finland
[2]Åbo Akademi University, Finland
Email: firstname.lastname@abo.fi

***Abstract*—Today, video content is delivered to a myriad of devices over different communication networks. Video delivery must be adapted to the available bandwidth, screen size, resolution and the decoding capability of the end user devices. In this work we present an approach to predict the transcoding time of a video into another given transcoding parameters and an input video. To obtain enough information on the characteristics of real world online videos and their transcoding parameters needed to model transcoding time, we built a video characteristics dataset, using data collected from a large video-on-demand system, YouTube. The dataset contains a million randomly sampled video instances listing 10 fundamental video characteristics. We report our analysis on the dataset which provides insightful statistics on fundamental online video characteristics that can be further exploited to optimize or model components of a multimedia processing systems. We also present experimental results on transcoding time prediction models, based on support vector machines, linear regression and multi-layer perceptron feed forward artificial neural network.**

***Keywords*-Transcoding; Prediction; Measurement;**

I. Introduction

Since Internet clients may vary greatly in their hardware resources, software sophistication, and quality of connectivity, different clients require different media format from streaming services. Transcoding is a method which is used to convert one multimedia format to another [1], [2]. However different formats require different transcoding operations and, therefore, produce variety of individual transcoding jobs to be scheduled among computing servers. In order to provide efficient transcoding service, a good resource management algorithm needs to be employed to pro-actively predict the CPU load for each job.

In this paper, we first present a large-scale video characteristics dataset useful to assess, model and optimize the performance of large-scale video processing services such as video transcoding. The dataset contains fundamental video characteristics for YouTubevideos.We first crawled YouTube for a period of one and half months and obtained a video dataset containing over a million distinct videos. This dataset contains 10 features representing ten fundamental characteristics of videos such as duration, bitrate, framerate, resolution, etc. We then make a systematic and in-depth statistical analysis of these characteristics to identify useful statistics. Based on the insight and the statistics gained from the dataset we then build a video transcoding time prediction model.

II. Related Work

Among others our work relates to web video characterization and video transcoding time prediction. In this section, we briefly summarize the related work on each topic.

There have been significant researches on understanding the workloads of new generation video servers. These research especially focus on the social aspect of videos and traffic characterization such as popularity, active life span, user access pattern, growth pattern and request patterns. Yu et al. [3] study user behaviour, content access pattern and their implications on the design of large-scale video-on-demand systems. Possible improvements on UGC design were proposed by Cha et al. [4] after studying YouTube and Daum, a popular UGC in Korea. After tracking YouTube transactions from a network edge, Gill et al. [5] have studied YouTube traffic characteristics and discuss the implications of their observation on key concepts such as caching. The caching problem in YouTube have been further studied by Zink et al. [6]. The social networking among videos was studied in the works of Halvey et al. [7] and Mislove et al. [8]. Our work is however focused on collecting statistics of video characteristics such as video length, size, bitrate, frame rate, codec type and resolution rather than the social aspect and traffic characterization of such services.

A method to predict decoding and transcoding time of videos is presented by Roitzsch et al [9] and Jiani et al [10]. These works focus on one coding algorithm and used only a brief set of videos to train their prediction model. Our dataset provides such systems with a video dataset having a realistic distribution of video characteristics. It also works across multiple coding algorithms.

III. Detail of the Proposed Method

A. Crawling YouTube and Collecting Data

We collected our dataset systematically from YouTube, the largest user generated content (UGC) video website. By 2011 YouTube is estimated to have over half a billion videos and accounts for a significant percentage of the worlds bandwidth utilization [11]. YouTube organizes its videos as a directed graph, where each video is considered as a node and edges represent links to a set of related videos. There are several known ways of crawling YouTube's video graph [11], [12]. In this paper we used the random prefix sampling method from [11]. Random prefix sampling over YouTube video ID space is done through utilizing a unique property of the YouTube's API which allows searching videos using a randomly generated prefix of a possible YouTube video ID. YouTube video ID consists of 11 characters drawn from a set $S = \{0-9, A-Z, a-z, , -\}$. Providing a search string of "v=abc...c" where "ab...c" is a randomly generated string of length less than or equal to eleven returns a list of video IDs which start with this string. This fact enables one to avoid the impossible task of uniform random sampling over the extremely large id space of the order 64^{11}. Sampling YouTube's video graph through random prefix sampling rather than breadth first search or other graph based sampling techniques produces an unbiased sample. In our experiment we used a randomly generated 4 character prefix to sample the YouTube's video graph. Experiments show that a prefix larger than 4 often returns an empty list indicating the fact that YouTube's video ID space is randomly generated.

In order to collect our online video characteristics dataset we implemented a small wrapper tool in Java over other well known open source tools, ffprobe [13] and youtube-dl [14]. Our tool first generates a four character random prefix from YouTube's video ID space and searches YouTube through its API. The API will return a list of valid video IDs starting with the random prefix. A video ID is then selected randomly from the returned list. Then, a set of direct links for all stored file formats of the video with that ID is fetched by the open source tool, youtube-dl. Each of these video links are then probed by ffprobe, a video analysis tool that can be used to collect video characteristics such as bitrate, framerate, resolution, container format, codec, duration and frame types.

The dataset contains 10 columns of fundamental video characteristics; Duration, video codec, framrate, estimated framerate, total bitrate, video bitrate, resolution ,category, direct video link and video ID. This dataset can be used to gain insight in characteristics of videos on UGC. Analysis of the dataset will be presented in section IV-A.

B. Transcoding Time Prediction Model

Prediction of video transcoding time is important for several reasons. For example, resource management algorithms deployed on a large scale transcoding service such as [15] can utilize this prediction to increase system utilization through proper load balancing and provisioning. Previous work [10] has proposed to model the prediction problem by characterizing a video using its size alone. However that model did not account for variability of video coding algorithms. In this work we will use more video characteristics features for better accuracy and generality over a range of coding algorithms.

An initial stage in building any prediction model is to understand the process itself, in our case video transcoding. The basic idea of video transcoding is to convert unsupported video formats in to supported ones. Unsupported videos include videos that are not playable by a given device due to lack of format support or those that require relatively higher system resources than the device can offer. The main types of video transcoding include, resolution transcoding, bitrate transcoding, temporal transcoding, codec transcoding , error reliance transcoding and any combination of these (see Figure 1).

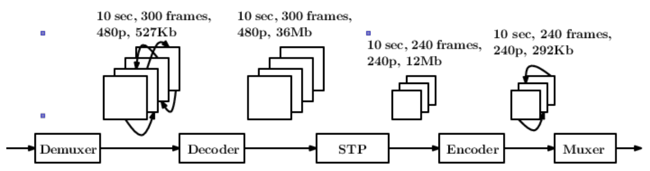

10 sec, 300 frames, 480p, 527Kb
10 sec, 300 frames, 480p, 36Mb
10 sec, 240 frames, 240p, 12Mb
10 sec, 240 frames, 240p, 292Kb
Demuxer
Decoder
STP
Encoder
Muxer

Fig. 1. Basic description of video transcoding. Demuxer and Muxer are I/O components used to read/write compressed video and the rest are video processing components which encode, decode and scale video

A general formulation of the transcoding time prediction problem is to construct a function that takes as input easily extractable characteristics of a video together with transcoding parameters specifying the characteristics of the output video and generate an estimated transcoding time of the video on a given platform. Given a set of t observations with n features (i.e. easily extractable input video characteristics such as bitrate, framerate, codec and transcoding parameters) each and a target variable (i.e. transcoding time) y as $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_t, y_t)\}$ where $\mathbf{x} \in \Re^n$, $y \in \Re$ we used supervised machine learning algorithm NN and SVR to find a model $f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b$ with $\omega \in \Re^n, b \in \Re$ with the best fit.

C. Input Video Characterization

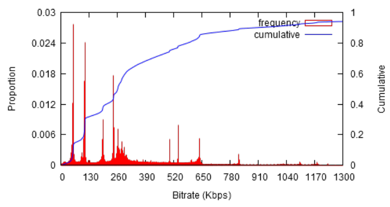
To characterize an input video we collected features which include, *bitrate, framerate, resolution, codec, number of i frames, number of p frames, number of b frames, size of i frames, size of p frames and size of b frames*. This list of video characteristic features is selected through expert analysis of the problem and the insight gained from the dataset presented in section IV-A. These features along with the transcoding parameters listed in table I constitute the full list of our features used in our transcoding time prediction model.

IV. Experiments

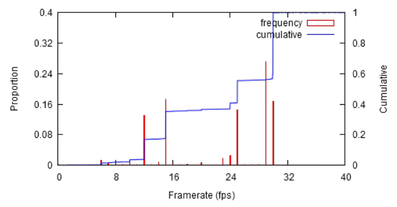
A. Analysis of Video Characteristics

A video has several fundamental characteristics that define its behaviour in-terms of quality, storage, bandwidth requirement and processing time. In this section we will present a statistical analysis of some of these characteristics obtained from our dataset.

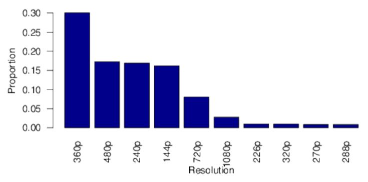
Bitrate Figure 2a shows the bitrate distribution of videos from our dataset. Most of the videos have a bitrate of around 56kbps or 109kbps. Other popular bitrates lay around 260kbps, 520kbps, 600kbps, 800kbps and 1150kbps. In addition we can notice that about 80% of the videos have a bitrate of less than 650kbps, 40% of the videos have a bitrate between 260kbps and 650kbps and another 10% of the videos have a bitrate between 109kbps and 260kbps. There are about 15% videos that have bitrate between 109kbps and 56kbps. This statistics indicates that YouTube uses a moderate bitrate for most of its



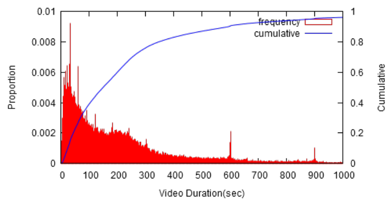
(a)



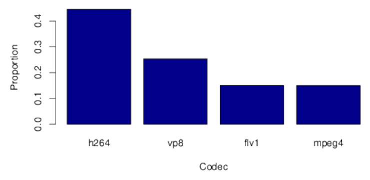
(b)



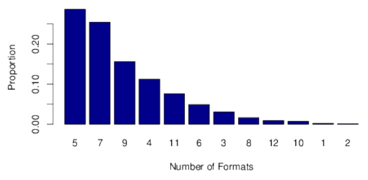
(c)



(d)



(e)



(f)

Fig. 2. Youtube videos characteristics

videos and a low or very high bitrate for only small portion of its videos.

Framerate Figure 2b shows the frame rate distribution of videos from our dataset in which about 15% of videos have a framerate of 12. Experiments show that these videos are older videos with mpeg4 codec. The figure also depicts that 40% of the videos have 30 fps and around 20% of the videos have 25 fps.

Resolution Figure 2c shows the resolution distribution of videos in pixels from the crawled data. Most popular video height measures include 240, 144, 360, 480, 720 and 1080 pixels. Similarly Figure 2c shows the width distribution of videos. This indicates that about 90% of videos in our dataset are of (standard) SD quality and the remaining 10% are of (high definition) HD videos. However the trend of storing videos in HD quality is increasing as the resolution of displays of users is increasing.

Video Duration Figure 2d shows the distribution of duration of YouTube videos. It confirms that indeed YouTube videos are short with the exception of some. The spikes at 600 sec and 900 second show the previous and the current limits that are imposed by YouTube as a maximum video length. However YouTube has currently allowed one to increase the video length limit through additional registration, a fact that can be noticed from the same figure.

Codec Figure 2e shows the distribution of the codecs used by YouTube for their compression. The figure shows most of the videos are encoded in h.264 format followed by mpeg4, vp8 and flv (h.263).

Frame Types Depending on the type of codec used and its parameters, encoded videos contain different types of frames such as I (intra), P (predicted), and B (bi-directional predicted). The proportion and size of each frame types in a video are important characteristics as they correlates well with video size and processing time. For our dataset the frame types are dominated by P frames followed by I frames.

Formats YouTube stores its videos in several formats. These formats enable YouTube to provide its services to wide range of devices having different resolution, supported codec and connected through different types of networks. Figure 2f shows the proportion for all number of formats YouTube uses to store its videos. As can be noted from the figure, most of the videos are stored mostly in 5 formats. However sometimes up to 12 formats are used and rarely more than 20 formats are used.

Detailed report of our analysis on the dataset can be found in [16], which is an extended technical report associated with this paper.

B. Building and Using a Prediction Model

In this section we present the experimental set-up, including the machine configurations and tools used in our modelling experiment. The training data used for modelling transcoding time contains 19 columns which include input video characteristics, transcoding parameters and the measured transcoding time as a label. This training data was collected on an Intel i7-3720QM CPU through randomly picking a video from our YouTube dataset (see section IV-A) and transcoding parameter space from table I.

Collecting Transcoding Time Table I gives the list of the transcoding parameters that are used to create our transcoding parameter space. When considering all the possible combination of all transcoding options, the transcoding parameter space can lead to a very large set of possibilities. We reduced the parameter space size using expert pruning based on the insight gained from the online video characterization results presented in section IV-A, but as seen in table I we still have 840 combinations to consider. We collected the transcoding time obtained for each transcoding combination on a set of 80 randomly selected 20 second YouTube videos. The resulting training data contains 67200 transcoding measurements. All possible combinations of video transcoding parameters from table I has been applied to each video when collecting the training data. The characterization of each online video along

Parameter	Value
Codec	H264, Mpeg4, Vp8, H263
Resolution	144p, 240p, 360p, 480p, 720p, 1080p
Bitrate	56k,109k, 242k, 539k, 820k, 3000k, 5000k
Framerate	12, 15, 24, 25, 29.97

TABLE I
TRANSCODING PARAMETER SPACE

with a transcoding parameter and the corresponding measured transcoding time makes up an instance of our training data. We used Ffmpeg [?] and Ffprobe to collect video transcoding time and video characteristics of each video in our training set.

Constructing the Model Once the training data is constructed, it can be fed to a learning algorithm that will automatically learn a prediction model. We used support vector regression (SVR), linear regression (LR) and multilayer perceptron (MLP) to construct our prediction models. In our experiments we used LibSVM package from RapidMiner [17] with radial basis function (RBF) kernel, γ set to 0.125 and error penalty term C set to 1024. We found these parameter settings using grid search method [18]. In this work we used feed-forward neural network trained by a back propagation algorithm (multilayer perceptron) from RapidMiner [17] to model transcoding time of online videos. The training cycle, the learning rate and the momentum are set to 500, 0.3 and 0.2 respectively.

Finally we used a linear regression model mostly to demonstrates the relationship between dependent and independent variables. Even though our problem has non linear components, we used this model to observe the dependent variables according to the change of given independent variables.

Model on Unseen Video We train and validate our models using 2/3 of the training data we have collected and the remaining 1/3 portion is left for testing and evaluation of the models.

C. Prediction Accuracy and Overhead

We used our training dataset to build and test a transcoding time prediction model based on Neural Network, SVR and LR. Table II shows squared correlation (predicted vs measured) of 0.958 for NN, 0.942 for SVM and 0.411 for LR. This result shows that NN and SVR perform very good in terms of prediction accuracy. We report in table II the training and

Algorithm	R^2	Absolute error	Training time	Testing time
NN	0.958	1.757 ± 2.834	7 ± 2 min	5 ± 2sec
SVR	0.942	1.484 ± 3.594	40 ± 3 min	13 ± 6sec
LR	0.411	7.233 ± 9.997	15 ± 2 sec	2 ± 1sec

TABLE II
COMPARISON OF PREDICTION ALGORITHMS

testing time for the 3 ML algorithms we have used. The three algorithms show different training and testing times over the whole of the training and test set instances. The NN model outperforms the others when we consider accuracy and training overhead.

V. CONCLUSIONS

We have presented a dataset that contains a set of important video characteristics and a summary of basic statistics of these characteristics. The knowledge of these characteristics will enable a proper design and test of large-scale video processing services such as video transcoding.

The dataset was obtained through random sampling of the most popular video UGC cite, YouTube. Such random sampling enabled us to construct an unbiased dataset. The data collection was done using a simple java based tool built on top of other open source tools, the Ffprobe and youtube-dl.

Our benchmark results show how the datasets can be used in constructing models to predict transcoding time with a very good accuracy. We finally encourage other researchers to use our dataset in discovering more interesting information about video characteristics on the web and test their own algorithms to build similar predictive models. For further details on this work readers are encouraged to take a look at [16]

REFERENCES

[1] A. Vetro, C. Christopoulos, and H. Sun, "Video transcoding architectures and techniques: An overview," in *Signal Processing Magazine, IEEE*, 2003.

[2] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video transcoding," *Proceedings of the IEEE*, 2005.

[3] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," *SIGOPS Oper. Syst. Rev.*, 2006.

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," ser. IMC, 2007.

[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," ser. IMC, 2007.

[6] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch global, cache local: Youtube network traffic at a campus network - measurements and implications," Tech. Rep., 2008.

[7] M. J. Halvey and M. T. Keane, "Exploring social dynamics in online media sharing," in *Proceedings of the 16th international conference on World Wide Web*, 2007.

[8] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," ser. IMC, 2007.

[9] M. Roitzsch and M. Pohlack, "Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video," in *RTSS*, 2006.

[10] J. Guo and L. N. Bhuyan, "Load balancing in a cluster-based web server for multimedia applications," *IEEE Transactions on Parallel and Distributed Systems*.

[11] J. Zhou, Y. Li, V. K. Adhikari, and Z.-L. Zhang, "Counting youtube videos via random prefix sampling," ser. IMC, 2011.

[12] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *in Proc. of IEEE IWQoS*, 2008.

[13] S. Sabatini, "Ffprobe," August 2013. [Online]. Available: http://ffmpeg.org/ffprobe.html

[14] R. G. Gonzalez, "youtube-dl," September 2013. [Online]. Available: http://rg3.github.io/youtube-dl/

[15] "Amazon elastic transcoder," August 2013. [Online]. Available: http://aws.amazon.com/elastictranscoder/

[16] T. Deneke, S. Lafond, and J. Lilius, "Analysis and transcoding time prediction of online videos," Tech. Rep. 1145, 2015.

[17] "RapidMiner," Online, Apr. 2012. [Online]. Available: http://rapid-i.com/content/view/181/190/

[18] C. wei Hsu, C. chung Chang, and C. jen Lin, "A practical guide to support vector classification," 2010.

Paper II

Video Transcoding Time Prediction for Proactive Load Balancing

Tewodors Deneke, Habtegebreil Haile, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the International Conference on Multimedia and Expo , 2014 IEEE International Conference*, pages 1–6. Chengdu,China.

VIDEO TRANSCODING TIME PREDICTION FOR PROACTIVE LOAD BALANCING

Tewodors Deneke[1,3], *Habtegebreil Haile*[1], *Sébastien Lafond*[1], *Johan Lilius*[1]

[1]Åbo Akademi University, Finland
[3]Turku Centre for Computer Science, Finland

ABSTRACT

In this paper, we present a method for predicting the transcoding time of videos given an input video stream and its transcoding parameters. Video transcoding time is treated as a random variable and is statistically predicted from past observations. Our proposed method predicts the transcoding time as a function of several parameters of the input and output video streams, and does not require any detailed information about the codec used. We show the effectiveness of our method via comparing the resulting predictions with the actual transcoding times on unseen video streams. Simulation results show that our prediction method enables a significantly better load balancing of transcoding jobs than classical load balancing methods.

Index Terms— Transcoding, Prediction, Machine Learning, Load Balancing

1. INTRODUCTION

Video content is being produced, transported and consumed in more ways and devices than ever. Meanwhile a seamless interaction is required between video content producing, transporting and consuming devices. The difference in device resources, network bandwidth and video representation types results in the necessary requirements for a mechanism for video content adoption. One such mechanism is called video transcoding. Video transcoding is a process of converting one compressed video representation to another. Currently transcoding is being utilized for such purposes as: bit-rate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for insuring high quality of service (QoS) [1, 2].

Transcoding is a computationally heavy process and several methods has been proposed in order to increase its efficiency [3, 4]. Among them many attempts have been made to decrease its computational complexity through reusing information like DCT coefficients and the motion vectors extracted from the original coded data instead of fully re-encoding the video content. On the other hand to realize multiple transcoding and speed up, studies has been done to integrate multiple processors to fully decode and re-encode incoming video [5]. And more recently, new large-scale cloud based elastic transcoding architectures and services are emerging [6, 7, 8].

Runtime scheduling of transcoding jobs in multicore and cloud environments is hard as their resource requirements may not be known before hand. Currently for video transcoding jobs one has to rely on worst-case values which lead to an over provisioning of resources to maintain satisfactory QoS. This is due to the fact that the resource requirement of a transcoding job is highly dependent on the video data to be converted and its conversion parameters. In order to allow such distributed and multicore systems overcome the problem of over provisioning a method for predicting the resource requirement of each job is required.

Today, computing systems vary significantly from one another and range from very small (e.g. cellphones, tablets, notebooks) to very large (servers, data centres, cloud). However, at the heart of each of these systems there are resource management components that decide how to schedule the execution of different tasks over time (i.e., ensuring high system utilization or efficient energy use [9, 10]) or allocate program resources such as memory, storage and networking (i.e., ensuring a long battery life or fair resource allocation). These management components typically must be able to predict how a given task will perform depending on its size and its other characteristics, so as to decide how best to plan for the future. For example, considering a simple scenario in a cloud transcoding service with a set of two types of transcoding requests, fast transcoding jobs in set A and slow transcoding jobs in set B, A scheduler is often faced with the decision of whether to run each set on different CPU resources, potentially taking longer to execute; or to interleave between the two sets and distribute the job fairly, potentially executing the tasks much faster. If the scheduler can predict accurately how long each job would take to execute on a given platform, it can make an optimal decision, returning results faster, possibly minimizing energy, waiting time and maximizing throughput. Figure 1 shows an example distribution of video transcoding times on a set of randomly selected YouTube videos with randomly selected but valid transcoding parameters. Notice the heavy-tailed distribution in transcoding time values which will have a large impact on the performance of the system if scheduled improperly [11].

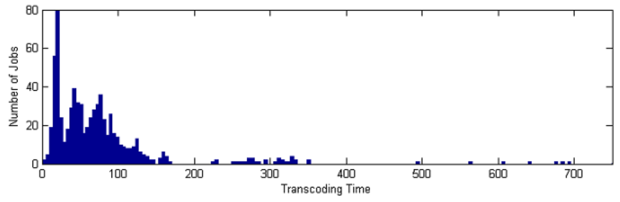


Fig. 1. Transcoding Time Distribution Over Randomly Selected YouTube Video Dataset

In order to leverage such opportunities, use of low overhead and accurate prediction mechanism is required. In this paper we present such prediction models trained based on an expert selected and easily obtainable video meta-data and transcoding (conversion) parameters. By basing scheduling decisions on such ahead of time knowledge, better resource utilization can be achieved.

2. RELATED WORK

Among others our work relates to video transcoding and prediction. In this section, we briefly summarize the related work on each topic.

2.1. Video Transcoding and Scheduling

Zhenhua Li et al. [12] implemented a cloud transcoder which requires a user to provide a video link and other transcoding parameters such as format, resolution, etc. Once a user provides the video link and other parameters, the required video is downloaded from the Internet and transcoded in a Cloud. After transcoding, the video is sent to the user. A copy of video is stored in a cloud cache to avoid repeated transcoding operations. The paper mainly focus on providing video transcoding service, and it does not talk about how such transcoding jobs are scheduled.

In [13] a distributed video transcoding was implemented with Message Passing Interface programming model. The video was segmented statically at Group of Pictures (GOP) level. The main focus of the paper was on parallelization and data distribution among computing units for bit rate reduction video transcoding. Although the paper provided a distributed video transcoding, however job scheduling was not the main topic.

In [8] prediction-based dynamic resource allocation algorithm to scale video transcoding service on a given Infrastructure as a Service cloud were discussed. The proposed algorithm provides mechanisms for allocation and deallocation of virtual machines based on a regression model that tracks and predicts the aggregate target transcoding rate required by the service. This work only uses queue length when load balancing transcoding jobs probably because tracking transcoding progress of individual streams is very expensive.

2.2. Machine Learning and Prediction

Roitzsch et al. [14] have presented per-frame decoding time prediction for modern video decoding algorithms. In this work they used expert selected metric to train and predict decoding time of videos encoded in MPEG. This work is specific to the MPEG family of codec and can not be applied directly to our problem as transcoding application should support a set of codecs from different codec families.

2.3. video characterization

There has been significant research on understanding the workloads of new generation video servers. These researches especially focus on the social aspect of videos and traffic characterization such as popularity, active life span, user access pattern, growth pattern, request patterns, etc.

Yu et al. [15] study user behaviour, content access pattern and their implications on the design of large-scale video-on-demand systems. Possible improvements on UGC design were proposed by Cha et al. [16] after studying YouTube and Daum, a popular UGC in Korea. After tracking YouTube transactions from a network edge, Gill et al. [17] have tried to understand video access characteristics and discuss the implications of their observation on key concepts such as caching. The caching problem in YouTube has been further studied by Zink et al. [18]. The social networking among videos was studied in the works of Halvey et al. [19] and Mislove et al. [20].

In this work we will reuse the traffic model from [15] to drive our experiments but further focus on collecting the missing statistics on video characteristics such as video length, size, bitrate, frame rate, codec type, resolution and etc that will be useful in our experiments.

3. SYSTEM OVERVIEW

Our work provides an approach for transcoding time prediction and shows its application in load balancing transcoding requests of a transcoding service. The main contribution of our work is to design an automated system that predicts the transcoding time of videos given an input video and a transcoding parameter set. These predictions can then be used for load balancing and QoS predictions by the service provider. Our system provides the opportunity for transcoding service providers to estimate the transcoding time of requests, and more intelligently manage their transcoding servers through proactive load balancing. We employ the idea of machine learning to design a framework that learns to predict transcoding time of videos. The key component of our work is to select fundamental video features that enable building precise transcoding time prediction model, and then use the resulting model on unseen videos to strategically load balance transcoding requests across multiple nodes. Figure 2 presents the overview of our framework. Typically transcoding service providers possess a log about transcoding requests (i.e. both transcoding parameter sets and the original video). Based on these traces, we can build a training dataset listing samples containing transcoding parameter set, the original video (or its fundamental characteristics such as resolution and bitrate) and measured transcoding time. Using such a dataset a prediction model can be trained via machine learning algorithms such as neural network and support vector machines (SVM). The model can then be used to predict and properly distribute load across transcoding nodes. The same prediction model can further be used to estimate the cost of transcoding a video and the QoS to be expected by the user.

4. TRANSCODING TIME PREDICTION

Machine learning techniques are often used as decision making mechanisms for a variety of systems. Basically, machine learning allows computers to evolve behaviours based on empirical data, in our case, this is a collection of samples with important video characteristics, transcoding parameter sets and measured

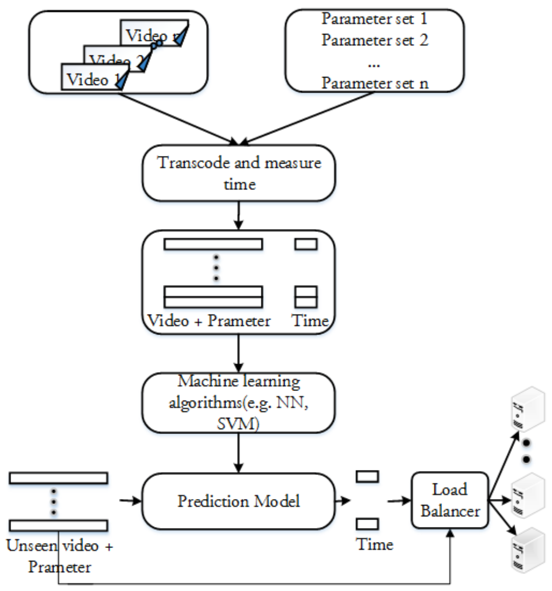


Fig. 2. Architecture of the prediction system

transcoding times. In this section, we present the detailed design of the proposed transcoding time prediction method based on the machine learning approaches.

4.1. Video Transcoding

An initial stage in building any prediction model is to understand the process itself, in our case video transcoding. The basic idea of video transcoding is to convert unsupported video formats in to supported ones. Unsupported videos include videos that are not playable by a given device due to luck of format support or those that require relatively higher system resources than the device can offer. The main types of video transcoding include, resolution transcoding, bitrate transcoding, temporal transcoding, container transcoding, codec transcoding , error reliance transcoding and any combination of these.

Resolution transcoding enables change of resolution of a given video allowing devices with lower or higher resolution to get served with the most appropriate resolution depending on the size of their display. Resolution of a video is defined by the number of pixels in its two dimensions. Transcoding rate and resolution have a clear correlation, the higher the resolution difference between the input and output video the more processing is needed and thus the lower the transcoding rate becomes.

Bitrate transcoding enables change of bitrate of the video allowing a given video to be served with the appropriate bitrate depending on the bandwidth capacity of the network the consumer device is connected to or storage media it has. Bitrate is one of the most important characteristics of a video stream. It indicates the number of bits in a video per unit time. Video transcoding rate correlates with the bitrate of the input and output video as it is directly proportional to the amount of bit that need to be processed. Larger bitrates enable higher quality but will require larger bandwidth and more processing power.

Temporal transcoding enables change of frame rate of a given video. The human visual system can perceives a sequence of more than 25 pictures per second as a video. This type of transcoding is sometimes used by video on demand providers to provide service with a lower frame rate (quality) in case of live events where encoding resources can become scares. Framerate indicates the number of frames (pictures) in a given video per unit time. The framerate of a video has a correlation with the transcoding rate of the video, the higher the framerate of the input or output video the lower the transcoding rate becomes.

Container transcoding also known as format transcoding is used to change the container (header) of a compressed video. Several container types such as flv and mp4 has been developed over the years to package video, audio and subtitle streams into one file. Each of these containers have some useful features that serve different purposes or are associated with a specific codec.

Codec transcoding Over the years different codecs (compression algorithms) have been developed. Usually the newer the codec the more advanced the algorithm it uses, allowing an ever increasing compression efficiency and quality. Example codecs that are being used today include h264, h263, vp8, and mpeg4. Backward (forward) compatibility with devices and infrastructures based on older (newer) codecs is maintained through codec transcoding. A video codec is a hardware or software implementation of video compression and decompression algorithms.

Video transcoding services provide parameters that control each of the transcoding types listed before. In addition to those parameters such applications (services) take the original video as an input which among others have fundamental characteristics such as bitrate, framerate, resolution, video duration, codec, frame types and count.

4.2. Dataset preparation and understanding

In order to build our model we need a training data. As we have discussed in the previous subsection transcoding time of a video is mainly dependent on a set of input and output video features. Based on expert knowledge we have partly presented in the previous subsection we have picked a set of features (metrics) that can be used in building our prediction model. This features include, *bitrate, framerate, resolution, codec, number of i frames, number of p frames, number of b frames, size of i frames, size of p frames and size of b frames* of the input video and the desired *bitrate, framerate, resolution and codec* of the output video which are given as a parameter to a transcoding service.

4.3. Modelling

To predict the transcoding time of a video, we use two of the most widely used supervised machine learning algorithms, the support vector regression (SVR) and Neural net. The fundamental idea behind any regression problem in machine learning algorithms such as SVR and the Neural Net can be summarized as: given a set of t observations with n features (in our case the input and output bitrate, framerate, codec, etc) each and a target variable (transcoding time) y as $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_t, y_t)\}$ where $\mathbf{x} \in \Re^n$, $y \in \Re$ the objective is to find a function (model)

$$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b \text{ with } \omega \in \Re^n, b \in \Re \qquad (1)$$

with the best fit as in equation 1.

Neural nets The idea of neural networks was first inspired by nervous system of human beings which consists of a number of simple processing units called neuron. Each neuron receives some input signals from outside or from other neurons and processes them with an activation function to produce its output and sends it to other neurons. These neurons can be understood as a mathematical function that take n element input vector and scale each data element x_i, by a weight w_j. The scaled data is offset by some bias b and put through a differentiable activation function such as equation 2. The output of a neuron can be analytically viewed as equation 1. The impact of Each input is weighted differently from other inputs thus a neuron is able to interpret the data differently depending on the weight and bias. Consequently the more is the weight the stronger the connection would be allowing that data point to influence the output more. The activation function f can be linear or non-linear. Non-linear activation functions are useful in mapping non-linear relationships. One such function is called sigmoid which is represented as

$$\frac{1}{1 + exp(-f)} \qquad (2)$$

A network of these neurons forms a feed forward multilayer neural networks as in 3. These networks are made of layers of neurons. The first layer is the layer connected to the input data. After that there could be one or more middle layers called hidden layers. The last layer is the output layer which shows the results. One of the learning methods in multilayer perception

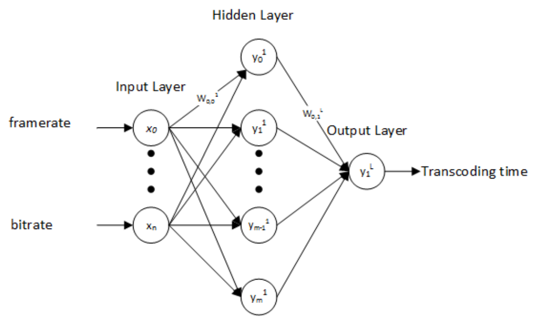


Fig. 3. Mluti-layer neural network used in our prediction

Neural Networks is the error back propagation in which the network learns the pattern in data set and justifies the weight of the connections in the reverse direction with respect to the gradient vector of Error function which is usually regularized sum of squared error.

Support vector machines treats the regression problem as a convex optimization problem:

$$\text{minimize } \frac{1}{2}\|\omega\|^2 \qquad (3)$$

$$\text{subject to } = \begin{cases} y_i - \|w, x_i\| - b \leq \epsilon \\ \|w, x_i\| + b - y_i \leq \epsilon \end{cases} \qquad (4)$$

Similar to the neural net the SVR allows for non-linear solution through the use of radial basis function (RBF) kernels which are represented as

$$exp(-\frac{1}{2\sigma^2}\|f\|^2) \qquad (5)$$

5. EVALUATION

5.1. Experimental Setup

To evaluate our proactive load balancing scheme, we simulate the throughput and job waiting time performance in CloudSim [21]. Every transcoding service provider has traces from its own log at its disposal. Although we do not possess such data, there are a number of sources (such as [15]) that provide information about web traffic in large-scale video-on-demand systems. In order to draw a realistic scenario, we imitate the video transcoding service request patterns with a shifted Poisson distribution as in [15]. We attempt to evaluate our proactive load balancing algorithm in a cloud environment. The largest provider of cloud infrastructure services in the world, Amazon, has published its virtual machine types, CPU types and the costs associated with them [22]. Under our CloudSim simulation environment, we deploy nodes, assign link capacities, and specify CPU and virtual machine characteristics according to data published by Amazon. The instance we used to base the characteristics of our CloudSim nodes is the c1.xlarge which is a 64-bit Intel Xeon E5-2680 machine with 8 virtual cores [22]. For generating the transcoding parameters and input source video associated with each request we used video characteristics data collected from YouTube.

5.1.1. YouTube Video Data Collection

For obtaining a realistic online video content statistics we use YouTube, the largest video sharing portal in the world. It is one of the most well known and widely used UGC (user generated content) website allowing users to upload, tag and share videos effortlessly. Users can also view, rate and comment on videos which brings a powerful social aspect to the site and in turn to its success. YouTube provides some statistics for its videos. Such information as view-count, number of likes/dislikes, duration and comments are public. However, more detailed statistics about fundamental video characteristics such as video bitrate, framerate, resolution and codec, which are essential to our experiment for generating realistic transcoding request parameters are not made publicly available by YouTube or any other similar service. Therefore we have implemented a crawler that uses the random prefix sampling method presented in [23] to sample over a million YouTube videos. For each video we sampled, we have probed, analysed and collected its fundamental characteristics using Ffprobe [24]. These collected video characteristics are then used to generate valid transcoding parameters associated with our requests. For each transcoding request we simulate we obtained its actual transcoding time through measurement on a cloud instance (c1.xlarge) provided by Amazon EC2. Note that the characteristics of c1.xlarge is used to instantiate the nodes in our simulation environment.

5.2. Evaluating Prediction Accuracy

We evaluate the performance of our prediction accuracy by comparing it with measured transcoding time. First we have built a dataset containing input video, video transcoding parameters, and a measured transcoding time while transcoding the input video with the parameters on an Amazon EC2 c1.xlarge instance. The input video and the transcoding parameters are obtained through randomly picking videos from our crawled data from YouTube. Our dataset contains 2733 instances. We have divided this set into 2/3 training plus validation set and 1/3 test set. This means we have 820 instances of unseen test set. Finally we trained two prediction models based on neural network and SVR using the training and the validation set. In our Neural net model we have used 12 nodes in the hidden layer and the input layer takes 19 input attributes, 1 id and 1 label. The number of iteration for the back-propagation is set to 500. For our SVM model we used the grid search method to search for hyper-parameters using training and validation set. The number of support vectors used is 1795.

Figure 4 shows the prediction accuracy as compared to the actual transcoding time on the unseen test set. The result shows a mean absolute percentage error of 8.78% for our neural network model and a mean absolute percentage error mean absolute error of 18.64% for our SVR model.

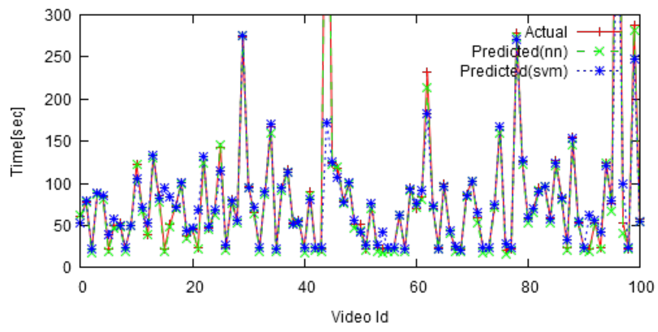


Fig. 4. Prediction Performance. Only 100 out of 820 results for figure readability

5.3. Evaluating Proactive Loadbalancing

We evaluate our proactive load balancing algorithms (see algorithm 1)in comparison with two widely used algorithms in video service systems. 1) queue length based load balancing. Load is being balanced based on queue length associated with each server. 2) Round robin approach, where transcoding requests are routed to transcoding servers in a round robin fashion.

Algorithm 1 Proactive load balancing algorithm

```
1: $server \leftarrow servers(0)$
2: for all $req$ in $requests$ do
3:     for all $s$ in $servers$ do
4:         if $predLoad(s) < predLoad(server)$ then
5:             $server \leftarrow s$
6:         end if
7:     end for
8:     $send(req, server)$
9:     $load(req) \leftarrow predictTranscodingTime(req, algo)$
10:    $predLoad(server) \leftarrow predLoad(server) + load(req)$
11: end for
```

Transcoding rate which measured in frames per second (fps) is defined as the number of frames transcoded per unit time (sec). Higher transcoding rate indicates better throughput, reduced delay and total cost of the system. Figure 5 illustrates the total transcoding rate of the system for a three hour load modelled from a realistic scenario described so far.

We observe that our proactive load balancing algorithms based on predicted load with our Neural Net and SVR models can achieve up to 15% improvement in terms of system throughput over the round robin and The queue length approaches before over provisioning occurs (i.e. 300 servers). One can also note that use of queue length or round robin produces the same result as the curves for these approaches overlap all the way.

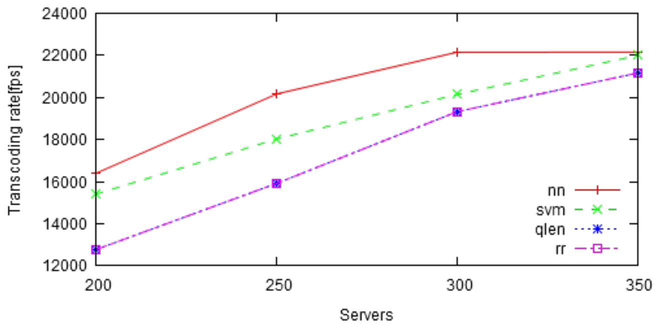


Fig. 5. Throughput

Average waiting time reflects the average waiting time of a transcoding request before it is given a resource and start being processed. Figure 6 shows the results from the simulation on the average waiting time of a job. Our proactive load balancing approaches enables reduction of average waiting time of a job by up to 18% enabling a better QoS.

6. CONCLUSION

This paper addresses the challenges exposed by the modern online video transcoding services. We design a proactive load balancing scheme to aid throughput and quality of service for video transcoding services. Our novel method explores the opportunities provided by trends from transcoding load of requests. Due

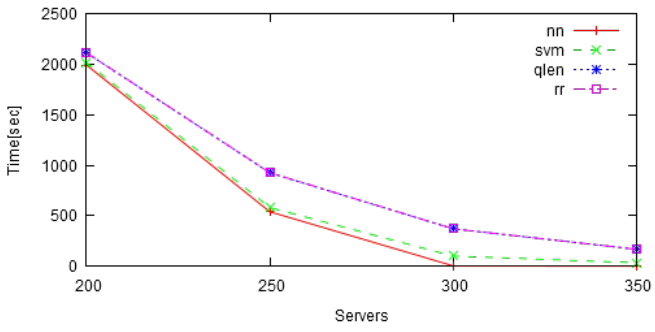


Fig. 6. Average waiting time for different load balancing approaches

to the correlation between transcoding time and transcoding parameters, we can predict transcoding time of a video given the fundamental characteristics of the input video and the transcoding parameters which in turn is used to properly load balance video transcoding requests across servers. In our experiment, we have used real-world data and designed the simulation scenario imitating real world Internet transactions. Our proactive load balancing algorithm shows effectiveness and significant improvement over the traditional methods. For the future work, we would like to further explore the effect of our load balancing on dynamic resource provisioning.

Acknowledgment

This work was supported by the Cloud Software Finland research project and by an Amazon Web Services research grant.

7. REFERENCES

[1] S. F. Chang and A. Vetro, “Video adaptation: Concepts, technologies, and open issues,” *Proceedings of IEEE*, 2005.

[2] J. Xin, C.-W. Lin, and M.-T. Sun, “Digital video transcoding,” *Proceedings of the IEEE*, 2005.

[3] G. Morrison, “Video transcoders with low delay,” 1997.

[4] Jeongnam Y., Ming-Ting S., and Chia-Wen L., “Motion vector refinement for high-performance transcoding,” *IEEE Transactions on Multimedia*, 1999.

[5] Y. Sambe, S. Watanabe, Dong Yu, Taichi N., and Naoki W., “High-speed distributed video transcoding for multiple rates and formats,” *IEICE - Trans. Inf. Syst.*, 2005.

[6] Amazon Inc., “Amazon elastic transcoder,” August 2013.

[7] Zencoder Inc., “Zencoder cloud transcoder,” August 2013.

[8] F. Jokhio, A. Ashraf, S. Lafond, I. Porres, and J. Lilius, “Prediction-based dynamic resource allocation for video transcoding in cloud computing,” in *PDP*, 2013.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, aug 2003.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, “Quincy: fair scheduling for distributed computing clusters,” SOSP 2009.

[11] Mor Harchol-balter, “The effect of heavy-tailed job size distributions on computer system design,” in *In Proc. of ASA-IMS Conf. on Applications of Heavy Tailed Distributions in Economics*, 1999.

[12] Z. Li, Y. Huang, G. Liu, F. Wang, Z. Zhang, and Y. Dai, “Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices,” in *22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2012.

[13] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, “Bit rate reduction video transcoding with distributed computing,” in *PDP*, 2012.

[14] M. Roitzsch and M. Pohlack, “Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video,” in *RTSS*, 2006.

[15] H. Yu, D. Zheng, B. Zhao, and W. Zheng, “Understanding user behavior in large-scale video-on-demand systems,” *SIGOPS Oper. Syst. Rev.*, 2006.

[16] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, “I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system,” IMC 2007.

[17] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, “Youtube traffic characterization: a view from the edge,” IMC 2007.

[18] M. Zink, K. Suh, Y. Gu, and J. Kurose, “Watch global, cache local: Youtube network traffic at a campus network - measurements and implications,” Tech. Rep., 2008.

[19] M. Halvey and M. Keane, “Exploring social dynamics in online media sharing,” WWW 2007.

[20] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” IMC 2007.

[21] R. Calheiros, R. Ranjan, A. Beloglazov, A. De Rose, and R. Buyya, “Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Softw. Pract. Exper.*, 2011.

[22] Amazon Inc., “Amazon ec2 instance types,” August 2013.

[23] J. Zhou, Y. Li, V. Adhikari, and Z. Zhang, “Counting youtube videos via random prefix sampling,” IMC 2011, ACM.

[24] Stefano Sabatini, “Ffprobe,” August 2013.

Paper III

Integration of Dataflow Components Within a Legacy Video Transcoding Framework

Tewodros Deneke, Lionel Morel, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the International Workshop on Signal Processing Systems, 2015 IEEE International Conference*, pages 1–6. Hangzhou,China.

Integration of Dataflow Components Within a Legacy Video Transcoding Framework

Tewodros Deneke[1, 2], Lionel Morel[3], Sébastien Lafond[2] and Johan Lilius[2]
[1]TUCS - Turku Centre for Computer Science, Finland
[2]Åbo Akademi University, Finland
[3]Universite de Lyon, Inria INSA-Lyon, CITI, F-69621, France

***Abstract*—Recently the RVC-CAL dataflow language has enabled video codecs to be specified in a more natural way than imperative languages by allowing implicit expression of parallelism and side effect freeness. The tools developed for RVC-CAL have also enabled the automatic generation of parallel C code, among others, from dataflow specifications.**

This paper introduces a new approach allowing the integration of dataflow components within legacy code. The approach makes use of a generic interface definition that allows seamless interaction between I/O components, which are mostly state operations and are best implemented in imperative languages with data processing components which are mostly stateless dataflow operations and are best implemented in dataflow languages. The advantage of the approach is the ease of development by allowing each language to be used on those parts of the application that it is most appropriate for.

The functionality of the approach is demonstrated by using the generic interface to add a new dataflow based MPEG and HEVC decoder into the legacy video transcoding library FFmpeg.

I. Introduction

Video is becoming an important medium of communication and accounts for a significant portion of the available bandwidth in the world. Today video is being consumed through various devices and networks. These devices and networks have varying capabilities in terms of screen resolution, storage space, processor speed and bandwidth. Video transcoding is needed to enable seamless exchange of videos among various devices on heterogeneous networks like the internet. Video transcoding is the process of transforming one video format with certain characteristics such as bitrate, framerate and resolution to another video with different characteristics that will better fit the target device. One typical scenario of transcoding would be the delivery of video content by video on demand (VOD) systems, like YouTube, to customers connected trough wireless or wireline using their mobile phone, tablet or IPTV. In order to reach such a wide range of consumer segments, VODs transcode and store videos in several formats that are tailored to each segment.

As presented in figure 1, the components of a video transcoder can be roughly categorized in to two: I/O and video processing components. The I/O components are responsible for reading/writing video/audio in different formats to/from a disk or network (e.g. Muxers and Demuxers). The video processing components are mainly responsible for manipulating the video/audio content (e.g, encoder and decoder). While the I/O components are inherently serial and implemented as state operations, the video processing components are parallelizable and compute intensive and are more naturally expressed with a dataflow programming model. For example a video codec, which is one of the video processing components, is a software that enables compression and decompression of digital multimedia content. The main use of a codec is to compress video so that it uses less resources during transmission and storage and later to decompress it for viewing.

Over the years video codecs have become more efficient in terms of their compression ability while becoming more compute intensive. Until recently the increase in computational complexity of codecs has not been a real concern because computer speed has been doubling every other year following Moore's law. However due to physics constraints like power dissipation and transistor scaling, multi-core architectures have become the solution to allow performance to keep increasing. Programming codecs for multi-cores is currently done using imperative languages such as C/C++ using threads to explicitly express parallelism. This is obviously a challenging, time consuming and error prone process as multi-threading is essentially a way to only prune non-determinism after its introduction [1]. Research in dataflow programming shows the advantages of using dataflow languages such as RVC-CAL for the purpose of video codec specification and implementation. Among others, the main advantages of using dataflow as the main programming method include ease of use, flexibility, automatic analyzability, automatic parallelizability, visual presentability and above all side effect freeness [2], [3], [4].

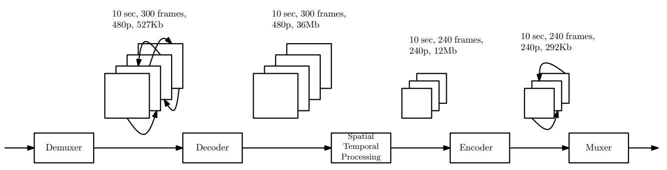


Fig. 1. Basic description of video transcoding. Demuxer and Muxer are I/O components used to read/write compressed video and the rest are video processing components which encode, decode and scale video.

While programming video processing components of a video transcoder (e.g. codecs) entirely in dataflow is an attractive idea, programming the I/O components in dataflow is unnatural on Von-Neumann architecture. I/O operations have been introduced as state operations in the Von-Neumann architecture as a convenient and clean way of reading in data into programs. On the other hand the absence of side effects which is fundamental to dataflow programs does not allow state operations and makes dataflow implementation of I/O operations difficult.

In this paper we propose an approach based on the definition of a generic interface that enables video/audio processing system designers to write dataflow components in a dataflow language and interface them with existing legacy video/audio applications such as a transcoder. The proposed approach allows the splitting of the development of video processing applications in two parts: one part which uses dataflow language and implements parallel sections and another, which uses imperative languages and implements the I/O sections. It combines the advantages of using dataflow languages with the features of legacy imperative code and helps quicker adoption of dataflow languages.

The rest of the paper is organised as follows. Section II introduces the reader with necessary background knowledge. Section III reports on related works. Section IV actually describes the body of work of our proposition. Section V presents experimental results evaluating the benefits in terms of performance and the seemingless integration of a substantial dataflow application into the FFmpeg transcoding platform. Finally, section VI concludes and gives few perspectives to this work.

II. Background

A. Video Transcoding and FFmpeg

The difference in device resources, network bandwidth and video representation types results in the need for a mechanism enabling video content adoption. This mechanism, called transcoding is currently being used for such purposes as: bitrate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for insuring high quality of service (QoS) [5], [6].

As can be noted from figure 2 a generic video transcoder contains five main parts, each having a set of components depicted as blocks. The five main parts include a demuxer, a decoder, spatial and temporal processors, an encoder and a muxer. The demuxer is used to read interleaved streams (e.g. one audio, one video and a subtitle stream) from a network or a file. Usually a set of streams are encapsulated in a container format such as MP4. Packets which are read by the demuxer and contain compressed audio/video frames are then passed to the appropriate decoder to be decompressed. The decompressed audio/video frames are then spatially or temporally processed to adapt the video/audio to a particular framerate and/or resolution. Spatially and temporally processed uncompressed frames are then passed on to the encoder to be compressed via removing temporal, spatial and statistical redundancy that exists inside and among uncompressed frames.

Legacy transcoders such as FFmpeg constitute libraries corresponding to each block. For example FFmpeg contains multiple codec implementations including H264, HVEC, MPEG4. Similarly it contains multiple implementations of muxers and demuxers that can be used to read and write several video and audio file formats such as MP4, MKV, FLV, MP3, WAV, etc.

FFmpeg is an open-source, complete, cross-platform solution to record, convert and stream audio and video[7]. Among others it includes libraries such as libavcodec, libavformat, libavfilter, libavswscale which contain implementations of several video and audio codecs, container formats, filters and spacial processing functions. It is a large project with over 570k lines of code.

The FFmpeg transcoding framework implemented based on FFmpeg libraries provides coarse-grained task (pipeline) and data parallelism. The different components such as the decoder and the encoder can run in a task-based parallel manner. The data parallelism in FFmpeg is currently based on slice-level and/or frame-level threading. In case of slice-level parallelism multiple threads can decode slices (independent parts of a frame) in parallel. In the case of frame-level parallelism multiple threads processes multiple frames. However frames have inter-dependencies and the number of slices in a frame is usually limited to one. Therefore the parallelism implemented in FFmpeg is obviously limited.

B. Dataflow Programming and RVC-CAL

A dataflow program is defined as a directed graph whose vertices are actors (the basic computational units) and edges are unidirectional FIFO channels with unbounded capacity. A stream of data tokens, is processed by actors and passed on to others actors via FIFOs. The advantage of such programming model is its ability to implicitly express concurrency and to enable analyzability. Dataflow graphs we consider here respect the semantics of Dataflow Process Networks (DPNs) [8], which are related to Kahn Process Networks (KPNs) [9]. The main difference between DPN and KPN is that DPN allows actors to check the availability of tokens in the FIFOs. Additionally to the KPN model, DPN introduces the notion of firing. An actor firing, or action, is an indivisible quantum of computation which corresponds to a mapping function of input tokens to output tokens applied repeatedly and sequentially on one or more data streams. This mapping is composed of three steps: input data reading, then computation, and finally output data writing. These functions are guarded by a set of firing rules which specifies when an actor can be fired, i.e. if the number and the values of tokens that are available on the input ports are sufficient.

Dataflow has been used for naturally expressing digital signal processing applications for decades. Currently it has gained particular attention in expressing video processing applications. RVC-CAL is one domain specific dataflow language for video coding based on the DPN model of computation [2], [3], [4], [10]. It was originally developed for specifying video coding standards in the most natural manner. This is due to the fact that video coding is a data-oriented application and can easily be visualized and specified as a dataflow graph where actors correspond to functional units such as discrete cosine transform (DCT) and motion compensation (MC), and video bit stream flows among them via FIFOs. In addition to providing a natural language for specifying video codecs RVC-CAL provides a compilation framework called ORCC[1]. ORCC is implemented as an eclipse plug-in and has backends for C, VHDL and others and comes with various example applications including MPEG4 and HEVC decoders.

An RVC-CAL dataflow program is described as a set of interconnected actors via unidirectional FIFOs. Actors are composed of a set of actions, I/O ports (FIFOs) and internal state variables. Actors perform computation by firing actions depending on the state of their I/O ports and their internal state variables.

A multicore platform ideally runs an RVC-CAL program

[1]see http://orcc.sourceforge.net/

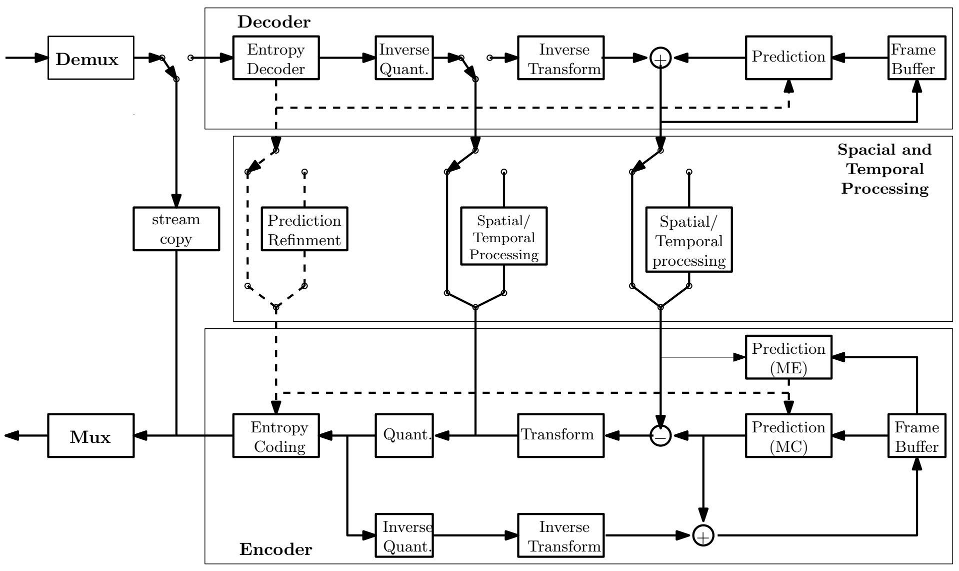


Fig. 2. Complete picture of video transcoding showing the main blocks it constitutes and the different options of doing transcoding depending on the required type of conversion operation, acceptable quality and complexity

by executing each actor in parallel as long as they are fireable. However because the number of actors in a typical dataflow application is usually much greater than the number of processors, several actors are usually executed on the same processor. The RVC-CAL runtime therefore maps and schedules actors using different approaches. Among others mapping is done in a round-robin or in a weighted load balancing shame, and scheduling is usually implemented using a round-robin or a data driven algorithm[11], [12].

III. Related Work

RVC-CAL [3], [4] uses CAL procedures to interface with legacy code and more specifically to call the I/O imperative functions such as file readers and display functions. CAL procedures are included in the RVC-CAL language for this particular purpose and can modify state variables and parameters passed to them by reference. Even though CAL procedures defeat some purposes of the dataflow programming approach such as side effect freeness they are, however practical considering the Von-Neumann architecture. In our work we propose an interface which can be used to call (use) dataflow components from the legacy code. This is important for legacy applications to tap in to dataflow components when appropriate.

In [13], Mark Green et al. have shown two ways of interfacing haskell which is a functional programming language and java an imperative language. Their motivation for their work was to provide each language an access to certain missing language features from the other. As a use case they showed how Java's I/O and UI related libraries which require imperative features that violate the referential transparency of functional programs can be used in haskell. On the other side they have also shown how the haskell features such as lazy evaluation and higher-order functions can be used in Java programs. In our work we explore a similar but more generic interfacing approach on a dataflow language (RVC-CAL) and the C imperative language using a more realistic application.

In [14], Chatterjee et al. presented the HCMPI programming model and runtime for programming distributed systems, which unifies asynchronous task parallelism at intra-node level with MPIs message passing model at the inter-node level. With HCMPI's task parallel model, users can benefit from MPI integration with structured task parallelism and dataflow programming. Similarly, in [15] and other research works around hybrid programming approaches, combining programming models is proven to be useful in dealing with hierarchical and various hardware designs.

In this paper, we propose the use of dataflow programming to express the available concurrency in video transcoding applications in a more natural way. At the same time, we allow to keep the effective legacy components of current transcoding applications. We enable this through the use of a generic interface defined between components implemented using different programming paradigms.

IV. Interfacing Dataflow with Legacy Code

Ideally we would need to find the most transparent way to accomplish the interfacing. It should be done in such a way that enables code in both languages to remain natural. There are two possible approaches to interface imperative code with dataflow code. One possible approach, adopted by current RVC-CAL application developments, is based on the interface driven by the dataflow code. The I/O imperative functions are called from the dataflow program using CAL procedures which are ad-hoc mechanisms put in place to accomodate legacy code in to dataflow components.

The approach proposed in this paper is also based on the use of imperative code for I/O. However in this approach it is the imperative code that calls the dataflow code with an input data to be processed. The main advantage of this approach is the ease of development. Each language is used to implement those parts of code for which the language is most appropriate for, without the need to accommodate the languages to each other (as in RVC-CAL procedures). This enables independent prototyping and reuse of code already written. It also permits legacy libraries to access dataflow components and helps adoption of dataflow programing.

A. Interface Definition

The interface is designed to be generic enough such that any new dataflow component, like a decoder, encoder or filter, can be added to any legacy video processing library. The proposed interface consists of three functions and a data structure which are explained and implemented as follows.

1) init_component: This interfacing function is used to launch the dataflow component. Launching a dataflow component can be conceived as starting a conveyor belt system in a factory. Once a conveyor belt along with the processing units connected to it are started, a factory is ready to receive a stream of items to be processed. In our case, the initialization function starts the dataflow runtime system. The RVC-CAL runtime mainly consists of mapping, scheduling and other utility routines. The runtime takes in a set of actors, their network and user-supplied parameters such as an input data and maps, schedules and executes actors on a number of processing units. Mapping is either done by simply assigning actors to available processing units in a round-robin manner or via more complex post-profiling weight-based methods [11]. Once actors are mapped to a processing unit they are scheduled using round-robin or more advanced data-driven methods.[12].

```
Data: context
Result: success
1 set_component_context($context$);
2 success =
  thread_create($launcher, launch, context, tid$);
```
Algorithm 1: Dataflow component initialization

The pseudo-code in Algorithm 1 shows the implementation of the *init_component* function. $context$ is any information that is needed to start the component. It contains the runtime options of the dataflow component such as mapping policy, scheduling policy, number of cores to be used and the dataflow network itself. Once the desired $context$ of the component is set, the dataflow component is launched with a new thread which ensures the *init_component* function returns control to the legacy code immediately. This allows the caller to continue its execution by sending row data and receiving processed data from the dataflow component.

2) process: This function is responsible for feeding the already initialized dataflow component with input data and grubbing the output data if available. Every call to this function from the legacy transcoder might fill the input FIFO of the dataflow component or get tokens from the output FIFO of the component, or both.

As shown in Algorithm 2, the *process* function takes in the *context* of the dataflow component which contains the network information and the data to be processed in *ipkt*. It then returns any result that might be available from the dataflow component in *opkt*. Note that this function also returns the size of the data consumed by the component through the variable *sent*. Any unconsumed data on the input FIFO of the dataflow component should be re-supplied to this function. *got_result* is used to tell a calling function if the dataflow component resulted in a valid output via *opkt*.

```
   Data: context, ipkt
   Result: sent, opkt, got_result
1  tosend=ipkt.size;
2  sent = 0;
   /* send input to dataflow                 */
3  sent += send(context,ipkt, tosend);
4  last_processed=processed;
   /* recive processed data                   */
5  processed += recv(context,opkt);
6  if $last\_processed < processed$ then
       /* we have got data                    */
7      got_result = 1;
8  else
9      got_result = 0;
10 end
11 ipkt.size -= sent;
12 ipkt.data += sent;
```
Algorithm 2: Dataflow processing

3) close_component: This interface function is used to end the already running dataflow component. More specifically it ends the runtime of the component by joining all created threads that were responsible for executing the component's actors.

```
Data: context
Result: success
1 thread_join(launcher);
2 success = free_component_context(context);
```
Algorithm 3: Dataflow component termination

Algorithm 3 shows the *close_component* function.

4) Component Structure: This structure definition enables the use of multiple dataflow components and ensures the generic nature of the interface.

```
typedef struct component {
    const char name;
    enum type component_type;
    enum id component_id;
    struct component *next;
    int (*init_component)(context *);
    int (*process)(context *, opkt*, ipkt*, *got_result);
    int (*close_component)(context *);
}
```
Algorithm 4: Component Structure

As can be noted from Algorithm 4, the component definition allows multiple dataflow components to be identified by a name or id. It also contains pointer to the three functions that are used to initiate, use and close a given dataflow component.

B. Generating Interface and Dataflow Component

In order to have an automated workflow for integrating dataflow components into a transcoding framework, we propose generating the interface automatically from the ORCC backend.

As shown in figure 3 we have modified the ORCC C backend to generate the dataflow components as a library along with a header file instead of stand alone executable. In addition

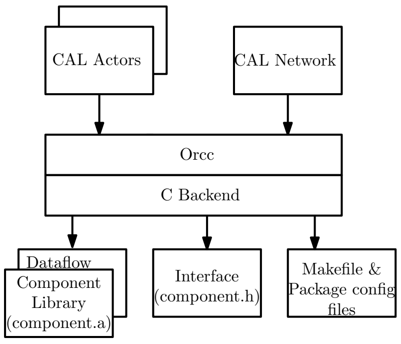

CAL Actors
CAL Network
Orcc
C Backend
Dataflow Component Library (component.a)
Interface (component.h)
Makefile & Package config files

Fig. 3. Generation of the dataflow component library and its interface

we have added generation of package configuration files so that the library can be installed and be used easily.

C. *Using the Interface*

In order to demonstrate the functionality of the approach we have generated MPEG and HEVC video decoders form the corresponding dataflow descriptions, written in RVC-CAL, using our modified C backend. Figure 4 shows the structure of the HEVC dataflow decoder.

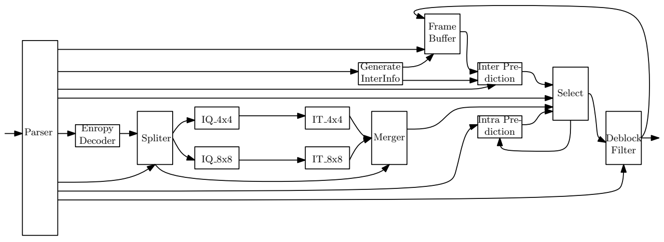


Fig. 4. Dataflow Decoder Implementation from ORCC

The generated dataflow component libraries are then compiled and installed in our system. Following that we have included the header files of the generated dataflow components in our legacy video transcoding framework and linked to the installed component libraries during its compilation.

Algorithm 5 shows the use of a dataflow decoder component by the legacy video transcoder FFmpeg. Three main points can be noted from the pseudo-code. The first is the `register_all` function that is used to register the available formats, codec and filters in a given system. This function is from FFmpeg libraries and we have also used it to register our new dataflow decoder component. The second point to note is the use of our interface which constitutes the three functions, *init_component, process and close_component*. This interface can be used to abstract the various types of dataflow components that can be implemented and integrated to FFmpeg or any other legacy transcoder. Finally one can note that the FFmpeg libraries supply the I/O (read/write) functions which are capable of parsing almost any known video container format efficiently. In addition to the I/O functions, data processing functionalities such as video scalers and encoders that are yet to be implemented by dataflow approach can also be used.

Using our interface we were therefore able to provide FFmpeg, a legacy video transcoding framework with dataflow components that implement fine-grained parallelism. Note from figure 4 that the dataflow decoder component provides a fine grained parallelism by implementing functional blocks as separate actors.

```
Data: $v_s$ $(source\ video), context$
Result: $v_p$ $(processed\ video)$
register_all();
init_component(context) ;            // initialize component e.g. decoder/filter/encoder
while read($context, v_s, ipkt$) do
    process(context, opkt, ipkt, got_result) ;  // e.g. decode/filter/encoder
    if got_result then
        rescale_frame (context, fpkt, opkt, got_result);
    end
    if got_result then
        encode_frame (context, opkt, fpkt, got_result);
    end
    if got_result then
        write(context, opkt);
    end
    ipkt.data += ret;
    ipkt.size -= ret;
end
flush();
close_component(context) ;  // close component e.g. decoder/filter/encoder
```

Algorithm 5: FFmpeg overview

V. EXPERIMENTS

In order to check the proper operation and benefits of our integration approach[2] we have made two evaluations which include quality and scalability measurements. In all of our experiments we used four different input videos from different categories with resolution of 1080p, bitrate of 1200kbps and frame rate of either 24 or 30 or 50. The transcoding operation performed in the experiments are resolution reduction transcoding. See result tables I and II for details on the input video characteristics and the transcoding parameters.

A. *Quality*

First, we made a quality difference measure between video transcoding operations which use the dataflow decoder and the original legacy decoders that comes with the FFmpeg transcoder. The original videos were of 1080p resolution and were transcoded to 240p, 480p or 720p. The quality difference were measured using three metrics, *psnr,ssim, msssim* [16], [17]. The visual similarity matrices *ssim, msssim* are close to 1.00 and the structural similarity metrics *psnr* is *inf* for all tests indicating that the resulting videos are similar. This means our proposed interfacing works properly.

B. *Evaluation of Scalability*

Besides showing the proposed interface works correctly we here present scalability measurements on a 6 core Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz Machine. Table I shows the speed up of 62-80% when using 4 cores for a dataflow component (MPEG4 part 2 simple profile decoder) against a single core. Similarly Table II shows the speed up of 48-65%

[2]https://github.com/tdeneke/ffmpeg-2.5.3, https://github.com/tdeneke/orcc and https://github.com/tdeneke/orc-apps

when using 4 cores for the dataflow component (HEVC simple profile decoder) against using a single core. The remaining 2 cores were assigned to the rest of the transcoding framework components such as the encoder. The speed up is calculated as $\frac{(fps_4 - fps_1)}{fps_1} * 100$ where fps_4 is the transcoding speed in frames per second (fps) when using 4 cores for the dataflow component and fps_1 is the transcoding speed in fps when using 1 core for the dataflow component. Each mesurment is repeated 10 times to calculate the confidence intervals.

TABLE I. EXPERIMENTAL RESULTS SHOWING THE FRAME RATE (FPS) OF THE DIFFERENT VIDEO STREAMS AND THE CONFIDENCE INTERVAL FOR A 97.5% CONFIDENCE LEVEL. THE SPEEDUP FROM RUNNING THE DATAFLOW COMPONENT OF THE TRANSCODER WITH A SINGLE CORE TO RUNNING IT IN FOUR CORES IS GIVEN FOR EACH RESOLUTION AND VIDEO SEQUENCE.

	240p	480p	720p
Cartoon - Elephant Dreams - 24fps			
1	3.456 ± 0.036	3.319 ± 0.027	3.075 ± 0.039
2	4.808 ± 0.011	4.529 ± 0.051	4.092 ± 0.027
4	6.085 ± 0.172	5.422 ± 0.537	4.995 ± 0.074
speedup	76.07%	63.36%	62.44%
Consumer Video - Old Town Cross - 50fps			
1	3.432 ± 0.030	3.302 ± 0.029	3.065 ± 0.036
2	4.805 ± 0.012	4.535 ± 0.017	4.106 ± 0.011
4	6.180 ± 0.066	5.742 ± 0.050	4.996 ± 0.172
speedup	80.07%	73.89%	63.00%
Documentary - Snow mountain - 30 fps			
1	3.487 ± 0.010	3.386 ± 0.040	3.191 ± 0.025
2	4.822 ± 0.028	4.607 ± 0.047	4.223 ± 0.020
G4	6.064 ± 0.380	5.863 ± 0.071	5.171 ± 0.292
speedup	73.90%	73.15%	62.05%
Sport - Touchdown Pass - 30 fps			
1	3.373 ± 0.040	3.241 ± 0.031	2.975 ± 0.033
2	4.528 ± 0.382	4.383 ± 0.073	3.917 ± 0.022
4	5.508 ± 0.764	5.478 ± 0.250	4.519 ± 0.583
speedup	63.3%	69.02%	51.9%

TABLE II. EXPERIMENTAL RESULTS SHOWING THE FRAME RATE (FPS) OF THE DIFFERENT VIDEO STREAMS AND THE CONFIDENCE INTERVAL FOR A 97.5% CONFIDENCE LEVEL. THE SPEEDUP FROM RUNNING THE DATAFLOW COMPONENT OF THE TRANSCODER WITH A SINGLE CORE TO RUNNING IT IN FOUR CORES IS GIVEN FOR EACH RESOLUTION AND VIDEO SEQUENCE.

	240p	480p	720p
Cartoon - Elephant Dreams - 24fps			
1	1.474 ± 0.001	1.473 ± 0.001	1.467 ± 0.002
2	1.800 ± 0.001	1.797 ± 0.003	1.791 ± 0.003
4	2.193 ± 0.089	2.185 ± 0.144	2.233 ± 0.033
speedup	48.78%	48.34%	52.22%
Consumer Video - Old Town Cross - 50fps			
1	1.396 ± 0.001	1.394 ± 0.002	1.391 ± 0.001
2	1.729 ± 0.003	1.729 ± 0.004	1.723 ± 0.003
4	2.176 ± 0.025	2.136 ± 0.119	2.144 ± 0.084
speedup	55.87%	53.23%	54.13%
Documentary - Snow mountain - 30fps			
1	1.493 ± 0.002	1.489 ± 0.003	1.482 ± 0.002
2	1.916 ± 0.011	1.916 ± 0.006	1.905 ± 0.001
4	2.461 ± 0.065	2.460 ± 0.044	2.273 ± 0.164
speedup	64.84%	65.21%	53.37%
Sport - Touchdown Pass - 30fps			
1	1.307 ± 0.001	1.304 ± 0.001	1.299 ± 0.001
2	1.356 ± 0.673	1.595 ± 0.002	1.588 ± 0.003
4	1.993 ± 0.054	2.009 ± 0.023	2.004 ± 0.009
speedup	52.49%	54.06%	54.27%

VI. CONCLUSION

In this paper we have proposed the use of a generic interface for integrating dataflow components such as decoders, encoders and filters with legacy transcoding libraries. The interface enables seamless interaction between dataflow and legacy imperative code allowing each programming approach to implement components for which it is appropriate for.

We have also tested and shown the proper functionality of our approach. Scalability evaluations also show the gain that can be obtained from using dataflow components via the proposed interface.

In the future we would like to further explore the effect of the dataflow component runtime and transcoding framework runtime on each other.

REFERENCES

[1] E. A. Lee, "The problem with threads," *Computer*, pp. 33–42, May 2006.

[2] Orcc, "Open RVC-CAL compiler," 2009. [Online]. Available: http://orcc.sourceforge.net/

[3] M. Wipliez, "Compilation infrastructure for dataow programs," Ph.D. dissertation, INSA Rennes, Sep. 2010.

[4] H. Yviquel, "From dataflow-based video coding tools to dedicated embedded," Ph.D. dissertation, UNIVERSITE DE RENNES 1, Oct. 2013.

[5] S. F. Chang and A. Vetro, "Video adaptation: Concepts, technologies, and open issues," *Proceedings of IEEE*, Jan 2005.

[6] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video transcoding," *Proceedings of the IEEE*, Jan. 2005.

[7] ffmpeg, "ffmpeg," 2000. [Online]. Available: https://www.ffmpeg.org/

[8] E. A. Lee and T. M. Parks, "Readings in hardware/software co-design," Norwell, MA, USA, 2002, pp. 59–85.

[9] G. Kahn, "The semantics of a simple language for parallel programming," in *Information processing*, J. L. Rosenfeld, Ed. Stockholm, Sweden: North Holland, Amsterdam, Aug. 1974, pp. 471–475.

[10] J. Eker and J. W. Janneck, "Cal language report: Specification of the cal actor language," University of California, Berkeley, Berkeley, California, USA, Tech. Rep., 2003.

[11] H. Yviquel, E. Casseau, M. Raulet, P. Jaaskelainen, and J. Takala, "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms," Sept 2013, pp. 732–737.

[12] H. Yviquel, E. Casseau, M. Wipliez, and M. Raulet, "Efficient multicore scheduling of dataflow process networks," in *SiPS*, Oct 2011, pp. 198–203.

[13] M. Green and A. E. Abdallah, "Interfacing java with haskell." in *Scottish Functional Programming Workshop*, ser. Trends in Functional Programming, vol. 1, 1999, pp. 79–88.

[14] S. Chatterjee, S. Tasrlar, Z. Budimlic, V. Cave, M. Chabbi, M. Grossman, V. Sarkar, and Y. Yan, "Integrating asynchronous task parallelism with mpi," in *IPDPS*, May 2013, pp. 712–725.

[15] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," Feb 2009, pp. 427–436.

[16] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, 2004.

[17] vqmt, "vqmt," 2013. [Online]. Available: http://mmspg.epfl.ch/vqmt

Paper IV

Bit Rate Reduction Video Transcoding with Distributed Computing

Fareed Ahmed Jokhio, Tewodros Deneke, Sébastien Lafond, Johan Lilius

Originally published *Proceedings of the 20th International Euromicro Conference on Parallel, Distributed and Network-based Processing, 2012, IEEE International Conference*, pages 206–212. Munich, Germany

Bit Rate Reduction Video Transcoding with Distributed Computing

Fareed Jokhio, Tewodros Deneke, Sébastien Lafond, Johan Lilius
Åbo Akademi University
Department of Information Technologies
Joukahainengatan 3-5, 20520 Turku, Finland
Email: {fjokhio, tdeneke, slafond, jolilius}@abo.fi

***Abstract*—This paper presents an approach to perform bit rate reduction transcoding by video segmentation. The paper shows how a high performance distributed video transcoder can be built using multiple processing units and a Message Passing Interface based parallel programming model. The computation parallelization and data distribution among computing units is discussed. For data distribution coarse grain approach is used in which significant gain in terms of execution speedup is obtained. The segmentation of video stream with (1) equal size having unequal number of intra frames and (2) unequal size having equal number of intra frames is performed to achieve high performance. The results show that the proposed distributed video transcoder provides very short startup times.**

***Keywords*-video Transcoding, Message Passing Interface(MPI), Distributed computing**

I. Introduction

Currently there is a diversity of multimedia applications and there are several video formats available with different characteristics. Users want to play a video in various formats and on different devices. Some users demand for high definition video while others demand for low resolution video. With the passage of time the number of video compression standards is growing therefore it is not practically possible to store a video in all possible formats to fulfill the end user requirements. Also the channels through which a video is distributed to the end user can have different capacities. The compressed video stream needs to be re-encoded to meet the bit rate of the communication channel. Video transcoding is a popular technique to solve these issues [1], [2].

A video transcoder takes a compressed video signal as input and produces another compressed video signal as output. The output video can have different bit rates, different frame rate, different frame size, any combination of these or even an entirely different compression standard.

To get better quality video, the video transcoder should decode the encoded video stream and then re-encode it with new characteristics. Both video decoding and encoding require a number of highly computational tasks hence this seems a time consuming process and waste of computational power. Other smart video transcoding techniques already exist which transcode the video in desired format by just partially decoding and reusing the motion vectors information [3]. Even with existing fast transcoding techniques, the process of transcoding still needs lot of computational power while dealing with high resolution videos such as 4CIF, 16CIF, HD 1080, etc. In order to get better performance, distributed video transcoding can be used to distribute the computational burden among processing units [4].

The main contributions of the paper are:

- the analysis of video segmentation for transcoding in a distributed environment
- the evaluation of the video startup time in such environment

This work also puts special attention towards scalability of the transcoder implementation. This means that we must be able to run the same transcoder in a distributed environment with any number of processing units.

In the next section, we briefly describe the background and related work. Section III gives an overview of the encoding, decoding and introduces the bit rate reduction transcoding. The method for achieving the bit rate reduction and motion vectors refinement is also discussed in this section. In section IV video stream structure is described briefly and then three different ways of video segmentation are discussed. Section V presents the system overview and describes the tasks performed by master and worker machines. In section VI experimental setup is discussed and results are provided in section VII. Finally conclusion and future work is given in section VIII.

II. Background and related work

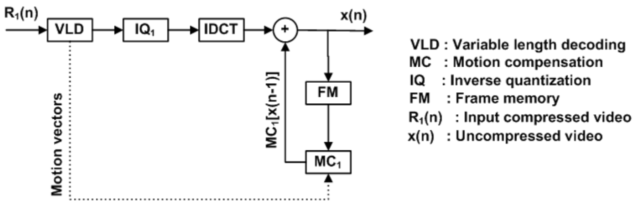
Video transcoding has been studied and improved in the past two decades. The quality of the transcoded video and the speed of the transcoding process are main issues in video transcoding. Distributed computing is a solution to get more speedup in the transcoding process and keep the same quality of video. However, how to optimally handle multiple video streams, their startup times and how to scale the transcoding architecture is still open research problem.

Jiani Guo [5] proposes a cluster based multimedia web server. The work was related to dynamic generation of video according to the requirements of bandwidth and bit rate for many clients. The partitioning of jobs is done by a media server and the computation is done on several nodes. The proposed system was designed for seven nodes. Sambe [4] worked on distributed transcoding of MPEG-2 to produce output video with different bit rates. His work was concerned to produce multiple formats and rates and he integrated multiple processors to fully decode and re-encode incoming video. He paid more attention on segment handling while Jiani Guo paid more attention on load balancing of multiple video streams. Neither author considered video startup time.

Among different methods of distributed computing we have chosen to use MPI (message passing interface) because of its maturity, support, open source nature, scalability and ease of use. MPI is a message passing interface for MIMD (multiple instructions multiple data) distributed memory concurrent computers and workstations [6]. In this programming model a set of tasks that use their own local memory during computation can be performed on the same physical machine and /or across an arbitrary number of machines. Tasks exchange data through communication channels by sending and receiving messages (i.e. message passing.). This means that data transfer usually requires cooperative operations to be performed by each process. That is for example, a send operation must have a matching receive operation. From programming perspective, message passing implementations commonly comprise a library of subroutines that are embedded in source code. The programmer is thus responsible and free to express all parallelism involved [6], [7].

III. Bit rate reduction video transcoding

The main goal of bit rate reduction transcoding is to reduce the bit rate while maintaining low complexity and achieving the best quality possible. The bit rate reduction video transcoding has wide range of applications such as television broadcast and streaming of video over the internet. In bit rate reduction video transcoding the compressed video is decoded and then re-encoded with new bit rates.



$R_1(n)$
VLD
IQ_1
IDCT
+
x(n)
FM
MC_1
$MC_1[x(n-1)]$
Motion vectors
VLD : Variable length decoding
MC : Motion compensation
IQ : Inverse quantization
FM : Frame memory
$R_1(n)$: Input compressed video
x(n) : Uncompressed video

Figure 1. Video decoder

The block diagram of a video decoder is shown in figure Figure 1. Video decoding is a complex operation and it consists of several other operations such as variable length decoding, inverse quantization, inverse discrete cosine transformation, motion compensation. The computation required in decoding operation for low resolution video frames is less as compared with the high resolution video frames. The video decoder has a compressed bit stream as input and produces an uncompressed video as output.

Figure 2 shows the block diagram of a video encoder. As shown in the figure, the video encoding has even more complex operations than video decoding. A video encoder consists of a number of other operations such as discrete cosine transform, quantization, variable length coding, and motion compensation. A video encoder also performs the decoding operation after the quantization operation and then computes the difference between the original video frame and the decoded frame after compression. This difference is termed as a residual frame and is also sent with the compressed bit stream. The information entropy for a residual frame is usually less due to similarities in the nearby video frames, and it requires fewer bits.

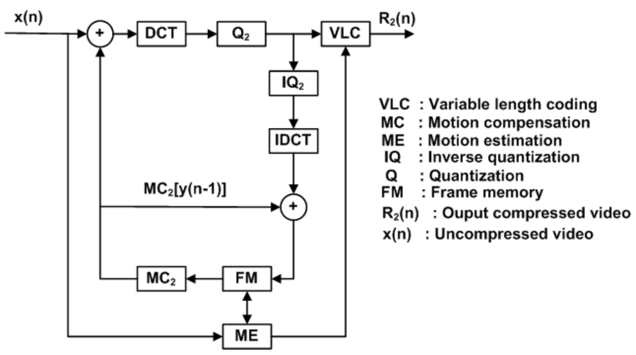


Figure 2. Video encoder

The different operations such as variable length encoding, variable length decoding, quantization, inverse quantization, discrete cosine transform, inverse discrete cosine transform, motion estimation and motion compensation are performed on the block level in a frame. Figure 3 shows the structure of a video frame. The number of blocks in a frame depends on its resolution. If the frame has high resolution, the number of blocks will also be high and more computation will be required during encoding and decoding process.

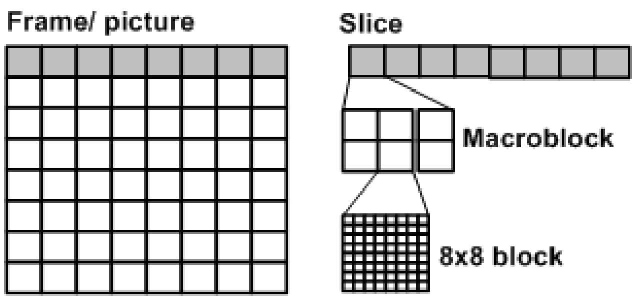


Figure 3. Video frame

In a bit rate reduction transcoder the video resolution and the frames rate are unchanged. The bit rate reduction is possible by compromising on the video quality [8], [9], [10]. It is possible to reduce the bit rate by applying the inverse quantization and then again applying the quantization with increased quantization step at the encoder side in the transcoder [11], [12], [13]. This operation will increase the zero quantized coefficients and hence fewer bits will be required to encode the data. In order to reduce the complexity in bit rate reduction transcoding the motion vectors computed at the original bit rate are reused in the reduced rate bit stream. Using the same motion vectors will lead to degraded video quality due to the mismatch between prediction and residual components [3]. To overcome this loss of quality motion vector refinement is needed. Video frames contain objects and background. Successive video frames may have similar objects and and these objects can be displaced at another location. Motion estimation is used to examine the movement of objects. In block based motion estimation, the similar blocks are searched in the reference frame. The estimated motion of a block is represented by a motion vector. The motion estimation is performed within a fixed search window and it may have size such as [-2, +2], [-16, +16] or any other suitable size. In order to keep the low complexity, motion vector refinement is performed with small search window[1]. The size of the search window is kept very small to reduce the computational load. Increase in the search window size will give slightly better quality but it will have more computational load. The [-2, +2] search window achieves the majority of gain due to the fact that the majority of macro blocks will have a best match within this range.

IV. Video stream structure

A video stream consists of several independent units called as video sequences where every sequence has its own headers. The video sequence consists of several group of pictures (GOP). The group of pictures consists of frames. There are different types of frames; I (intra) frame, P (Predictive) frame and B (bidirectional) frame. The frame is further divided in slices, each slice consists of macro blocks and every macro block consists of blocks. Figure 4 shows the video stream structure down to the frame level.

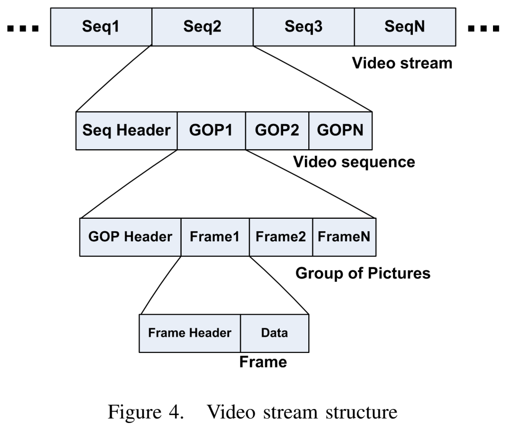


Figure 4. Video stream structure

A. Segmentation of video sequence at Group of Picture

The first issue with distributed transcoding approach is how to perform the segmentation of the source video so that parts of video can be distributed among worker machines to perform the transcoding operations. Compressed video files contain different types of frames (I, P, B) which have different compression rates and inter-dependencies among them. Therefore one cannot split a given video at any particular frame. Among the frame types a frame of type I (intra) is independent and can be decoded without any other reference frame. In a given video sequence a group of frames containing one I frame followed by a number of other B and P frames is called a group of pictures (GOP). Our video sequence partitioning algorithm utilizes this concept to divide a video file in to smaller parts. The master machine divides the incoming video file into parts which contain a number of GOPs and sends these parts to worker machines.

In any video sequence there are two kinds of group of pictures, either the entire video sequence will consist of open-GOP or it will consist of closed-GOP. In the case of closed-GOP it is possible to decode the entire GOP independently. In the case of open-GOP, the last I or P frames of the previous GOP is needed as a reference frame to decode the first B type frame. Segmentation of open-GOP is further discussed in [4]. In the case of open-GOP in every segment there will be one extra I or P frame from the other GOP. This extra frame will be discarded by the master machine while performing the merging. However it will require some extra computation in the transcoding process. In our experiments we used closed-GOPs in the source video.

We segmented the video in three difference ways:

- each segment has equal number of intra frames but the size of segment may be different due to the different sizes of GOPs.
- each segment has equal size but the number of intra frames may be different.
- each segment has unequal size and unequal number of intra frames.

Most video sequences have different number of frames in each GOP. The size of the source video sequence segments for first two cases is shown in table I. The source video used is big buck bunny, further details about the source video are provided in the experimental setup.

The third method of video segmentation is used to get the minimum video startup time. The video startup time is the time at which the end user will be able to view the video.

The MPI based transcoder implementation can handle the transcoding if the number of segments is more than the number of workers.

B. Video segmentation with unequal load

Figure 5 shows the video segmentation of source video with unequal load.

paritions	equal size partitions	Unequal size in Mega Bytes
2	39.95MB each	28, 51
3	26.63MB each	19, 21, 39
4	19.97MB each	14, 15, 19, 33
5	15.98MB each	11, 12, 13, 16, 29
6	13.32MB each	8.6, 9.0, 11, 12, 12, 27
7	11.41MB each	7.5, 7.8, 8.7, 8.8, 8.9, 11, 26

Table I
SIZE OF SEGMENTS FOR TRANSCODING

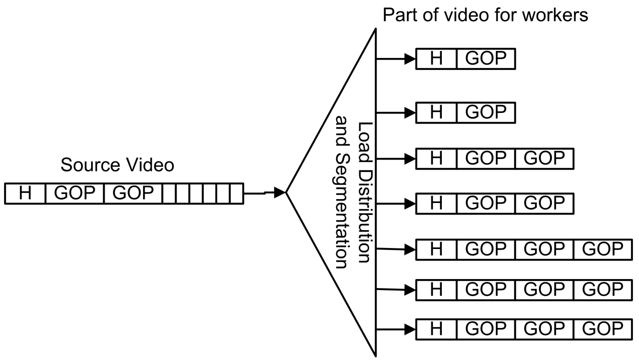


Figure 5. Video segmentation with unequal load

The MPI based transcoder with unequal size partitions is designed in such a way that the master will produce a very small size segment and will send it to the first worker. By keeping the small size of first video segment, it will require very less processing power for transcoding and hence the video start up time will be very less.

The second segment will have slightly more number of I frames and will be bigger in size than the first segment. While producing the second segment the master machine will already have the information about the first segment size and type of frames inside it. The master will make sure that the transcoding time of the second partition is less than the play time of first segment. In the same way it will keep record of the play time of the first n-1 segments while making the nth segment. Since the transcoding operation is performed in parallel the master machine will have a choice to make bigger segments after sending some parts of source video to workers. More care is required when sending segments to the first few workers.

After sending small size segments to few workers, the master will send bigger size segments to other workers. If the segments size will be bigger, there will be better efficiency in overall transcoding and there will be less traffic on the network due to fewer messages between master and workers.

V. SYSTEM OVERVIEW

We selected the ffmpeg open source video transcoder which is designed to work on a single machine as the basis for our experiments. Further details about this transcoder can be found in [14]. We modified this transcoder to execute on multiple processing units in a distributed environment using the MPI. In MPI based implementation we create multiple processes of the transcoder and each process transcode its own part of video stream.

We have two different scenarios for our transcoding system. In the first case each worker will get only one segment to perform the transcoding for one video sequence. Hence we cannot have more partitions than number of available workers. This scenario is used for the first two possible ways of video segmentation i.e. equal size segmentation with unequal number of intra frames and equal number of intra frames with unequal segment size. The one to one mapping of video segments to worker machines is helpful in getting the overall high performance in transcoding.

The second scenario is used to get the shortest possible startup time. In this case the number of video segments is higher than the number of available workers. The video segments are sent to workers in round robin fashion.

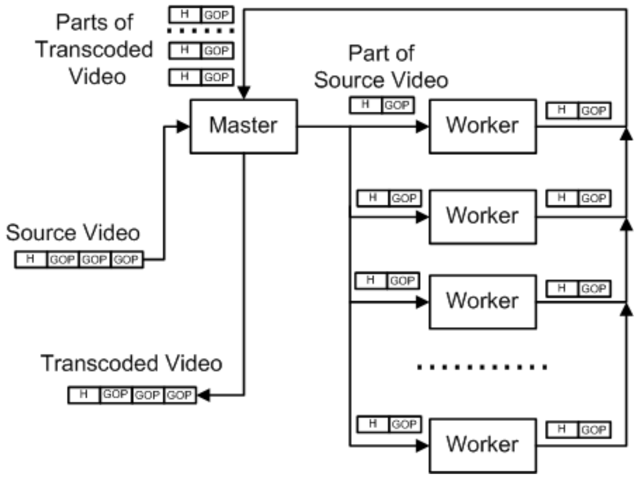


Figure 6. Distributed Video Transcoder with Message Passing Interface

Figure 6 shows the architecture of the distributed video transcoder for a single video sequence. The source video sequence header is attached to every GOP to make it a video sequence so that the transcoder can transcode it according to given parameter values.

In the MPI based implementation the number of total worker machines for transcoding a given video stream is decided by the master; video segmentation and load balancing are performed according to the number of worker machines.

In MPI based systems every machine has its own ID and the work is assigned according to their IDs. The master machine will have ID zero and it will perform the video sequence segmentation task first and then will send the data to worker machines to perform the transcoding operation. The master machine will wait until it gets back the transcoded results from all workers. After receiving the transcoded results it will perform the merging task. It is also possible to make any other machine as a master with ID other than zero. If the number of video streams to be transcoded is high then multiple masters can also be created.

All worker machines perform the same kind of transcoding operation. The instructions for transcoding the video sequence are the same for all worker machines but they all get different parts of the video stream. The worker machines have to receive the part of video stream from the master machine then perform transcoding operation and send back the results to master machine.

VI. Experimental Setup

The experimental system consists of AMD Opteron(tm) Processor workstation and the configuration of the system is shown in table II. Each core of the Dual-Core processor is used as a processing element and takes part in the transcoding operation. The same implementation of the MPI based video transcoder can be mapped on a multi-core system.

model name	Dual-Core AMD Opteron(tm) Processor 2214 HE
cpu MHz	2194.498
cache size	1024 KB
address sizes	40 bits physical, 48 bits virtual

Table II
Configuration of the machines in cluster

The big buck bunny video sequence [15] was used as source video to perform transcoding operations. The source video has H263 4CIF (704x576) format with 24 fps and 1125 kbps. The size of the source video is 79.9 MB and its play time is 09 minutes and 56 seconds. The total number of frames in this video sequence is 14314.

VII. Results

To test our approach we transcode the video down to lower bit rates. The original size of the video sequence was 79.9 Mega Bytes. We started transcoding video sequence at 982kbps and went down to 349kbps. With lower bit rates the quality of video was degraded. Table III shows the file size after transcoding with different bit rates.

Bit rate	File size	Bit rate	File size
982 kbps	70Mb	518 kbps	44Mb
883 kbps	63Mb	428 kbps	38Mb
789 kbps	57Mb	359 kbps	31Mb
699 kbps	50Mb	349 kbps	25Mb
608 kbps	44Mb		

Table III
File size after transcoding for different bit rates

A. Transcoding time

The bit rate reduction transcoding requires the same number of operations for transcoding the video sequence at different bit rates hence the transcoding time is the same for different bit rates. Figure 7 shows the transcoding time for both equal size partitions having unequal number of intra(I) frames and unequal size partitions having equal number of intra (I) frames. The graph shows that there is gain in terms of speed up when using more workers. With equal number of intra (I) frame partitions the overall transcoding time is less and the performance is better for a single video stream.

The quantization process requires more computation in bit rate reduction transcoding; this process is performed only on intra macro blocks. The intra frames have only intra macro blocks and require more computational power as compared with P and B frames having inter macro blocks. The P and B frames may also have intra macroblocks but the number of those intra macro blocks is very less as compared to the macro blocks of intra frames.

The equal size segmentation with unequal number of intra frames partitioning also provides speed up as the number of workers is increased but is less efficient than the equal number of intra frames and unequal size segmentation.

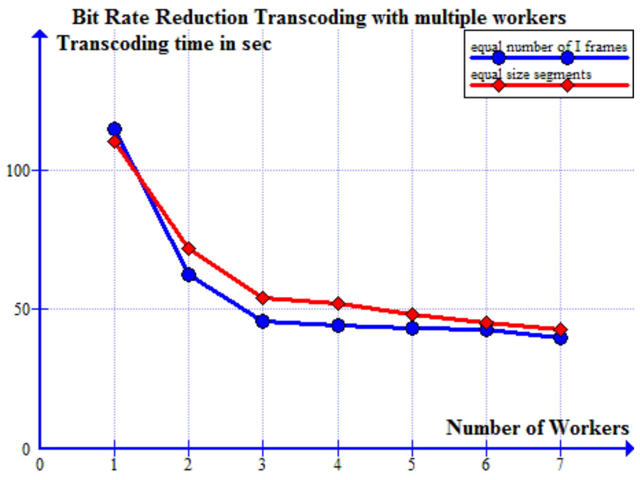


Figure 7. Bit rate reduction Transcoding Time

B. Startup time

The figure 8 shows the start up time for different sizes of video segments. With unequal size segmentation having small number of Intra frames, the startup time can be very less. The results show that for 2 megabytes video segment the transcoding time is less than 2 seconds. The number of frames in 2 mega bytes video is more than 360. With 24 frames per second the play time for this video segment will be 15 seconds. Hence the second worker will be able to transcode a bigger size video segment and it has to send back the transcoded result before the 15 seconds deadline so that the user may able to see uninterrupted video. The third worker will be able to transcode on even bigger size video segment. The video segmentation time is small as compared with transcoding time. It takes less than 1 second to perform the video segmentation operation and send segments to workers for transcoding.

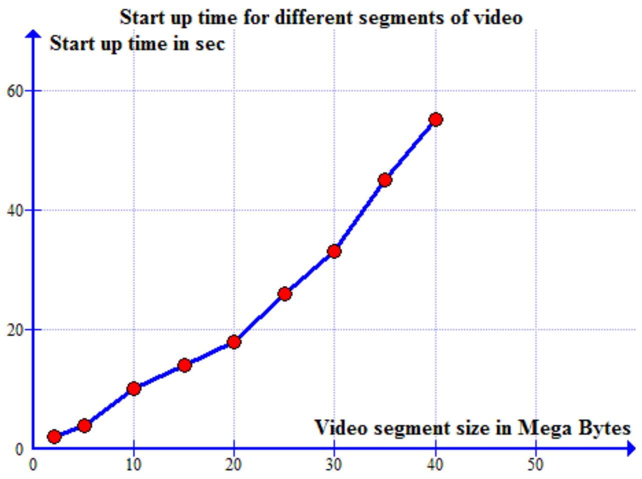


Figure 8. Start up time for different sizes of video segements

C. Analysis

The Peak Signal to Noise Ratio (PSNR) is used to measure the quality of compressed images. The Average Peak Signal to Noise Ratio is used to measure the quality of video. Here PSNR is calculated for all frames of video and then finally Average of PSNR is calculated to get APSNR.

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

The x_{ij} and y_{ij} shows the pixel values of source image and compressed image. The W and H indicate the width and height of the image.

We performed the transcoding operation for different bit rates. The Peak Signal to Noise Ratio at 349kbps is shown in figure 9.

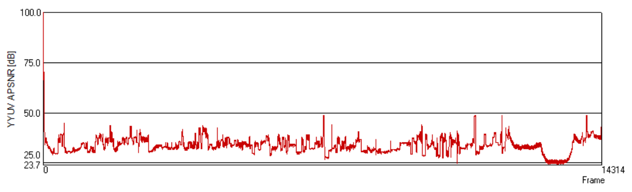


Figure 9. Peak Signal to Noise Ratio at 349kbps

Figure 10 shows the APSNR for different bit rates starting from 349kbps to 982kbps. If the value of APSNR is above 30 it means the video quality is acceptable and if the value of PSNR is higher it means the quality of the compressed image is better. Maximum value of PSNR can be 100 and in that case two images will be exactly identical.

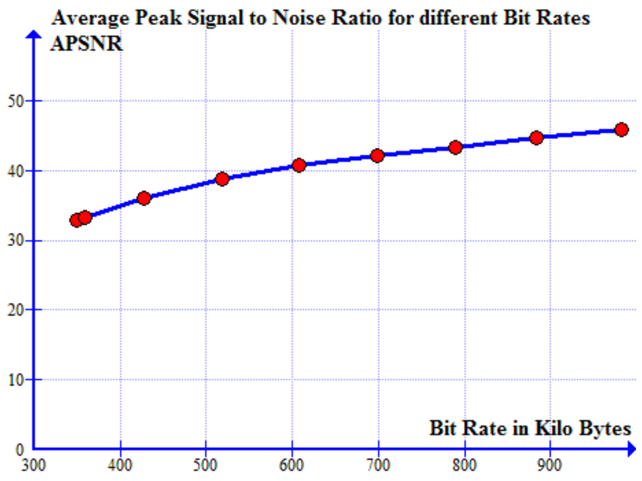


Figure 10. Average PSNR for various bit rates

The transcoding experiment was performed several number of times starting with one master and one worker to one master and seven workers. The output video quality was the same with different number of workers hence the MPI based transcoder did not degraded the video quality.

VIII. Conclusion and Future Work

In this paper, we have proposed a scalable distributed MPI based transcoder implementation. In this implementation a master node (workstation) partitions a given input video file into a number of parts and distributes among worker nodes. The actual transcoding is performed by worker machines in parallel to get more speedup of overall transcoding process. The worker machines send back the transcoded video to master for merging. We were able to see a considerable performance gain with MPI based transcoder as number of worker machines increases. It was observed that the segmentation with equal number of intra frames is more efficient than equal size segmentation. The unequal size segmentation is better for having short startup time. The video start up time can be as low as 2 seconds and then uninterrupted service is possible for the end user. In addition the MPI based transcoder needs no change in its design as the number of processing units grows up. The MPI based transcoder can handle any other type of video format but the headers information needs to be handled while performing segmentation. The MPI based transcoder can also be used for other types of transcoding just like spatial and temporal transcoding and further experiments can be performed on both these types of transcoding.

The MapReduce can also be used to perform video transcoding in a cloud computing environment. In future we intend to perform distributed transcoding using the cloud computing with both MPI and MapReduce and then see which one of them provides better results.

References

[1] A. Vetro, C. Christopoulos, and H. Sun, “Video transcoding architectures and techniques: an overview,” *Signal Processing Magazine, IEEE*, vol. 20, no. 2, pp. 18 – 29, mar 2003.

[2] S. F. Chang and A. Vetro, “Video adaptation: Concepts, technologies, and open issues,” *Proceedings of IEEE*, vol. 93, no. 1, pp. 148–158, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1109/JPROC.2004.839600

[3] N. Bjork and C. Christopoulos, “Transcoder architectures for video coding,” *Consumer Electronics, IEEE Transactions on*, vol. 44, no. 1, pp. 88 –98, feb 1998.

[4] Y. Sambe, S. Watanabe, D. Yu, T. Nakamura, and N. Wakamiya, “High-speed distributed video transcoding for multiple rates and formats,” *IEICE Transactions*, vol. 88-D, no. 8, pp. 1923–1931, 2005. [Online]. Available: http://dx.doi.org/10.1093/ietisy/e88-d.8.1923

[5] J. Guo, F. Chen, L. Bhuyan, and R. Kumar, “A cluster-based active router architecture supporting video/audio stream transcoding service,” in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, april 2003, p. 8 pp.

[6] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard.* Knoxville, TN: University of Tennessee, Jun. 1995.

[7] Gropp, W., Lusk, E., and Skjellum, A., *Using MPI, Portable Parallel Programming with the Message Passing Interface.* MIT Press.

[8] P. Assuncao and M. Ghanbari, “Transcoding of single-layer mpeg video into lower rates,” *Vision, Image and Signal Processing, IEE Proceedings -*, vol. 144, no. 6, pp. 377 – 383, dec 1997.

[9] T. Shanableh and M. Ghanbari, “Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats,” *Multimedia, IEEE Transactions on*, vol. 2, no. 2, pp. 101 –110, jun 2000.

[10] H. Sun, W. Kwok, and J. Zdepski, “Architectures for mpeg compressed bitstream scaling,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 2, pp. 191 – 199, apr 1996.

[11] Y. Nakajima, H. Hori, and T. Kanoh, “Rate conversion of mpeg coded video by re-quantization process,” in *Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3 - Volume 3*, ser. ICIP ’95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 3408–. [Online]. Available: http://portal.acm.org/citation.cfm?id=839284.841401

[12] M.-T. Sun, T.-D. Wu, and J.-N. Hwang, “Dynamic bit allocation in video combining for multipoint conferencing,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 45, no. 5, pp. 644 –648, may 1998.

[13] O. Werner, “Requantization for transcoding of mpeg-2 intraframes,” *Image Processing, IEEE Transactions on*, vol. 8, no. 2, pp. 179 –191, feb 1999.

[14] “Ffmpeg project.” [Online]. Available: http://www.ffmpeg.org/

[15] “Big buck bunny video sequence.” [Online]. Available: http://www.bigbuckbunny.org/index.php/download/

Paper V

Proactive Management of Video Transcoding Services

Tewodros Deneke, Sébastien Lafond, Johan Lilius

Submitted *ACM Trans. Multimedia Comput. Commun. Appl., 2016*, pages 1–19. NY, USA

Proactive Management of Video Transcoding Services

Tewodors Deneke, Department of Information Technologies, Åbo Akademi University
Sébastien Lafond, Department of Information Technologies, Åbo Akademi University
Johan Lilius, Department of Information Technologies, Åbo Akademi University

Following the explosion of digital video consumption over the Internet, video processing applications such as encoding, transmuxing and transcoding are increasingly being deployed on cloud-based environments as a service. However, these services exhibit high variations in their required computational power for two reasons: 1) because transcoding jobs exhibit a heavy-tail or exponential service time distribution 2) because transcoding job arrival rates to these services vary significantly based on time of the day.

In this paper we present a proactive transcoding service management approach based on transcoding task size prediction to optimize the usage of cloud platforms. The proposed approach uses machine learning based task size prediction to enable more efficient resource management in terms of load balancing and auto-scaling algorithms. It also allows for a clear definition of service level agreement (SLA) in terms of average waiting time of transcoding jobs. Simulation results show that our proactive transcoding service management methods based on transcoding task prediction enables a significantly better resource utilization for a given quality of service.

CCS Concepts:•**Computer systems organization** → **Cloud computing;**•**Information systems** → *Multimedia streaming;*

Additional Key Words and Phrases: Video Transcoding, Prediction, Resource Management, Cloud Computing

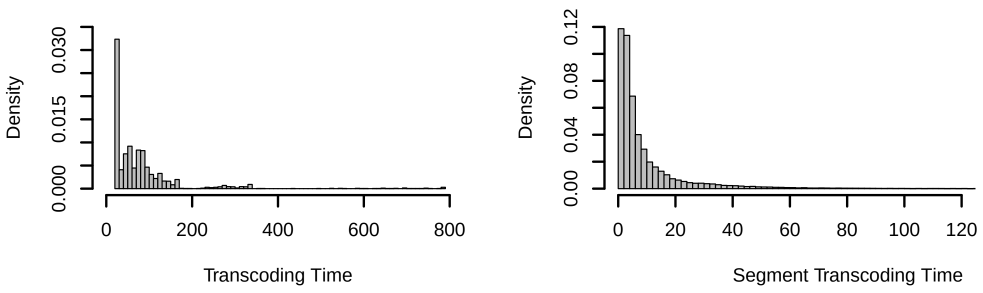
ACM Reference Format:
Tewodros Deneke, Sébastien Lafond, Johan Lilius, 2016. Proactive Management of Video Transcoding Services. *ACM Trans. Multimedia Comput. Commun. Appl.* , , Article (2016), 20 pages.
DOI: 0000001.0000001

1. INTRODUCTION

The consumption of online video by individuals has become more heterogeneous in terms of requested content, network connections, and devices. Video transcoding together with adaptive streaming solutions aim to address this growing heterogeneity by offering users multiple versions of a given video tailored to their various devices and network connections. This means, each version of a given video is encoded at a different bitrate, resolution and with a different codec so that the user is served with the most suitable format for his choice of device and network connection. Currently transcoding is being utilized for such purposes as: bit-rate reduction in order to meet network bandwidth availability, resolution reduction for display size adoption, temporal transcoding for frame rate reduction and error resilience transcoding for insuring high quality of service (QoS) [Chang and Vetro 2005; Xin et al. 2005; Sodagar 2011; Stockhammer 2011; Toni et al. 2014].

Because video transcoding is a computationally intensive process and requires expert knowledge, transcoding operations can usually not be done on the consumer side.

Author's addresses: Tewodros Deneke, Sébastien Lafond and Johan Lilius, Department of Information Technologies, Åbo Akademi University, Åbo, Finland.



(a) Full videos (b) Video segments

Fig. 1. Video transcoding workload distribution of randomly sampled YouTube videos and their 20sec segments while applying valid YouTube transcoding parameters from [Deneke et al. 2015].

Transcoding video content at different resolutions, bitrates and quality levels requires not only time, expertise and equipment but also storage capacity and extensive planning in the design and automation of video transcoding workflows. Therefore content providers, such as YouTube, rely on private and public cloud computing services for large scale video transcoding. As a result, cloud based transcoding services have emerged to automate and virtualize the complex and cumbersome process of video transcoding for content providers and individuals [Inc 2013b; 2013c; Li et al. 2012]. At the heart of each transcoding service are resource management components which decide how resources are allocated, efficiently used and insure service-level agreement (SLA) terms are respected. These management components need to work with either worst-case assumptions on system workload leading to over provisioning of resources and poor management or be proactive and able to predict the workload of incoming transcoding tasks. Such ahead of time predictions can then be used to provide efficient provisioning and load balancing for a given SLA. Proactive management is especially important for two reasons; 1) because service request rates vary over time (as shown in Figure 6) and 2) because transcoding jobs follow a heavy-tailed or exponential distribution depending on whether videos are segmented before processing or not. For example, if we consider the CPU requirement of transcoding jobs required by a video on demand site like YouTube, it turns out that few jobs have relatively very high CPU requirements leading to a heavy-tail or exponential job distribution, as shown on Figure 1. When job sizes are variable it is advantageous to estimate the task size and be able to use dynamic resource management algorithms [Harchol-balter 1999].

In this work we propose a proactive resource management approach for video transcoding services that optimize the usage of cloud platforms while safeguarding their SLAs. Specifically we:

— propose a mechanism to predict the workload of transcoding requests and
— use these predictions to make provisioning and load balancing decisions for a required, agreed upon, quality of service defined as an average waiting time of transcoding job.

2. VIDEO TRANSCODING AND THE CLOUD

To set the scene for this paper, we begin with a brief overview of video transcoding services, the cloud environment and machine learning techniques which are the basis for our proactive resource management approach.

2.1. Video Transcoding

The basic idea of video transcoding is to convert unsupported video formats into supported ones. Unsupported videos include videos that are not playable by a given device due to lack of format support or due to required system resources being limited (e.g. device screen resolution and bandwidth). The main types of video transcoding include, resolution transcoding, bitrate transcoding, temporal transcoding, container transcoding, codec transcoding , error reliance transcoding and any combination of these [Vetro et al. 2003].

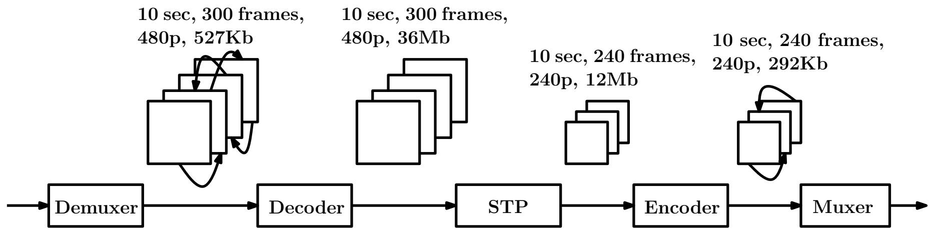

10 sec, 300 frames, 480p, 527Kb
10 sec, 300 frames, 480p, 36Mb
10 sec, 240 frames, 240p, 12Mb
10 sec, 240 frames, 240p, 292Kb
Demuxer
Decoder
STP
Encoder
Muxer

Fig. 2. Basics of Video Transcoding

A typical video transcoding software consists of five main components. These include the Demuxer, Decoder, Temporal Spatial Processor (STP), Encoder and Muxer. The Demuxer is the first part of a transcoder and is responsible for reading the container information of a compressed video content and unpacking the different streams (i.e. audio, video and possible subtitle). The decoder reads one of the compressed streams (e.g. video) and decompress it to remove interdependencies among consecutive frames. The Temporal Spatial Processing component then takes the decompressed stream and applies possible temporal and spatial processing (e.g. adjusts the resolution in pixels and/or the number of frame per second). After the stream has been adjusted both spatially and temporally it will be re-compressed (i.e. unnecessary redundancies that occur across a frame and among frames are removed) by the encoding component. At the end, compressed audio, video and subtitle streams are interleaved and packed together by the Muxer. Depending on the transcoding type a variable number of components will be needed. For example in container transcoding, only the Demuxing and Muxing components are needed while in resolution transcoding all components are required. This leads to a variable transcoding workload pattern as presented in Figure 1.

At this point two challenges can already be pointed out. The first is the complicated and cumbersome nature of the transcoding work flow and the second is the apparent variability of transcoding tasks in terms of required computing resources. This leads to interesting and open research problems related to resource management. The first challenge is currently being addressed by many through moving the transcoding operation into the cloud where it can be automated and virtualized form an end user and content provider point of view. The second challenge however have not been addressed fully and affects the overall cost of transcoding services. Our work tries to address the second challenge by introducing a proactive computing resource managment approach for video transcoding services.

2.2. Cloud Environment

Cloud computing provides access to large amount of computing resources (e.g. CPU and storage) in a fully virtualized manner, delivering computing as a utility based on pay-as-you-go business model. This allows businesses and individuals to access applications such as video transcoding from anywhere in the world on demand without being concerned about how and where it is done [Armbrust et al. 2010]. The main distinguishing features of cloud computing are: 1) scalability where users can request seemingly unlimited resources 2) abstract interface which are based on WEB 2.0 protocols that enable easy interaction for users 3) economics of scale where a pool of resources are utilized more efficiently by many users as per demand leading to cost reduction 4) dynamic configureability where users are provided with a simple measuring unit that they can utilize to easily and dynamically reconfigure Cloud resources they require.

In recent years cloud computing has become attractive for the IT industry. This is in particular due to its pay as you go business model which allows small start-ups or private users to gain access to high quality IT infrastructure for a small cost. With such a business model start-ups and private users will avoid building their own infrastructure before they prove the susses of their business model and cloud providers will benefit from the economics of scale.

Among other applications video transcoding is one application that is utilizing the benefits of Cloud [Inc 2013c; 2013b; 2016]. Video transcoding is a computationally expensive process and requires large scale computing infrastructure. At the same time media content producers and other entities that require transcoding services would be better off concentrating in creation of original content where their expertise lie than building and maintaining their own computing infrastructure. Just like these media content producing companies do not own their electric power plants they now do not need to own their own datacenters as it has become a utility that can be bought readily online when needed. Currently there are multiple cloud based transcoding service providers. These transcoding services utilize the cloud infrastructure (IaaS) to provide their services as Platform as a Service (PaaS) or Software as a Service (SaaS). Various research studies are being carried out in the area of large scale video transcoding in the cloud which will allow efficient management of resources and better usability [Garcia et al. 2010; Ko et al. 2013; Aparicio-Pardo et al. 2015].

2.3. Machine Learning

Prediction of video transcoding time is important for several reasons. For example, resource management algorithms deployed on a large scale transcoding service such as [Inc 2013a] can utilize this prediction to increase system utilization through proper load balancing and provisioning. Previous works such as [Guo and Bhuyan 2006] has proposed to model the prediction problem by characterizing a video using simple and few features (e.g. size alone). However such models did not account for variability of video coding algorithms and other factors that make prediction more complex. In this paper we propose the use of more video characteristics and machine learning for better accuracy and generality over a range of coding algorithms. Machine learning techniques are often used as decision making mechanisms for a variety of systems. Basically, machine learning allows computers to evolve behaviours based on empirical data, in our case, this is a collection of samples with important video characteristics, transcoding parameter sets and measured transcoding times. This means video transcoding time is treated as a random variable and is statistically predicted from past observations. More specifically our proposed method predicts the transcoding time as a function of several parameters of the input and output video stream.

The fundamental idea behind any regression problem in machine learning algorithms such as SVR and the Neural Net can be summarized as: given a set of t observations with n features (in our case the input and output bitrate, framerate, codec, etc) each and a target variable (transcoding time) y as $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_t, y_t)\}$ where $\mathbf{x} \in \Re^n$, $y \in \Re$ the objective is to find a function (model)

$$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b \text{ with } \omega \in \Re^n, b \in \Re \tag{1}$$

with the best fit as in equation 1.

Neural nets The idea of neural networks was first inspired by nervous system of human beings which consists of a number of simple processing units called neuron. Each neuron receives some input signals from outside or from other neurons and processes them with an activation function to produce its output and sends it to other neurons. These neurons can be understood as a mathematical function that take n element input vector and scale each data element x_i, by a weight w_j. The scaled data is offset by some bias b and put through a differentiable activation function such as Equation 2. The output of a neuron can be analytically viewed as Equation 1. The impact of each input is weighted differently from other inputs thus a neuron is able to interpret the data differently depending on the weight and bias. Consequently the more is the weight the stronger the connection would be allowing that data point to influence the output more. The activation function f can be linear or non-linear. Non-linear activation functions are useful in mapping non-linear relationships. One such function is called sigmoid which is represented as:

$$\frac{1}{1 + exp(-f)} \tag{2}$$

A network of these neurons forms a feed forward multilayer neural networks. These networks are made of layers of neurons. The first layer is the layer connected to the input data. After that there could be one or more middle layers called hidden layers. The last layer is the output layer which shows the results. One of the learning methods in multilayer perception Neural Networks is the error back propagation in which the network learns the pattern in data set and justifies the weight of the connections in the reverse direction with respect to the gradient vector of error function which is usually the regularized sum of squared error.

Support vector machines treats the regression problem as a convex optimization problem:

$$\text{minimize } \frac{1}{2}\|\omega\|^2 \tag{3}$$

$$\text{subject to } = \begin{cases} y_i - \|w, x_i\| - b \leq \epsilon \\ \|w, x_i\| + b - y_i \leq \epsilon \end{cases} \tag{4}$$

Similar to the neural net the SVR allows for non-linear solution through the use of radial basis function (RBF) kernels which are represented as

$$exp(-\frac{1}{2\sigma^2}\|f\|^2) \tag{5}$$

3. PROACTIVE TRANSCODING SERVICE MANAGEMENT

In this work our main aim is to provide an approach for proactive management of transcoding services driven by transcoding workload prediction. In order to achieve our goal we first provide a machine learning based approach for predicting transcoding time of videos from their basic characteristics and their desired transcoding parameter set. Typically transcoding service providers possess a log about transcoding requests (i.e. both transcoding parameter sets and the original video). Based on these traces, we can build a training dataset listing samples containing transcoding parameter set, the original video (or its fundamental characteristics such as resolution and bitrate) and measured transcoding time. Using such a dataset a prediction model can be trained via machine learning algorithms such as neural network and support vector machines (SVM). Based on the predictions we can then design transcoding service management components. In our case these components constitute a provisioner, load dispatcher and admission controller. In the end our system is aimed at providing the opportunity for transcoding service providers to estimate the transcoding time of requests, and more intelligently manage their transcoding servers. Figure 3 presents the overview of our framework.

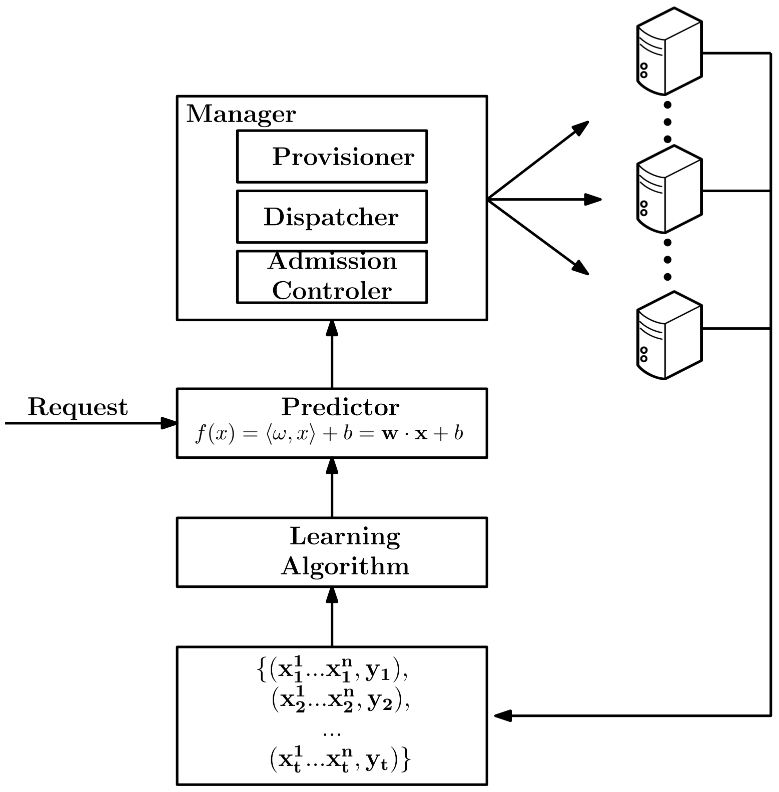

Manager
Provisioner
Dispatcher
Admission Controler
Request
Predictor
$f(x) = \langle \omega, x \rangle + b = \mathbf{w} \cdot \mathbf{x} + b$
Learning Algorithm
$\{(\mathbf{x}_1^1 ... \mathbf{x}_1^n, \mathbf{y}_1), (\mathbf{x}_2^1 ... \mathbf{x}_2^n, \mathbf{y}_2), ... (\mathbf{x}_t^1 ... \mathbf{x}_t^n, \mathbf{y}_t)\}$

Fig. 3. System Architecture

3.1. Transcoding workload modelling and Prediction

Transcoding Time Prediction Model. The prediction model used in this work is depicted in Figure 4. We refer this model as a *transcoding time predictor*. It takes as input a video characterization $C = \{c_1, c_2, ..., c_n\}$ and transcoding parameter sequences $P = \{p_1, p_2, ..., p_n\}$, and it outputs the predicted transcoding time.

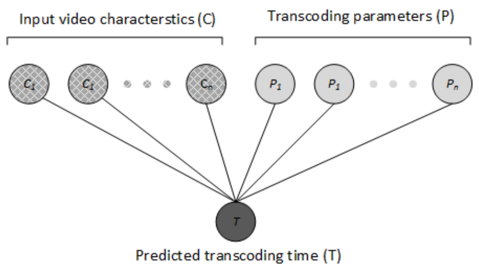

Input video characterstics (C)
Transcoding parameters (P)
C_1
C_1
C_n
P_1
P_1
P_n
T
Predicted transcoding time (T)

Fig. 4. Transcoding time prediction

A general formulation of the transcoding time prediction problem is to construct a function that takes as input easily extractable characteristics of a video together with input parameters that specify the characteristics of the output video and generate an estimated transcoding time of the video on a given platform.

Input Video Characterization. A video can be characterized through a number of basic characteristics such as bitrate, framerate, codec and resolution. To characterize an input video we collect and use all features listed in Table I. This list of video characteristic features are selected through expert analysis of the problem.

Table I. Video characteristics features

Characteristics	Description
Codec	Coding standard used
Resolution(W,H)	Width and height in pixels
Bitrate	Bits processed per second
Framerate	Frames per second
Frmaes (I,P,B, Total)	Number of frames per type
Size (I,P,B, Total)	Size in byte per type

Transcoding Parameters. Transcoding parameters specify the caracterstics of an output video from a transcoder. Table II shows the list of parameters we used in our work. This list is obtained through analysis of randomly mined videos from the YouTube platform [Deneke et al. 2015].

Table II. Transcoding parameter space

Parameter	Value
Codec	H264, Mpeg4, Vp8, H263
Resolution	144p, 240p, 360p, 480p, 720p, 1080p
Bitrate	56k,109k, 242k, 539k, 820k, 3000k, 5000k
Framerate	12, 15, 24, 25, 29.97

Figure 5 shows the effect of some of the input and output video characteristics listed in Table I and Table II on the transcoding time of video segments from randomly sampled online YouTube videos. The figure shows the relationship between the dependent and independent variables under a controlled experiment where only one independent variable (i.e. predictive feature) is monitored while the rest are kept constant. Even though the plots show correlation between our independent variables and transcoding time, the relationship is often non linear. This means we either need to pre-process (be able to apply a proper transformation) of our predictive features to achieve linear relationship with our target variable or use a non-linear learning algorithm to build our prediction models. In this work we choose the later as 1) our main aim is to predict rather than explain and 2) the number of our predictive features are considerable due to the need to support multiple video coding algorithms [Shmueli 2010; Breiman 2001].

Collecting Transcoding Time. When considering all possible combination of the transcoding parameters from Table II in constructing valid transcoding parameter sequences, we will end up with a set of 840 valid transcoding sequences. However in this work we only use 90 of the most important and meaningful transcoding sequences. For example we combined resolution and bitrate parameters in such a way that only the most appropriate bitrate is used for a given resolution. We then collected the transcoding time obtained for each transcoding sequence on a set of 84 randomly selected 60 second segment of YouTube videos. The resulting training data contains 7560 transcoding measurements (instances).

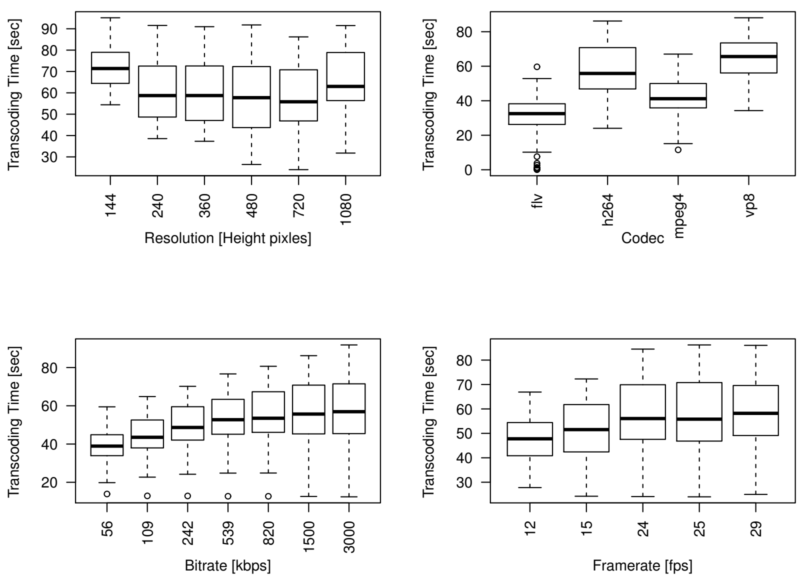
The characterization of each online video along with a transcoding parameter and the corresponding measured transcoding time makes up an instance of our training data. We used Ffmpeg and Ffprobe (version N-73166-g72e98fa) built with gcc 4.7 and runnning on an Intel i7-3720QM CPU to collect video transcoding time and video characteristics of each video in our training set.

Constructing the Model. Once the training data is constructed, it can be fed to a learning algorithm that will automatically learn a prediction model. We used support vector regression (SVR) and multilayer perceptron (MLP) to construct our prediction models. Support vector machines (SVR) is a supervised machine learning algorithm, used for regression, which applies linear techniques to non-linear problems. The idea of SVR is based on the computation of a linear regression function in a high dimensional feature space where the training data are mapped via a non linear kernel function. SVRs not only finds a regression function, but it also finds the best function, i.e., minimize the generalized error bound so as to achieve generalized prediction performance. In our experiments we used LibSVM package from RapidMiner [rap 2012] with radial basis function (RBF) kernel, γ set to 0.125 and error penalty term C set to 1024. We found these parameter settings using grid search method [wei Hsu et al. 2010]. MLP (MultiLayer Perceptron) is also a supervised learning algorithm which learns a model by means of a feed-forward neural network trained through a back propagation algorithm (i.e. multi-layer perceptron). An artificial neural network (ANN or NN), is a mathematical model or computational model that is inspired by the structure and functional aspects of biological neural networks. A neural network consists of a set of interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases an NN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are usually used to model complex relationships between inputs and outputs or to find patterns in data. In this work we used feed-forward neural network trained by a back propagation algorithm (multi-layer perceptron) from RapidMiner [rap 2012] to model transcoding time of online videos. The training cycle, the learning rate and the momentum are set to 500, 0.3 and 0.2 respectively.

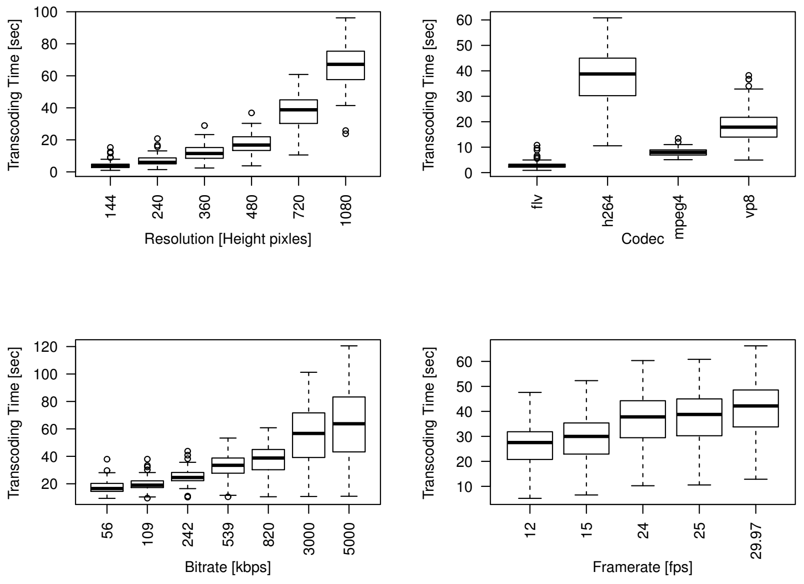
Model on Unseen Video. We train and validate our models using 2/3 of the training data we have collected and the rest 1/3 portion is left for testing and evaluation of the models.

3.2. Proactive Provisioning

Due to the variability of individual video transcoding jobs (see Figure 1) and the variable transcoding requests made towards transcoding services (see Figure 6), it is important to have a mechanism for proactive provisioning of transcoding servers. Without such mechanism, video transcoding services would have to resort to a fixed computing capacity which leads to either over-provisioning or under-provisioning of com-



(a) Effect of input video characteristics on transcoding time of a set of 80 Youtube video segments when transcoding using a fixed transcoding parameter set. I.e. transcoding to 3000 kbps, 25 fps, 720p using h264 codec. In this case the transcoding parameters are fixed to show the effect of each input video characteristics on the total transcoding time



(b) Effect of transcoding parameters on transcoding time of a set of 80 YouTube video segments. The effect of each transcoding parameter is shown while keeping the other parameters fixed. Each box plot corresponds to transcoding time values of the 80 randomly selected YouTube videos. The set videos is fixed and all transcoding parameters except the controlled parameters is fixed

Fig. 5. Effect of input video characteristics and transcoding parameters

puting resources for a required level of Quality of Service. In this paper we propose a proactive provisioning approach based on ahead of time transcoding workload prediction. Algorithm 1 shows our approach. It runs periodically and starts by calculating the number of servers to provision. This number is calculated trough the formula:

$$\left(\left\lceil \frac{predLoad(servers)}{\alpha * slaWaitingTime} \right\rceil - 1\right) * len(servers) \tag{6}$$

where $predLoad(servers)$ is the average predicted load of servers, $slaWaitingTime$ is the required system wide average waiting time for transcoding jobs and α is an adjusting parameter that can be used to compensate prediction errors. The average predicted load of servers for a given period is calculated by summing up workload predictions of all transcoding tasks assigned to servers and dividing it by the number of servers at the same period. Such averages from past and current period are then smoothed using exponential moving average technique. They are used by simple regression model to predict the average predicted load of the next period, which we use in our formula for calculating the number of VMs. The formula calculates the number of servers to add or remove in order to get an average load on each server as close as possible to the waiting time specified by the SLA of the service. This ensures provisioning the right amount of servers for the agreed Quality of Services. Note that a negative result from the formula indicates the number of servers to be removed while a positive number indicates the number of servers that needed to be added. Since most cloud providers that host transcoding services charge for their virtual machines on hourly bases, the provisioning algorithm checks the servers renting time before removal. Servers are removed only if they are near to the end of their renting time and have no job under processing. Servers that have running jobs and are near to the end of their renting time are flagged. A flagged server will not receive any more jobs until it is removed or become unflagged due to a system load increase, in which case its renting time will be renewed. Similarly when the system needs to add servers, it will start first by unflagging flagged servers. This allows the servers to receive new jobs and their renting time to be renewed. Therefore the algorithm provisions new servers only when there are no flagged servers at disposal.

3.3. Proactive Load Balancing

Requests arriving to transcoding services need to be assigned to a specific transcoding server for processing. We call this process load dispatching or load balancing. In this paper we propose a proactive load dispatching algorithm driven by a transcoding time prediction and is listed in Algorithm 2. The algorithm selects the server to dispatch the current request to based on the predicted total load of transcoding servers in the system. Specifically the current request is dispatched to a server with the least load. However just before the request is dispatched, its load is predicted and is added to the current load of the server it is being sent [Deneke et al. 2014].

3.4. Proactive Admission Control and SLA

SLA is mainly used to communicate the quality of service to be expected from a given system. In this work we employ a fixed SLA in terms of average waiting time of transcoding jobs and provision the required number of servers that satisfy the SLA. In case the average waiting time used as SLA is defined to be much smaller than the time it takes to initialize a new server an admission controller is required. This is a typical case for realtime systems and we will not consider it in this particular work. However since we predict the system load ahead of time through regression of past and current average predicted loads, we are able to start provisioning of VMs in advance and avoid the SLA violations that might occur in case of strict SLA.

ALGORITHM 1: Proactive Provisioning Algorithm

```
Require: $servers$
 1: $toProvision \leftarrow \left(\left\lceil \frac{predLoad(servers)}{\alpha * slaWaitingTime} \right\rceil - 1\right) * len(servers)$
 2: if $!ratecentlyModified()$ then
 3:   if $toProvision > 0$ then
 4:     for all $s$ in $servers$ do
 5:       if toProvision == 0 then
 6:         break
 7:       end if
 8:       if isFlagged(s) then
 9:         unflag(s)
10:         $toProvision --$
11:       end if
12:     end for
13:     $servers.add(provision(toProvision))$
14:   else
15:     for all $s$ in $servers$ do
16:       if $toProvision == 0$ then
17:         break
18:       end if
19:       if $isNearEndOfRenting()$ and $!hasJobs()$ then
20:         $server.remove(deprovision(s))$
21:         $toProvision ++$
22:       end if
23:     end for
24:     for all $s$ in $servers$ do
25:       if $toProvision == 0$ then
26:         break
27:       end if
28:       if $isNearEndOfRenting()$ then
29:         $flag(s)$
30:         $toProvision ++$
31:       end if
32:     end for
33:   end if
34: end if
```

ALGORITHM 2: Proactive Load Balancing Algorithm

```
Require: $requests, serverIds, algo$
Ensure: $serverId = id$
 1: $server \leftarrow servers(0)$
 2: for all $req$ in $requests$ do
 3:   for all $s$ in $servers$ do
 4:     if $predLoad(s) < predLoad(server)$ then
 5:       $server \leftarrow s$
 6:     end if
 7:   end for
 8:   $send(req, server)$
 9:   $load(req) \leftarrow predictTranscodingTime(req, algo)$
10:   $predLoad(server) \leftarrow predLoad(server) + load(req)$
11: end for
```

4. EXPERIMENTAL SETUP AND EVALUATION

4.1. Load Generation

Load of transcoding services is determined by two important variables, service time distribution of transcoding tasks and the arrival rate of tasks in to the system. Figure 6 shows a scaled down form of a one week request arrival rate pattern for a typical streaming service where the inter arrival time is 5 seconds [bam 2007]. We scaled the arrival rate by a factor since streaming requests are often a factor of times more than transcoding requests of a system. This means a transcoded version of a given video is often streamed multiple times. We used this request pattern in our experiments in this work. From the figure we can also note the daily periodicity of the request rate over the week. The variation in terms of request rates are also apparent over the course of a day which ranges from 5 requests to 30 requests in a 5 second interval. This can be explained by the heavy used of streaming services in the evening as compared to mornings and late nights. Such load variation leads to the need for proper resource provisioning and utilization.

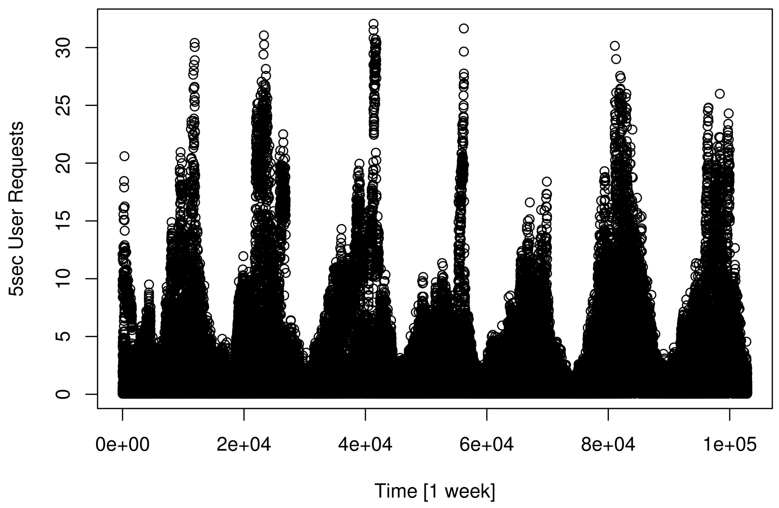


Fig. 6. 5sec user accesses distribution during the course of a single week.

The service time of transcoding jobs in our experiment is obtained from an experimental data collected and described in Section 3.1.

4.2. Evaluation of Workload Prediction

We train and validate our prediction models using 2/3 of the training data we have collected in Section 3.1. We then evaluated our prediction models using the remaining 1/3 portion of the transcoding time measurement data. Figure 7 shows the correlation between the prediction and the actual transcoding time based on a neural network and support vector machine learning algorithms. More specifically it shows the squared correlation (predicted vs measured) to be 0.955 for neural network and 0.945 for support vector machine. Our results also show a root mean absolute error to be 2.382 and 2.62 seconds for neural network and support vector machine respectively.

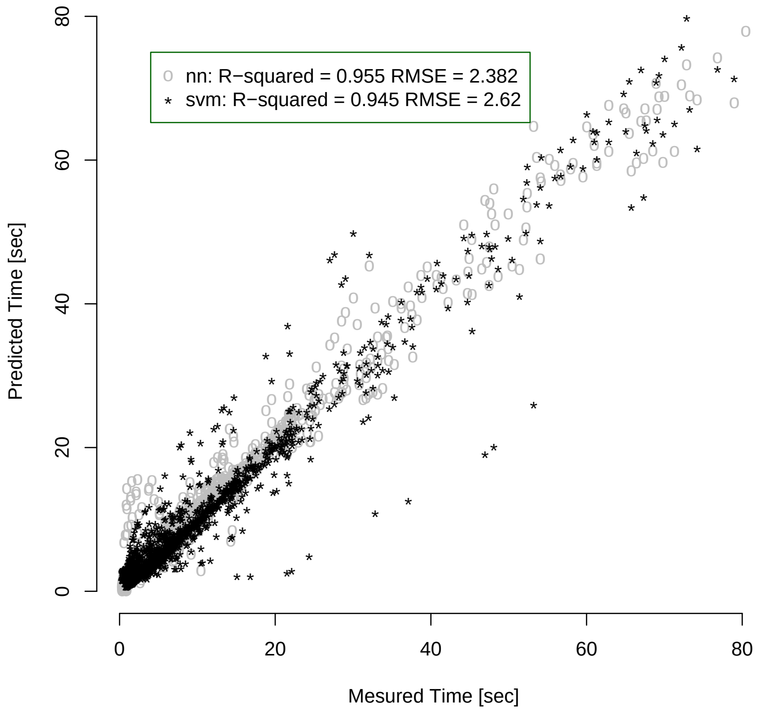


Fig. 7. Prediction Performance

4.3. Evaluation of Proactive Provisioning

To evaluate our proactive provisioning and load balancing schemes, we simulate their VM provisioning and job waiting time performances using CloudSim [Calheiros et al. 2011]. Under our CloudSim simulation environment, we deploy nodes and specify CPU and virtual machine characteristics according to the machine we used for transcoding time data collection (Intel i7-3720QM) in Section 3.1. Figure 8 shows the transcoding server provisioning results for 1 minute interval based on the proactive provisioning Algorithm 1 while using the neural net or support vector machine algorithms for prediction of transcoding time of tasks. The performance of the provisioning algorithm is mostly similar when using either of the two algorithms for task workload prediction. This result can also be attributed to the similar workload prediction performance of the two algorithms shown in Figure 7.

Note that without such proactive VM provisioning, we would need to provision based on the worst case scenario which leads to over provisioning most of the time. In our experiment the number of VM hours saved when using proactive provisioning rather

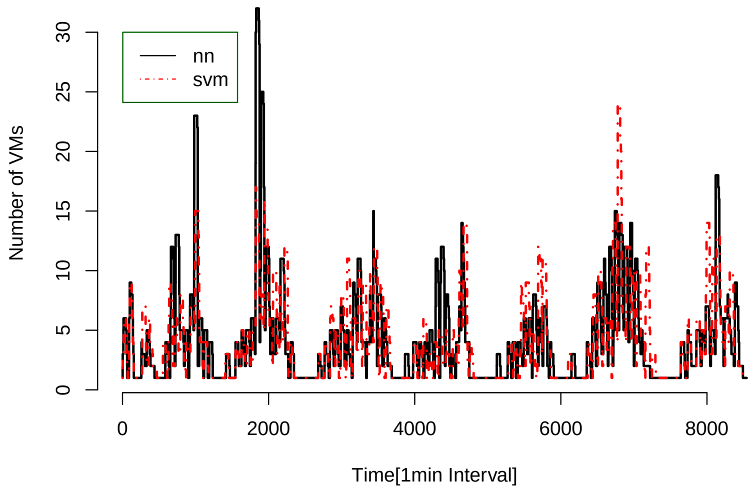


Fig. 8. Proactive VM provisioning

than fixed worst case based provisioning was 3941 hours (86%) over the course of a week.

4.4. Evaluation of Proactive Load Balancing

Figure 9 shows the effect of load balancing on proactive provisioning of VMs. Provisioning VMs using our proactive approach results in the right amount of VMs for the required SLA in terms of waiting time. This also insures that all servers are always loaded with tasks requiring a CPU time equivalent to the average waiting time defined in the SLA. In such a case, where all VMs are loaded all the time, the type of load balancing used has limited effect on the number of VMs provisioned. Figure 9 also shows this fact, where the use of queue length as well as predicted queue load based on neural network for load balancing as in Algorithm 2 results in similar VMs provisioning except in certain pick load conditions where being proactive is better. In our experiment the number of saved VM hours when using proactive load balancing rather than queue length was 51 hours (8%) over the course of a week.

4.5. Evaluation of Proactive Admission Control and SLA

Figure 10 shows the average of the actual waiting time of jobs over 1 minute interval along with the service level agreement. From the Figure we can note that almost all jobs have respected the SLA set in terms of their average waiting time.

5. RELATED WORK

Among others our work is related to video transcoding workload prediction and proactive management of computing resources. In this section we briefly summarize the related works on each.

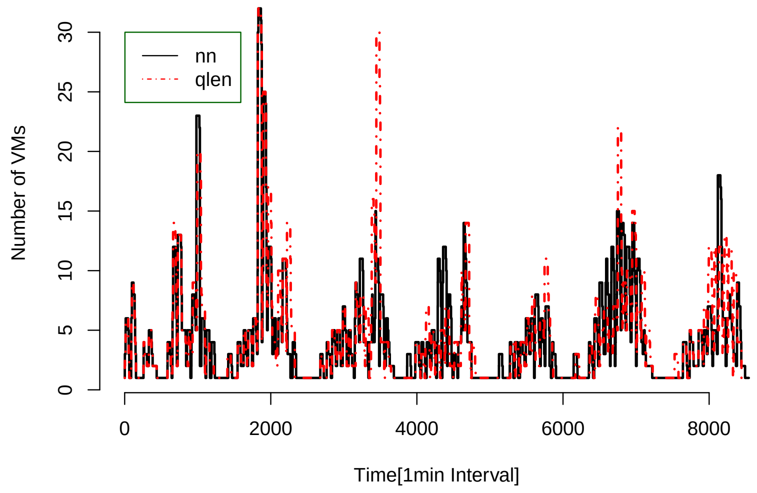


Fig. 9. Comparison of Load Balancing Algorithms

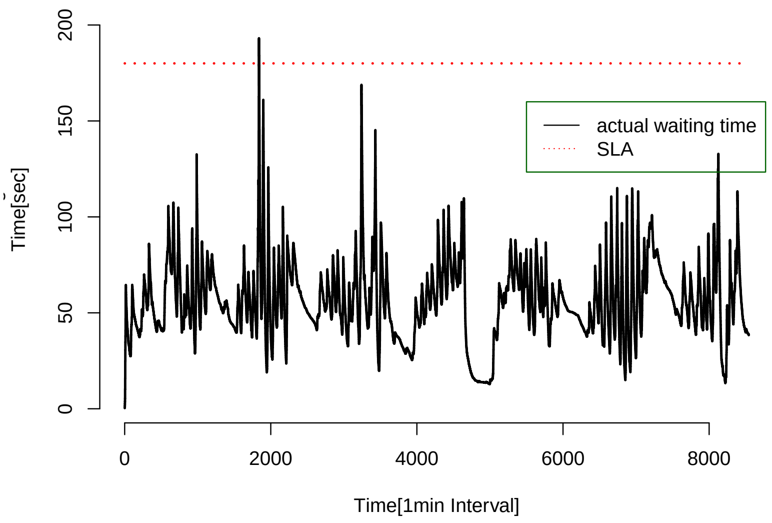


Fig. 10. SLA and actual waiting time of Jobs

5.1. Transcoding workload prediction

Several workload prediction models for video processing applications have been proposed in the literature. The existing models can be classified into two categories: models based on history and models relaying on information extracted from the video bitstream. In the history based methods the workload of the current work item (e.g. a frame) is predicted as weighted average of previous work items [Choi et al. 2002; Bavier et al. 1998; Guo and Bhuyan 2006; Jokhio et al. 2013]. However due to the large variability in video processing workload of consecutive work items, the history-based models often suffer in terms of accuracy. Consequently research in this area focus on improving prediction accuracy. On the other hand models based on information extracted from video bitstream predict future workload based on models constructed from predictive features obtainable from the bitstream [Mattavelli and Brunetton 1998; Huang et al. 2005; Huang et al. 2007; Roitzsch and Pohlack 2006]. These models often tend to be more accurate but incur more overhead due the time required for video bitstream parsing. Research in this area thus mostly focus on finding less expensive yet predictive features.

In [Choi et al. 2002] the decoding time of current frame is predicted based on a moving-average over past frames of a similar type. This means that decoding time prediction of I, P and B are done separately. The resulting prediction is then used for scaling the processor voltage and frequency in order to provision the proper amount of computing power required to decode a frame.

[Bavier et al. 1998] proposed a model which can predict video decoding workload at frame and network packet level. They utilized linear regression analysis in order to gain insight on relationship between MPEG bitstream components and decoding time of a frame or a packet. Their frame level predictor uses running average of past frames of similar type while taking in to account the byte length of the frames. Similarly their packet level predictor uses a running average of past frames of similar type taking in to account the number of blocks of the packets. Their prediction approach is designed to be computationally cheap and usable for real-time multimedia application scheduling.

[Guo and Bhuyan 2006] determined the necessity of predicting the CPU load of transcoding tasks in order to schedule them on a cluster of computing nodes. They, therefore, proposed an online prediction algorithm that can dynamically predict the processing time of video segments (GOPs). Their predictor is a linear model where its slope is incrementally approximated according to the difference between accumulated regional and global means of GOP transcoding times and sizes.

[Huang et al. 2005] proposed a workload prediction technique for video decoding which is based on an offline bitstream analysis of a video. The predictions are then used to insert metadata information, a sequence of frequency values, with which the processor needs to run while decoding various segment of the video. The ultimate goal of their work is saving energy while decoding video streams. In a later work [Huang et al. 2007] further proposed a new workload-scalable transcoding scheme which converts a pre-recorded video bitstream into a new video bitstream that satisfies a given playback device workload constraint, while keeping the transcoding distortion minimal as measured in terms of their proposed compressed domain distortion measure, a function of frames per second and bits per frame.

[Roitzsch and Pohlack 2006] presented the design and implementation of per-frame decoding time prediction method for MPEG based video decoders. In order to find useful prediction metrics, they divided the decoding process in to logical steps and established metrics from the video bitstream that are useful to get reasonable execution time estimates for each step. They then modeled the prediction problem as linear least square problem and solved the model coefficients using a training dataset collected over test video sequences.

Most of the related works in the area of multimedia workload modelling and prediction focus on one coding algorithm and use only a brief set of videos to train their prediction models. Our transcoding workload modeling and prediction is designed to work across multiple coding algorithms. It is also modeled based on our video dataset having a realistic distribution of online video characteristics.

5.2. Proactive Transcoding Resource Management

Most of the research work in proactive management of computing resources for video transcoding applications and services focus on such things as loadbalancing, provisioning and task migration. These works are based on and mostly follow from workload modelling and prediction works presented in subsection 5.1.

[Choi et al. 2002] used a moving average based frame decoding workload prediction for scaling the processor voltage and frequency so that they provision the proper amount of computing power required to decode a frame. In [Jokhio et al. 2013] prediction-based dynamic resource allocation algorithm to scale video transcoding service on a given Infrastructure as a Service cloud were discussed. The proposed algorithm provides mechanisms for allocation and deallocation of virtual machines based on a regression model that tracks and predicts the aggregate target transcoding rate required by the service. In [Huang et al. 2005] and [Huang et al. 2007] the authors applied frequency scaling for proper provisioning of computing resources for decoding and transcoding applications. The frequency value at which the processor should run at specific times during the video play back or transcoding is inserted in to the video bitstream.

In [Guo and Bhuyan 2006] the authors proposed a loadbalncing scheme based on prediction of video transcoding workload at GOP level. Their main aim was to minimize the total processing time while maintaining the order of media units for each outgoing stream. In their work they designed and evaluated algorithms such as First Fit (FF) and Adoptive Load sharing (ALS). [Kuang et al. 2010] proposed and evaluated a power-efficient and traffic aware transcoding system for multicore servers. The approach manages computing resources by adjusting processor operating levels that match the incoming traffic rate. More specifically their approach is capable of configuring the number of active cores and core frequency on-the-fly according to the varying traffic rate.

In our work we used transcoding workload prediction models that relay on information extracted from the bitstream as the variability of the workload across a sequence of frames is high. Based on these prediction models we then show how it is possible to efficiently manage computing resources. We demonstrate our idea through evaluation of a proactive loaadbalancing approach.

5.3. video characterization

There has been significant research on understanding the workloads of new generation video servers. These researches especially focus on the social aspect of videos and traffic characterization such as popularity, active life span, user access pattern, growth pattern, request patterns, etc.

Yu et al. [Yu et al. 2006] study user behaviour, content access pattern and their implications on the design of large-scale video-on-demand systems. Possible improvements on UGC design were proposed by Cha et al. [Cha et al.] after studying YouTube and Daum, a popular UGC in Korea. After tracking YouTube transactions from a network edge, Gill et al. [Gill et al.] have tried to understand video access characteristics and discuss the implications of their observation on key concepts such as caching. The caching problem in YouTube has been further studied by Zink et al. [Zink et al. 2008]. The social networking among videos was studied in the works of Halvey et al. [Halvey and Keane] and Mislove et al. [Mislove et al. 2007].

In this work we will reuse the traffic model from [Yu et al. 2006] to drive our experiments but further focus on collecting the missing statistics on video characteristics such as video length, size, bitrate, frame rate, codec type, resolution and etc that will be useful in our experiments.

6. CONCLUSION

The main focus of this work was the different research issues related to proactive management of transcoding services. First we showed how one can construct a transcoding workload prediction model based on past transcoding measurements, a set of easily extractable input video features and a set of transcoding parameters. Based on our prediction models, we then designed proactive computing resource management algorithms which mainly include provisioning and load balancing. The provisioning algorithm enables transcoding services to always maintain just the right number of VMs that are needed to maintain the required quality of service defined in terms of average waiting time of jobs. The load balancing algorithm is useful to get beter performance of the system in terms of worst waiting time of jobs. In our experiments, we have used real-world data and designed the simulation scenario imitating real world Internet transactions. Our proactive provisioning and load balancing algorithms showed a significant improvement over the traditional methods in terms of number of VMs used and worst case waiting time of jobs.

REFERENCES

2007. Bambuser. Online. (Sept. 2007). http://bambuser.com/

2012. RapidMiner. Online. (April 2012). http://rapid-i.com/content/view/181/190/

Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. 2015. Transcoding Live Adaptive Video Streams at a Massive Scale in the Cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, New York, NY, USA, 49–60. DOI:http://dx.doi.org/10.1145/2713168.2713177

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei"i Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (April 2010), 50–58. DOI:http://dx.doi.org/10.1145/1721654.1721672

Andy C. Bavier, A. Brady Montz, and Larry L. Peterson. 1998. Predicting MPEG Execution Times. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '98/PERFORMANCE '98)*. ACM, New York, NY, USA, 131–140. DOI:http://dx.doi.org/10.1145/277851.277892

Leo Breiman. 2001. Statistical modeling: The two cultures. *Statist. Sci.* (2001).

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, De Rose ;sar A. F, and Rajkumar Buyya. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Softw. Pract. Exper.* (2011).

Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system *(IMC '07)*. San Diego, California, USA.

S. F. Chang and A. Vetro. 2005. Video Adaptation: Concepts, Technologies, and Open Issues. *Proceedings of IEEE* (Jan. 2005). http://dx.doi.org/10.1109/JPROC.2004.839600

Kihwan Choi, K. Dantu, Wei-Chung Cheng, and M. Pedram. 2002. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*. 732–737. DOI:http://dx.doi.org/10.1109/ICCAD.2002.1167613

T. Deneke, H. Haile, S. Lafond, and J. Lilius. 2014. Video transcoding time prediction for proactive load balancing. In *2014 IEEE International Conference on Multimedia and Expo (ICME)*. 1–6. DOI:http://dx.doi.org/10.1109/ICME.2014.6890256

T. Deneke, S. Lafond, and J. Lilius. 2015. Analysis and Transcoding Time Prediction of Online Videos. In *2015 IEEE International Symposium on Multimedia (ISM)*. 319–322. DOI:http://dx.doi.org/10.1109/ISM.2015.100

Adriana Garcia, Hari Kalva, and Borko Furht. 2010. A Study of Transcoding on Cloud Environments for Video Content Delivery. In *Proceedings of the 2010 ACM Multimedia Workshop on Mobile Cloud Media Computing (MCMC '10)*. ACM, New York, NY, USA, 13–18. DOI:http://dx.doi.org/10.1145/1877953.1877959

Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge *(IMC '07)*. San Diego, California, USA.

Jiani Guo and Laxmi Narayan Bhuyan. 2006. Load Balancing in a Cluster-Based Web Server for Multimedia Applications. *IEEE Trans. Parallel Distrib. Syst.* 17, 11 (Nov. 2006), 1321–1334. DOI:http://dx.doi.org/10.1109/TPDS.2006.159

Martin J. Halvey and Mark T. Keane. Exploring social dynamics in online media sharing *(WWW '07)*. Banff, Alberta, Canada.

Mor Harchol-balter. 1999. The Effect of Heavy-Tailed Job Size Distributions on Computer System Design. In *In Proc. of ASA-IMS Conf. on Applications of Heavy Tailed Distributions in Economics*.

Yicheng Huang, Samarjit Chakraborty, and Ye Wang. 2005. Using Offline Bitstream Analysis for Power-aware Video Decoding in Portable Devices. In *Proceedings of the 13th Annual ACM International Conference on Multimedia (MULTIMEDIA '05)*. ACM, New York, NY, USA, 299–302. DOI:http://dx.doi.org/10.1145/1101149.1101209

Yicheng Huang, An Vu Tran, and Ye Wang. 2007. A Workload Prediction Model for Decoding Mpeg Video and Its Application to Workload-scalable Transcoding. In *Proceedings of the 15th ACM International Conference on Multimedia (MM '07)*. ACM, New York, NY, USA, 952–961. DOI:http://dx.doi.org/10.1145/1291233.1291443

Amazon Inc. 2013a. Amazon EC2 Instance Types. (Aug. 2013). http://aws.amazon.com/ec2/instance-types/

Amazon Inc. 2013b. Amazon Elastic Transcoder. (Aug. 2013). http://aws.amazon.com/elastictranscoder/

Bitmovin Inc. 2016. Bitmovin Cloud Encoding. (May 2016). https://bitmovin.com/encoding/

Zencoder Inc. 2013c. Zencoder Cloud Transcoder. (Aug. 2013). http://zencoder.com/en/

Fareed Ahmed Jokhio, Adnan Ashraf, Sebastien Lafond, Ivan Porres, and Johan Lilius. 2013. Prediction-Based Dynamic Resource Allocation for Video Transcoding in Cloud Computing. In *PDP, 2013*.

Seungcheol Ko, Seongsoo Park, and Hwansoo Han. 2013. Design Analysis for Real-time Video Transcoding on Cloud Systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, New York, NY, USA, 1610–1615. DOI:http://dx.doi.org/10.1145/2480362.2480663

J. Kuang, D. Guo, and L. Bhuyan. 2010. Power optimization for multimedia transcoding on multicore servers. In *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*. 1–2.

Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei Dai. 2012. Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices. In *22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*.

Marco Mattavelli and Sylvain Brunetton. 1998. *Real-time constraints and prediction of video decoding time for multimedia systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 425–438. DOI:http://dx.doi.org/10.1007/3-540-64594-2_113

Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07)*. ACM, New York, NY, USA, 2942. DOI:http://dx.doi.org/10.1145/1298306.1298311

M. Roitzsch and M. Pohlack. 2006. Principles for the Prediction of Video Decoding Times Applied to MPEG-1/2 and MPEG-4 Part 2 Video. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*. 271–280. DOI:http://dx.doi.org/10.1109/RTSS.2006.36

Galit Shmueli. 2010. To Explain or to Predict? *Statist. Sci.* 25, 3 (08 2010), 289–310. DOI:http://dx.doi.org/10.1214/10-STS330

I. Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *MultiMedia, IEEE* 18, 4 (April 2011), 62–67. DOI:http://dx.doi.org/10.1109/MMUL.2011.71

Thomas Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, New York, NY, USA, 133–144. DOI:http://dx.doi.org/10.1145/1943552.1943572

Laura Toni, Ramon Aparicio-Pardo, Gwendal Simon, Alberto Blanc, and Pascal Frossard. 2014. Optimal Set of Video Representations in Adaptive Streaming. In *Proceedings of the 5th ACM Multimedia Systems Conference (MMSys '14)*. ACM, New York, NY, USA, 271–282. DOI:http://dx.doi.org/10.1145/2557642.2557652

A. Vetro, C. Christopoulos, and H. Sun. 2003. Video transcoding architectures and techniques: An overview. In *Signal Processing Magazine, IEEE*. 1829.

Chih wei Hsu, Chih chung Chang, and Chih jen Lin. 2010. A practical guide to support vector classification. (2010).

J. Xin, C.-W. Lin, and M.-T. Sun. 2005. Digital Video Transcoding. *Proc. IEEE* (Jan. 2005). DOI:http://dx.doi.org/10.1109/JPROC.2004.839620

Hongliang Yu, Dongdong Zheng, Ben Y. Zhao, and Weimin Zheng. 2006. Understanding user behavior in large-scale video-on-demand systems. *SIGOPS Oper. Syst. Rev.* (April 2006).

Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. 2008. *Watch global, cache local: YouTube network traffic at a campus network - measurements and implications*. Technical Report.

ISBN 978-952-12-3536-8
ISSN 1239-1883

Tewodros Deneke

Proactive Management of Video Transcoding Services

Tewodros Deneke

Proactive Management of Video Transcoding Services