Distributed Multimedia Processing Using Opportunistic Resources

by

Daskalakis Thomas, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Daskalakis Thomas

September 1, 2008

Acknowledgments

First of all i would like to thank my supervisor Jonathan Dukes for his advice, guidance, feedback and non-stop support during the year.

I would also like to express my love and gratitude to Angie, Paul and Helen for the endless support that they gave me and for truly believing in me since the day i was born.

Last but not least i would like to thank (almost) all my classmates for making this last year a really fascinating and exciting experience combining hard work and fun.

Daskalakis Thomas

University of Dublin, Trinity College

September 2008

Distributed Multimedia Processing Using Opportunistic Resources

Daskalakis Thomas, M.Sc.
University of Dublin, Trinity College, 2008

Supervisor: Dukes Jonathan

The increasing interest in mobile multimedia streaming applications demands an in-depth understanding of all the factors that can possibly affect their success. These applications have to cope with problems like limited bandwidth, low capabilities and heterogeneity of the devices. The need for an architecture that would consider all the previous points and provide acceptable levels of QoE is obvious. In this dissertation, the design, implementation and evaluation of an application that supports transcoding on-demand using opportunistic resources is described. Personalized multimedia streams are served to mobile clients using a transcoding on-demand scheme on opportunistic resources while taking the capabilities of each client's device into account. While the use of opportunistic resources provides numerous benefits to an application a time overhead is introduced which affects the overall QoE. We evaluate the overhead and suggest solutions to optimize the latency introduced by the use of opportunistic resources using standard non-modified protocols and tools.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile applications are becoming popular and constantly penetrating deeper into the market. A great amount of those applications are based on multimedia. Devices used as clients, range from simple mobile phones with MP3 and Video players to more advanced smartphones that are able to receive and playback network streams over the mobile network using technologies like 3G. In this context, some issues arise concerning the requirements that need to be met in order to guarantee a successful mobile multimedia application.

One basic concern is the low capabilities of those devices. Most of the mobile clients used in such applications are handheld devices with limited battery life, low CPU power and memory capacity. In addition to that, different devices might have completely different playback capabilities. One device might be able to playback video encoded using an MPEG4 codec whereas another device might be MPEG2. This heterogeneity is greatly increased if different properties of multimedia content such as frame-rate, bit-rate, aspect-ratio and screen resolution, are taken into account.

The above problems require a solution that will be able to provide all the devices with the content they will be able to playback returning the best result. This means that for a mobile multimedia streaming application, all the devices that are used as clients will receive a multimedia stream that matches their full compliment of factory defined playback capabilities. Thus, in order to produce this personalized optimized multimedia stream for each device, the process of transcoding from one video format to another that matches the device's capabilities, is essential.

1.1 Solution

A typical multimedia streaming application will consist of a number of servers that will be used to store and stream the content, and the clients that will receive the multimedia stream via an infrastructure. The transcoding process has to be injected in the described architecture between the server and the client in order to match the requirements described above. There are different architectures that can be used to achieve that. The simplest one is to pre-transcode the content offline using some *standard* parameters and produce a stream that most of the devices will be capable of recognizing. Although this doesn't meet the above requirements, it can be easily deployed and used at the expense of storage cost and management overhead to maintain all the different versions of the same video file. A different solution could be transcoding on the fly the received stream on the client side. However, this will often be impossible using mobile devices due to the lack of resources, since the process of transcoding is a very intense and demanding process in terms of CPU utilization and memory consumption.

Another approach that could produce the desired result is using a transcoding on-demand scheme. A client would request some content providing its exact specifications and a dedicated transcoding and streaming process would deliver the specialized content to the client. The transcoding can either be done on the host that stores the content or somewhere between the client and the server using a dedicated transcoding farm. However, this will have a huge overhead in terms of maintenance and management and will be inefficient since most of the dedicated transcoders will be staying idle for much of the time. A different approach is proposed in this dissertation that combines the benefits of all these architectures. This approach is outlined in the next section.

1.2 Transcoding Using Opportunistic Resources

The use of dedicated transcoders would provide the desired service but at the expense of high overheads. An optimization is needed to increase the performance of the transcoding farm. This optimization can be done by using opportunistic resources instead of fixed ones. Practically, using resources that are primarily used for another purpose, can be used only during their idle time to perform transcoding tasks. A student computer lab in a university for example provides a large number of resources that most of the time remain idle waiting for a student to log in. Therefore, the proposed architecture is as seen in Fig.1.1. The video servers that store the content stream it to a host of a computer lab and this host transcodes it and streams it to the client. While a host is running a transcoding job and thereafter becomes occupied by a student, the transcoding job automatically migrates to another free host, thereby maintaining the primary usage of the computer lab.

1.3 Aims of this Research

In this project we will investigate how multimedia processing can be done in a distributed environment using opportunistic resources. These resources might be the PCs of a student computer lab, which will be used only when they are idle. The organization and management of the opportunistic resources can be handled by the Condor system [4]. Condor is a workload management system which is capable of executing serial or parallel jobs in a distributed environment while it manages and monitors the available resources. Multimedia processing will be done with the use of open source utilities such as NMM

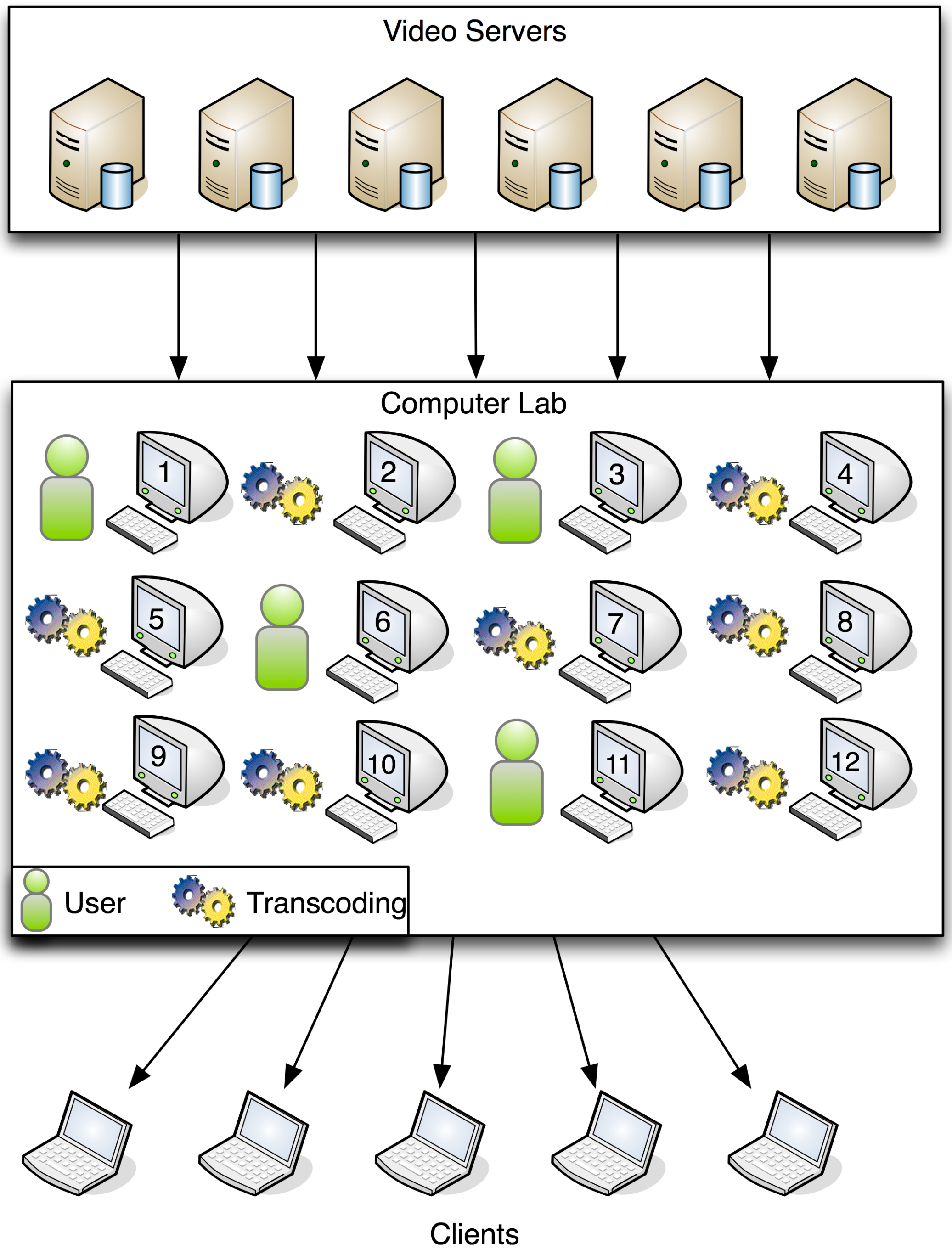

Figure 1.1: Using Opportunistic Resources in a computer lab

[14] or VLC [17].

Condor is well known for its performance as far as cpu intensive batch processes are concerned. However, the result of multimedia processing such as transcoding and streaming is highly effected by latency, asynchronous data distribution and low availability of resources. Thus, in this project we will try to examine how an open source platform such as VLC can be used to perform multimedia processing using opportunistic resources provided by the Condor system.

Our research is done on the basis of a mobile video application. Clients are different mobile handheld devices that are unable to perform any kind of cpu intensive jobs like transcoding, and have limited capabilities in terms of codec and video format support (aspect ratio, frame rate, bit-rate, resolution). All the resources that will be used by this application will be provided by condor.

In Fig.1.1 hosts 1,3,6 and 11 are used by students whereas the rest of them are used for transcoding. Using opportunistic resources to perform transcoding tasks, can help to provide a very efficient architecture without wasting the available resources while at the same time the cost of maintenance and management is small. The subjective measure that is related with the overall experience of the end point client is called QoE. Keeping the available opportunistic resources organized may decrease the overall QoE in two ways: a) by adding a small overhead to the time that is needed for the stream to start and b) by making a change from one transcoder to another noticeable, when the first one becomes occupied in order to perform its primary purpose. It was calculated that an overhead of approximately 1.31 seconds is added to the start of the stream which corresponds to the process of finding an available resource.

1.4 Contribution of the Dissertation

This work provides a proof of concept that multimedia processing is possible using opportunistic resources, and uncovers certain points that need further attention in order to maximize the QoE. A proof of concept system has been designed, implemented and evaluated as described in the later sections.

Chapter 2

Background

2.1 Multimedia Streaming Applications

Video applications are becoming more and more popular. An increasing number of companies and service providers offer solutions that serve video content to customers over different kinds of infrastructures. Infrastructures include the internet, with websites like YouTube.com [19], break.com [3], MetaCafe [13], Google Video [9], the telephone landline network (IPTV services) as well as a variety of wireless communication networks (Wifi, 3G).

Each application might have completely different and unique needs and requirements which every provider must consider in order to provide a video service. Moreover, there is a rising demand for quality of service improvement in order to come to terms with the users' needs. In this context, one of the most important factors that influence the quality of service and the requirements of the hardware, is the video and audio format. The choice of the content's format can greatly affect the success of the application.

There is a great number of competitive video and audio codecs that can be used to serve various clients and would suit better in specific environments. Thus, converting from one video and audio format to another, has become critically essential in order to each time match the requirements and specifications of a specific environment. This procedure is widely known as transcoding [33] and its considered to be one of the most demanding processes in terms of computer resources.

With transcoding being an intense procedure it becomes hard and in most cases impossible to use each client's resources for it, especially if the clients are small devices with serious limitations in terms of CPU power, memory and energy.

2.2 Mobile Multimedia Applications

Mobile video applications include local playback, streaming, telephony/conferencing, and broadcast/multicast applications. There already is a large number of companies worldwide that offer video services over a mobile network and it has proven to be a very lucrative business as seen in Fig.2.1. Gartner reported in 2007 [27], that the number of mobile TV subscribers worldwide in 2006, 2007 was 17 and 60 million respectively, whereas in 2010 mobile TV subscribers will reach 489 million. The constant rising popularity has created the need to examine all the possible opportunities that would make a mobile video application more efficient and competitive.

The most serious limitations on a mobile video application are imposed by the client itself. Mobile clients' hardware limitations include smaller screen size, less color depth, low CPU, low memory and low energy resources [24]. Moreover, a successful mobile video application should be compatible with as many mobile devices as possible and should serve the content offering the maximum quality that is available for a specific device [29]. Some devices are only able to recognize specific video formats in specific screen resolutions and certain bitrates; for example Apple's iPod can only play MPEG-4 and H.264 video files whereas Creative's Zen can play WMV, XVID MPEG-1/2/4 and DIVX 4 and 5.

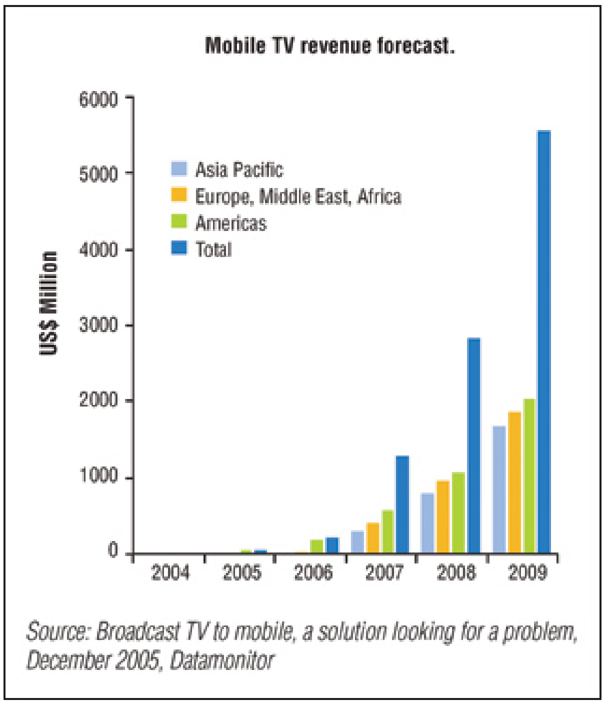

Figure 2.1: Mobile TV revenue forecast

2.3 Transcoding on-demand

Over the last few years the continuous research work in the distributed computing area has met great success and approval by both the academic and the business communities. The physical limitations of the CPU speed was the main reason that led to this direction. Huge companies like Amazon.com, Inc [1] and Google, Inc [8] have developed and deployed web services (Amazon S3 [2], Google App Engine [7]), that allow anyone with internet connection to access and use large amounts of storage and/or computational resources. These services have a front-end which can be used to initiate multiple processes and get the results back. The idea is to have a large number of hosts available transparent to the user which using all different kinds of techniques and fault tolerance strategies perform the requested task. This is a proof of concept that these kinds of distributed environments have become very mature and can be used for different purposes. The same idea can be used for transcoding on-demand.

In a transcoding on-demand scheme, a client would send its exact requirements with each initial request and a transcoding process would start streaming the requested content with specifications that match exactly the ones of the client. With the use of this scheme we can be absolutely sure that the client will get the content in the most appropriate format. The great disadvantage though is that a transcoding process is required for every stream that is served by the application. Hence, this would be a huge overhead due to the fact that transcoding is very CPU intensive process.

A sample architecture for the video application that is powered by a transcoding on-demand scheme can be seen in Fig.2.2. A centralized manager can be used to have an overall view of the application and as the first contact point of the clients. A farm of hosts will be responsible for the transcoding procedure, and the content is stored in hosts that are accessible by them. The numbers in the figure illustrate the sequence of the events. The client contacts the manager which assigns a transcoder for the requested stream. The transcoder fetches the content from the video servers, transcodes it and streams it back to the client.

2.4 Architectures

A fully detailed architecture of a video application can be found here [23]. A more abstract sample architecture can be seen in Fig.2.3. The content is stored in the contribution network and the clients can access it via a distribution network. Hence, the contribution network can be seen as a repository of video content with equipment that can be used to stream the content upon request. The distribution network consists of the infrastructure that connects the client with the servers and allow the video and audio streams to reach the client.

In an internet video application, the contribution network is the website offering the service, the distribution network is the internet and clients are usually PCs connected to the internet over a broadband line. The IPTV service is similar with the distribution

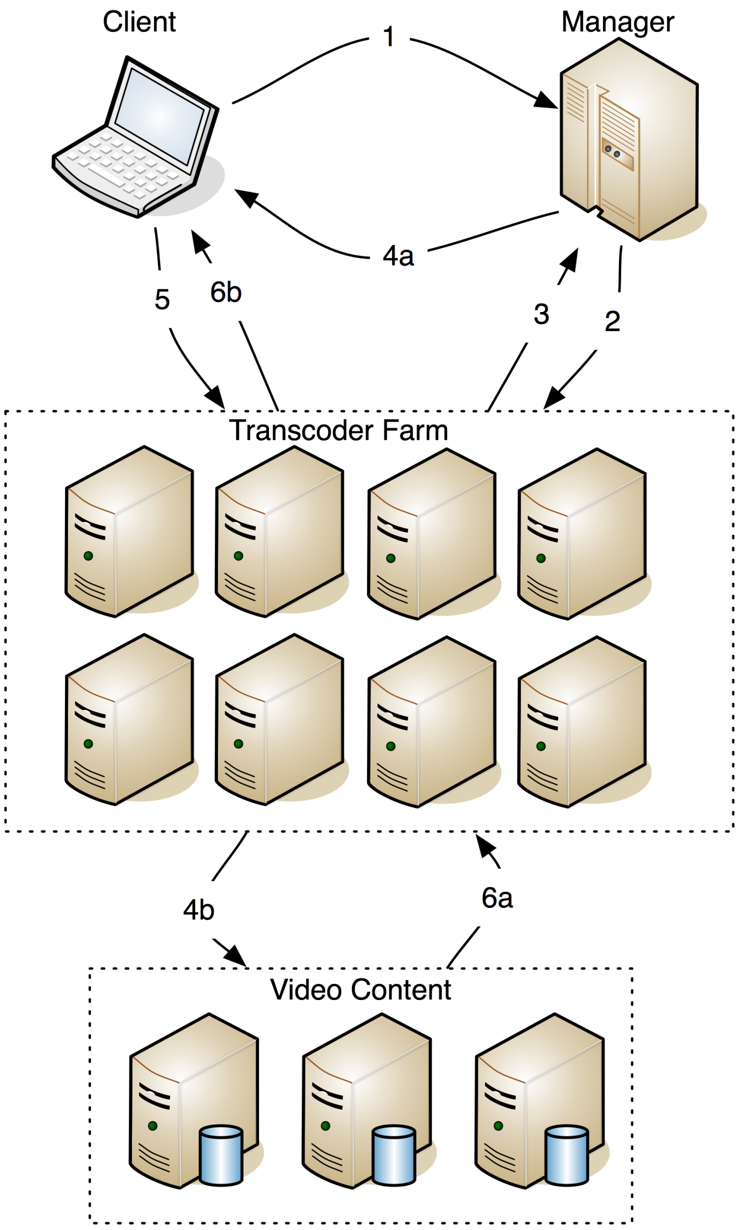

Figure 2.2: Transcoding On-Demand Architecture

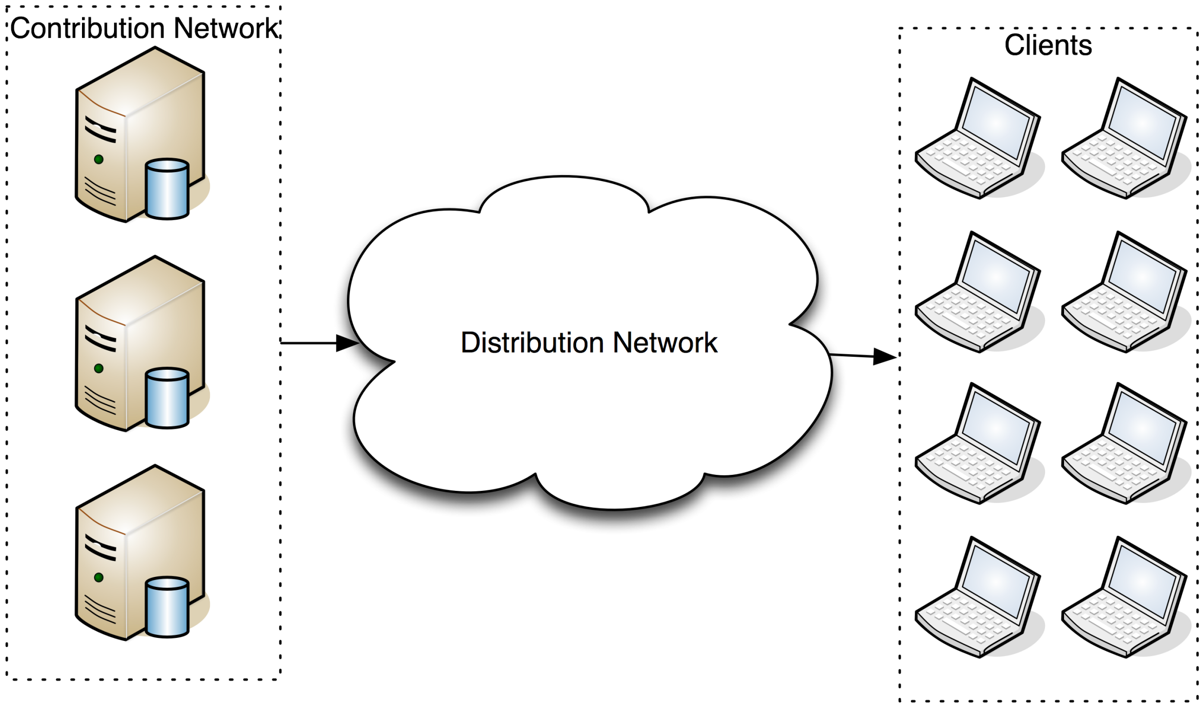

Figure 2.3: Simplified Video Application Architecture

and contribution networks being very stable and with almost no restrictions. These kinds of environment provide a lot of flexibility and can be highly efficient mainly due to the capacity of the network bandwidth, the rich capabilities of the client (CPU speed, memory, power supply) and the relatively low cost of transmission.

However, in a different environment where the distribution network is not as stable as on the above examples and the client has serious restrictions, a video application has to take into account some serious limitations in order to be successful. A video application used by mobile clients is a perfect example for such an environment.

Five different architectures for video transcoding and multimedia streaming are discussed in the next sections.

2.4.1 Off-line transcoding

All the content is pre-transcoded using different options each time and stored locally on the servers. The contribution and distribution network are combined together. Therefore, according to the client's request the appropriate format will be chosen to be served. However, this solution requires a lot of storage space as the same content will be stored multiple times, and more importantly, it is almost certain that there will still be clients that will not be compatible or clients that will not be able to take advantage of their full capabilities because the appropriate video format would not be amongst the available versions. In a real life scenario, a mobile carrier has almost no control on the devices that the subscribers use to connect to the network and as a result this heterogeneity makes it impossible to pre-transcode the content. A definition of standards as seen in [46], can been seen as a recommendation for the video specifications that can be used for mobile video applications in order to restrict the heterogeneity caused by different clients. This scenario also suffers from a single point of failure due to the fact that if a server goes offline all the versions that are stored there, become unavailable.

2.4.2 Server-side transcoding

The content is stored on the servers and when a client makes a request, the content is transcoded and streamed on the fly by the same host that stores it. As with the previous scenario, the contribution and distribution network are combined. This architecture would solve the client heterogeneity problem but a huge amount of resources is required in order to guarantee high quality and performance because each host would only be capable of performing a specific number of concurrent transcoding jobs. Moreover, a situation where some hosts will be idle whereas other won't be able to respond to the incoming requests is highly probable.

2.4.3 Client-side transcoding

This solution is similar to the previous one in terms of the architecture with the difference that instead of doing the transcoding on the servers, it is done on the clients. The content is stored once in a format that will be appropriate for network streaming and when a client receives the stream, it transcodes it on the fly according to its requirements. This is also a good solution for the heterogeneity problem but all the complexity is moved to the client side and therefore this scenario can not be used in certain environments such as a mobile video application. The clients' poor performance as discussed in Section 2.2, don't allow the execution of intense jobs like transcoding. Moreover, the content will have to be streamed using full bit-rate, possibly over a restrictive in terms of bandwidth mobile network.

2.4.4 Transcoding in the Network with fixed resources

As explained before transcoding from one multimedia format to another can be a very intensive procedure in terms of CPU utilization and memory usage. Hence, according to the application size a great number of dedicated hosts is needed to respond to the incoming transcoding requests [39, 44]. The number of the required transcoders might be calculated taking into account the number of requests per time slot, the type of content and the overall QoS that needs to be achieved.

A proper setup of a fixed transcoder farm would allow the video application to be very efficient and stable. The only thing that is really required is to be sure that there are enough transcoders to cope with the incoming requests at all times. There are no issues as far as scalability is concerned because more transcoders can be added if needed without any significant problem.

However, the static nature of a fixed resources for transcoding on-demand video application can lead to a huge waste of resources due to the fact that most of the time the vast majority of the dedicated transcoders will be staying idle waiting for a request. Furthermore, the management and the maintenance of a system like that can add a huge overhead. The ideal solution would be to have exactly the same number of transcoders as the number of the clients that intend to use the application at each given time; due to the nature of the application though, such a solution is impossible. As a result, switching from a completely fixed architecture as the one described above, to one that could somehow gather only the available resources that are needed at any given moment will keep the benefits that the transcoding on-demand scheme can provide without putting the huge overhead that the large number of idle hosts introduce.

2.4.5 Transcoding in the Network with opportunistic resources

The constant rising interest in distributed computing has resulted to the development of distributed processing systems like Entropia [22] and Condor [4] which are used for resource management. Another example is BOINC [21], a platform for public-resource distributed computing which forms the base for different projects such as SETI [20], Predictor [15], Folding [45] and Climateprediction.net [26]. The main characteristic of all of the above systems is that the resources they manage are not fixed and each host which participates in the system can be either available or unavailable according to certain criteria and policies. These systems are designed to use resources given by a large number of computing units but only when these resources are not used for their primary purpose. A university student lab can be a fine example of such a system: the desktop grid system uses all resources provided by the idle computers until a student logs in to any of them. In that case, the resource is not available to the desktop grid anymore until it becomes idle again. A desktop grid like that described above can be identified as a grid using opportunistic resources.

These systems have a job management mechanism via which new jobs can be submitted and results can be retrieved. Moreover, they include resource allocation and powerful fault tolerance mechanisms. The use of opportunistic resources for any kind of computation can greatly increase the throughput of a given system because the idle time of the resources is eliminated. However, all these systems are best effort systems which means that they can be an excellent choice for running batch processes but might not be appropriate for an application that is highly concerned about QoS such as a video application [31].

The opportunistic nature of the resources imposes a serious issue: a host is being used for transcoding and streaming some content that was requested from a client. During this period the host becomes occupied by its owner and therefore the transcoding job has to be stopped on this specific host. In order to continue the service, the transcoding job has to be resumed by using other available resources with the minimum possible impact of the QoE. Authors of [40] and [34] suggest mechanisms that can be used in order to migrate transcoding and streaming jobs between different transcoders while the client is changing networks.

2.5 Protocols

In a video application the selection of the tools and protocols that will be used for video streaming is highly crucial because different choices serve different needs and are suitable for specific environments. RTSP and RTP are the main protocols used in video applications.

2.5.1 RTSP

RTSP or Real-Time Streaming Protocol is a protocol developed by IETF and is described in RFC2326 [43]. Although the protocol is similar to HTTP in terms of syntax and operation, it is not stateless as HTTP is. It is used by video clients to initialize and manage a video stream session with a video server. RTSP supports different kinds of commands which are sent from the client to the server and vice versa. Some of the most important RTSP commands include OPTIONS, DESCRIBE, SETUP, PLAY, TEARDOWN and REDIRECT. RTSP is not used to deliver video content but only to control various streams between video servers and clients. The streams that are controlled by RTSP usually use RTP as their transport protocol. Here is a short description of the above commands.

- OPTIONS: It is sent by the client to the server in order to determine the RTSP commands that are available from the server.
- DESCRIBE: It is sent by the client to the server to retrieve information about the requested resource. The answer is in the form of an SDP document [28] and contains all the required information.
- SETUP: It is sent by the client to the server for a URI and is used to specify the transport mechanism to be used for the streamed media.
- PLAY: It is sent by the client to the server in order to begin the transmission of the video content with the mechanism that was agreed in the SETUP command.
- PAUSE: It is sent by the client to the server to pause a continuous stream with the intent to resume in the immediate future by issuing a PLAY command.
- TEARDOWN: It is sent by the client to the server to terminate the current session.
- REDIRECT: It is sent by the server to the client with a URL information and it indicates that the client should continue its current session with the video server specified in the URL.

Every command is acknowledged with a status code exactly like is done with HTTP. The most status common code is *200* which represents a successful command. Other status codes include: *302* which is accompanied with a URL that indicates the location that a requested resource has been moved to, *404* as a response to an unknown resource and *405* to acknowledge unidentified or unsupported RTSP commands.

A typical RTSP session negotiation can be seen in Fig. 2.4. The client initiates the negotiation by issuing an OPTIONS command, getting back a list with the RTSP commands that are available on the server. In step 2 a successful DESCRIBE command results in a detailed description of the video resource in the form of an SDP file. In step 3, one or more SETUP commands are issued depending on the number of different streams that the video content might have e.g. audio and video. When the server replies to a SETUP request it provides a session number with which the client will be identified from now on. At this point, the client and the server have all the required information about each other. Therefore, a client can issue a PLAY command in order to tell the server to start sending the content using the information described in the above steps. At any point during playback, the client can pause and resume the stream using the PAUSE and PLAY commands respectively, as well as stop the stream and end the session with a TEARDOWN command as seen in step 6.

2.5.2 RTP

RTP or Real-Time Transport Protocol [42] provides end-to-end delivery services for audio and video data. Those services include payload type identification, sequence numbering, time stamping and delivery monitoring. RTP runs on top of UDP to make use of its multiplexing and checksum services.

RTP and RTSP combined together for the transport and control stream respectively can provide the mechanisms to deploy a fully functional, very efficient and detailed multimedia video service application over an IP network.

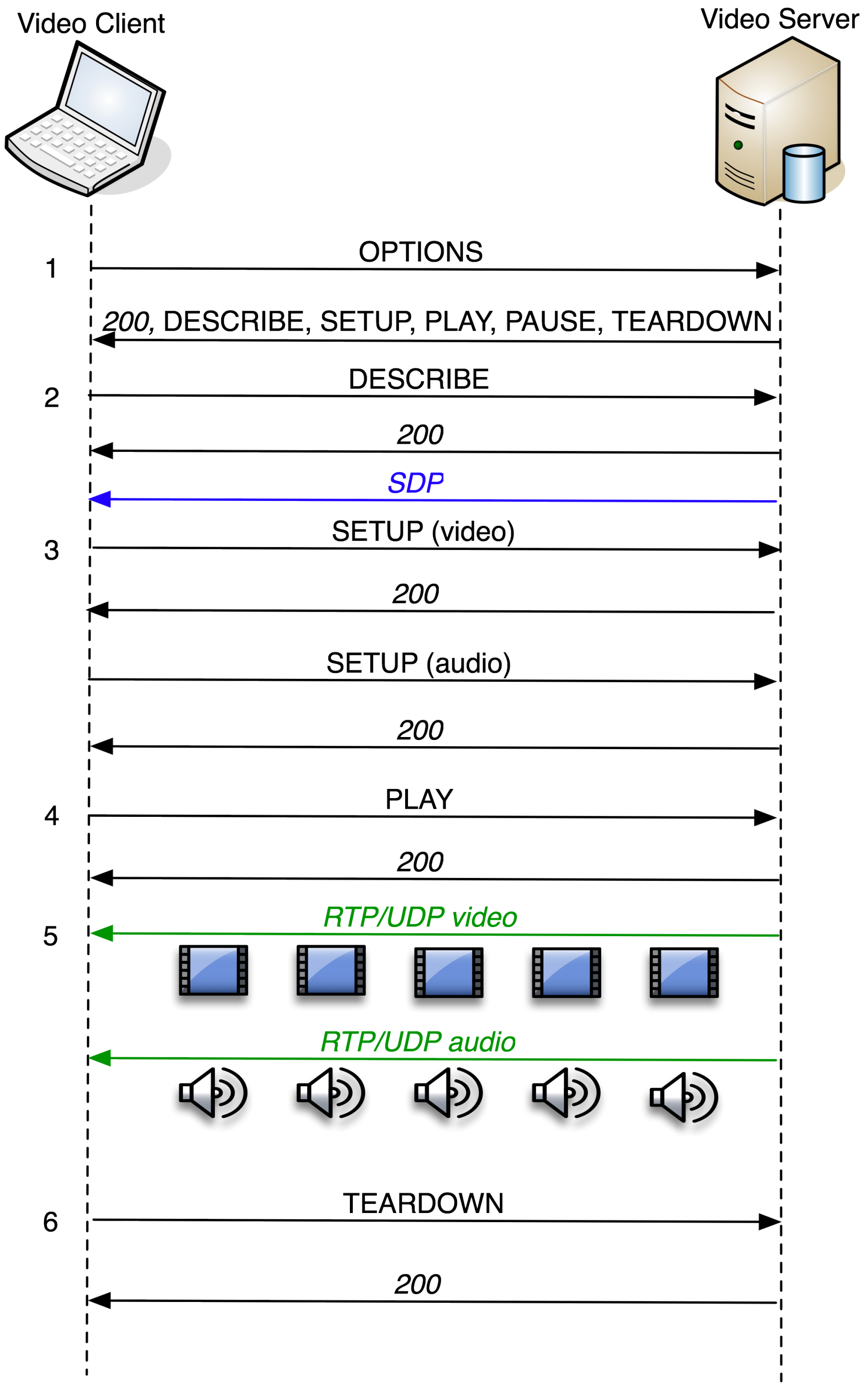

Figure 2.4: Typical RTSP Session Negotiation

2.6 Applications

2.6.1 VLC

VLC is a free, open-source, cross-platform media player with advanced streaming capabilities over IPv4 or IPv6 high bandwidth networks. It supports multiple video formats and streaming protocols and its current stable version is 0.8.6. It is mainly based on the *libavcodec* open-source library and the *ffmpeg project* [6]. VLC provides different interfaces (command line, web, GUI, telnet) as well as APIs for popular languages like C, C++, C#, Python [16], and Java [10] which make it highly flexible when used for any kind of playback or streaming.

RTSP is supported by VLC by using the *liveMedia* library [12] which is part of the *live555* project [11]. VLC with RTSP support can be used for online or offline transcoding, video on demand services and unicast or multicast streaming. In addition to that, the various interfaces and APIs that VLC provides, allow it to be integrated in a more complicated environment and can be the basis of a more advanced video application.

VLC is implemented in a way that permits the creation of *output chains*. VLC reads any kind of video input (file, stream, DVD etc) and by defining one or multiple output chains the content can be streamed of used for playback. One chain can be *attached* to another chain and as a result creating a more complicated process. An example can be seen in Fig. 2.5. A VLC instance can be set up to read a local file, transcode it and then stream it to a multicast address by using two chains. Chain 1 will read the file and transcode it, and chain 2 will read the output of chain 1 (transcoded file) and output the stream to the specified address. The whole procedure happens on the fly in the similar way UNIX pipes work.

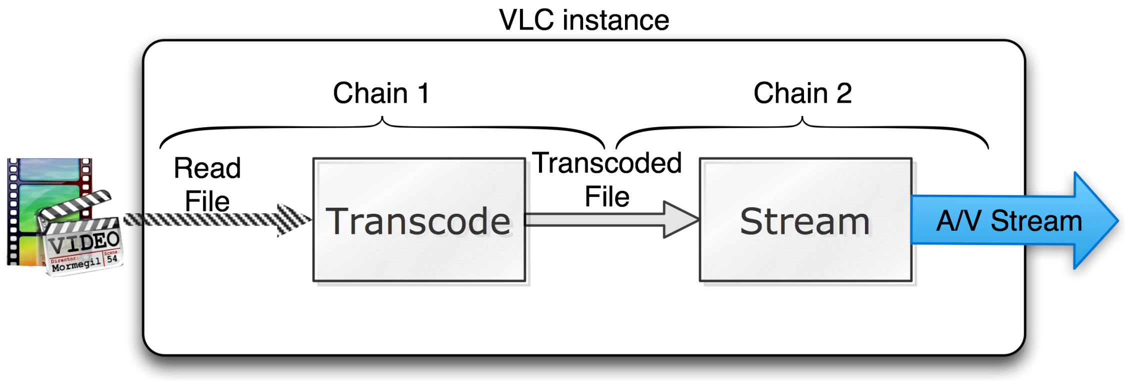

Figure 2.5: VLC Example

2.7 Existing Solutions

In terms of multimedia session management and migration, the authors of [30] provide the design and the evaluation of a platform that enables a multimedia session to migrate or to join another existing one, which can be very useful in order to optimize the performance of an existing multimedia application.

Content adaptation for mobile devices has originally been defined by the World Wide Web Consortium (W3C) as "a process of selection, generation or modification that produces one or more perceivable units in response to a requested uniform resource" [18]. Numerous different companies, provide specialized and personalized services according to the client's request. Huge websites like YouTube.com and Facebook.com deliver different content for mobile clients in an effort to match the client's capabilities and therefore improve the quality of the delivered service. In [47] the authors discuss different ways of delivering optimized content to mobile handheld devices. The content varies from simple web pages to multimedia files. Another implementation of a context aware multimedia application is proposed in [41] which is called RMob. "The RMob system uses and extends W3C standards to implement profile-based transcoding policies." [41]. RMob is a web-based distributed application that is powered by a transcoding on-demand scheme and provides personalized content depending on the client's capabilities.

Finally, the idea of using a grid to support transcoding on-demand applications has been explored by other researchers. The authors of [35] suggest an architecture which can be used for this purpose.

2.8 Video Metrics, QoE

The most important parameter that every multimedia application takes into account is QoE (Quality of Experience). QoE is a subjective measure that is related with the overall experience of the end point client. It is related with the objective QoS but they differ. QoS is used to describe infrastructure and overall system performance and productivity whereas QoE is used to illustrate the overall result of the service regardless the performance of the system that produces this result. Furthermore, multimedia applications widely suffer from different typical network effects such as latency, jitter and packet loss. Even a small amount of any of the above may have a great negative influence on the QoE, leaving QoS untouched most of the times. In terms of our application two metrics are critical and are bound to affect the QoE: a) the latency between client selecting a stream and the actual beginning of playback and b) the elimination of any kind of pauses or break during playback.

Chapter 3

Opportunistic Resources Using Condor

3.1 Introduction to Condor

Condor is a distributed batch system that was designed to provide an elegant way to take advantage of computing resources when they are not used [32, 25, 4]. Condor maintains a pool of hosts and is able to assign jobs to the hosts that are available. Jobs can be submitted to the Condor system and they will be kept in a queue until the appropriate resources are available. All the hosts in a condor pool are meant to have a primary purpose apart from running jobs submitted by condor therefore condor will always give priority to the host's owner. Condor implements the following mechanisms:

- Determine whether a host is available or not. If the host is not used by its owner, it becomes part of the available hosts pool and therefore can be used for running condor jobs. Condor provides a policy based mechanism which administrators can use in order to determine the availability of the host. For example a computer might be considered idle if the CPU load is less than 0.3% or the keyboard and mouse have been idle for 10 minutes.

- Provide “fairness” to the queued jobs. This means that a really heavy job in terms of resources will not take over the system and smaller jobs will be given the opportunity to run. An *up-down* algorithm is used to provide priorities to various jobs. Therefore, a job that has been waiting in the queue for too long will periodically increase its priority and will eventually be executed.

- Remote execution is supported. Jobs written in different languages like C,FORTRAN or JAVA can be executed from a remote host. Condor will manage to transfer the input and output files.

- Condor will automatically stop the execution of a running job if the host that the job is running on gets occupied by its owner. As explained above the definition of an occupied host can be described by custom defined policies.

- A checkpoint mechanism gives the ability to resume a suspended unfinished job by migrating it from the recently busy host to an idle one. This whole procedure is completely transparent to the host’s owner.

3.2 Condor Architecture

3.2.1 ClassAds

Condor’s way to determine whether a machine is available or not and whether a job is possible to run on this specific host or not is by using a schema-free resource allocation language called *ClassAds* [38, 36, 37]. The ClassAds language provides a very flexible and expressive way of matching resource requests with resource offers [25]. Each job has a list of properties called *JobAds* that represent the job and each host in the condor pool has a list of properties called *MachineAds* that describe the host. When a job is submitted, condor uses its matchmaking mechanism to determine the appropriate host for this job.

The matchmaking process involves comparison of the JobAds and different MachineAds. When a match is found, the submitted job is being scheduled and eventually executed by the hosts that matched the job's requirements.

3.2.2 Daemons

There are five different daemons that complete the condor system: the *Collector*, the *Master*, the *Negotiator*, the *Schedd* and the *Startd* [25, 4]. In any condor pool there must be one machine which will be the Central Manager (CM) and which runs the *Negotiator* and the *Collector* daemons. All the hosts that are part of the condor pool and are able to execute jobs will run the *Startd* daemon, which is responsible for advertising the machine's Ads and therefore determine if the machine is busy or idle. It is also responsible for starting, monitoring and terminating a job. *Schedd* manages the incoming jobs by maintaining a queue and seeking available resources for them. The *Shadow* daemon communicates with the *Startd* running on the host that is about to execute a new job and makes sure that all the executables, input and output files are transfered before and after the job is executed.

3.2.3 Condor Policies

Condor needs to keep track of all the activities done in the host so that it can decide whether the host is available or not. The host is only available to run condor jobs when it remains idle. Condor has a very efficient and flexible way to determine and identify any changes on the host. It allows the administrator of the system to define policies on which condor will base its decisions. For example an idle host can be one that its keyboard and/or mouse haven't been used for 10 minutes or the CPU load has been less than 5%. Thus, a condor installation using the specified policies determines when a host is available, used, suspended or unavailable. The policies are highly customizable. However condor's default installation comes with all the necessary policies pre-defined.

3.2.4 Invocation Mechanism

The way the condor daemons participate in the submission of a job can be seen in Fig. 3.1. In step 1 the *Schedd* daemon accept the submission of a job. The *Collector* and the *Negotiator* daemon determine if the specific host is able to run the submitted job and if it is and the host is available, *Startd* communicates with *Schedd* acknowledging that it is ready to execute a specific job. As a result, *Schedd* passes all the necessary information to the *Shadow* daemon, which makes sure that all the prerequisites all being met before *Startd* launches the executable.

3.3 Job Management

Once a job has been submitted to condor, it uses its internal mechanisms to manage the various steps of the job. Firstly, the job is placed in the queue until its priority is high enough and resources that are able to run this job are available. When the job is matched for execution it starts running on a condor host until it finishes or until the host becomes unavailable. If it manages to finish, condor notifies the job's owner and reports the results, else if it stops executing due to the resource becoming unavailable (and therefore available to the host owner), condor tries to reallocate the job if that is possible otherwise it is resumed once the resource is available again.

3.4 Condor Example

The Condor *Collector* and *Negotiator* daemons have information about the condor hosts in the form of a MachineAds list. A part of this list can be seen in Table 3.1. The

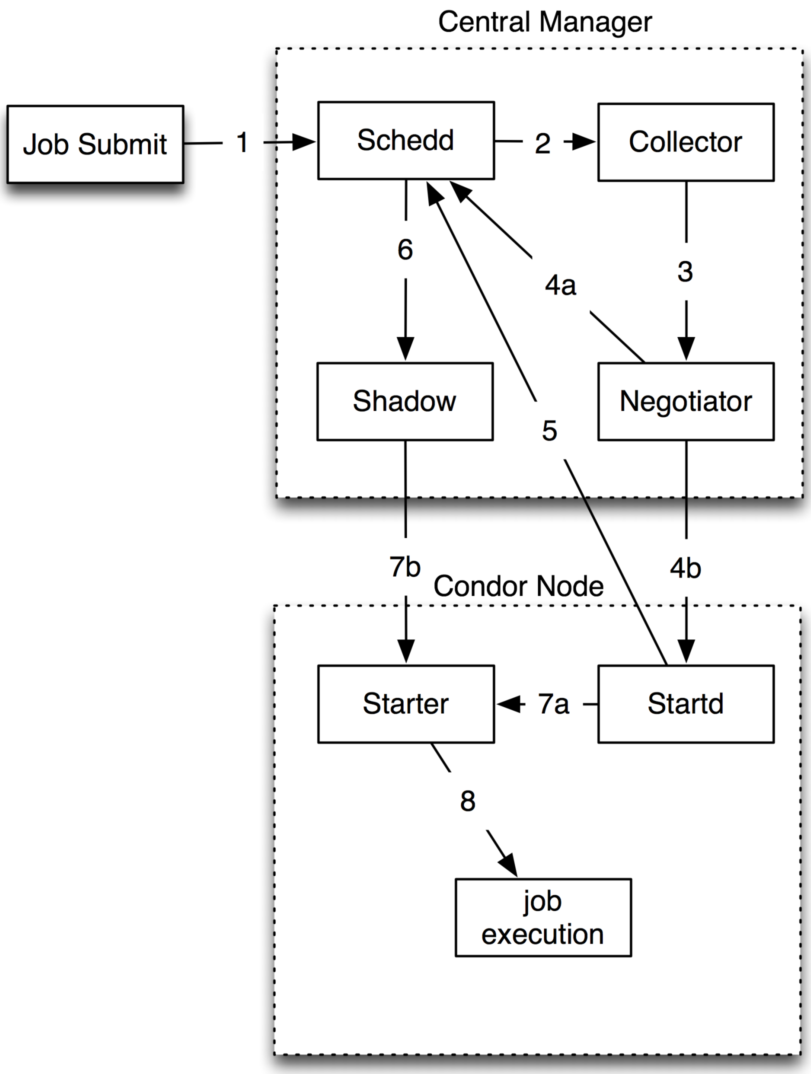

Figure 3.1: Condor Invocation Mechanism

MachineAds list illustrates the full state that the machine is at the moment as well as its specifications (CPU, available memory, disk space). When a new job is submitted to condor, it is done in the form of a JobAd list as seen in Table 3.2. Hence, the *Schedd* daemon which accepts the job, passes the information (JobAds) to the *Collector* and then the *Negotiator* tries to find a host that will be able to run this job by comparing the JobAds with the MachineAds of every host. In the given example the job has only one requirement and that is the architecture and the operating system. The job can only be run in a condor host that has an Intel processor and runs Linux or Solaris2.6, or a SPARC processor and runs Solaris2.8. Thus, the host that is represented in Table 3.1 is able to run this job because it matches the job's requirements.

Condor Host MachineAds
Machine = "cagcotest03.cs.tcd.ie"
PublicNetworkIpAddr = "<134.226.53.173:1098>"
CollectorHostString = "cagcotest04.cs.tcd.ie"
VirtualMemory = 626476
TotalDisk = 3073272
Disk = 3073272
CondorLoadAvg = 0.000000
LoadAvg = 0.010000
KeyboardIdle = 77729
ConsoleIdle = 1181463
Memory = 377
Cpus = 1
Arch = "INTEL"
OpSys = "LINUX"
Activity = "Idle"

Table 3.1: Condor Host MachineAds

3.5 Fault Tolerance

Condor is a fully fault tolerant system. The checkpoint and migration built-in mechanism allows any job to resume execution after a failure. Moreover, Condor allows its Central

Manager to be replicated so as to eliminate any single points of failure.

3.6 Problem Solvers

Condor is designed in a way the permits the use of it as a low level mechanism for running jobs in a distributed environment. A higher level structure built on top of condor is known as a *problem solver* [25]. A problem solver uses the mechanisms that condor provides to submit and monitor jobs. Condor handles all the details that are involved when a job is executing in a distributed environment, such as failure. In addition to that, the problem solver can itself be considered reliable if it runs as a condor job. Therefore, a problem solver can provide an architecture with application specific details implemented, while at the same time taking advantage of the major benefits that condor provides. There are two well known problem solvers available: *Master-Worker* (MW) and the *directed acyclic graph manager* (DAGMan) [25, 4]. Both of them provide a different architecture that would suit a specific set of applications and problems. Any other problem solver can be built on top of condor using the available interfaces that condor provides.

Condor JobAds
universe = vanilla
Executable = povray.$$(OpSys).$$(Arch)
Log = povray.log
Output = povray.out.$(Process)
Error = povray.err.$(Process)
Requirements = (Arch == “INTEL” && OpSys == “LINUX”) || /
(Arch == “INTEL” && OpSys ==“SOLARIS26”) || /
(Arch == “SUN4u” && OpSys == “SOLARIS28”)
Arguments = +W1024 +H768 +Iimage1.pov
Queue

Table 3.2: Sample Condor JobAds

3.7 Interfaces

Condor provides a wide variety of powerful command line tools which can be used to submit, monitor and manipulate jobs and condor hosts. Other condor APIs include DRMAA, Condor GAHP and a condor perl module. DRMAA is a GGF [5] standardized job-submission API which is easy to use but lacks some important features, including two-phase commit support, fault tolerance and transaction support. Condor GAHP is a low-level protocol based on ASCII messages through stdin and stdout. Most importantly condor has a SOAP interface which allows condor to become a service. Different hosts can submit and monitor condor jobs doing RPCs using XML over HTTP or HTTPS. Any SOAP toolkit can be used with any language that supports SOAP.

Chapter 4

Multimedia Processing Using Condor

4.1 Description of Video Condor

The skeleton of our mobile video application called Video Condor is based on the one seen in Fig. 2.3. Our contribution network consists of a video-on-demand (VoD) server and a farm of transcoders, all provided by condor. Thus, the transcoding jobs will be running on hosts that are part of a condor pool and managed by the central manager of condor. The distribution network can be any IP network. Our clients are mobile devices with limited capabilities that make a request for specific content and receive a video and an audio stream of the requested content that matches the playback characteristics and capabilities of the device.

4.2 Requirements

During our application's design we imposed two requirements: the use of RTSP as the control protocol and the use of a simple client that will be as less modified as possible.

The choice of RTSP was made due to the fact that it is a rich widely known open protocol that is supported by a large number of different clients and applications. The client must not be modified and specialized for the application because one of our motives for building Video Condor was to deal with the heterogeneity that all the different kinds of clients would introduce. Thus, the use of a default unmodified client is considered essential. These two very essential decisions had a big impact on our design, which will is covered on the next sections.

4.3 Design

Video Condor uses condor hosts to run transcoding and streaming jobs. It is highly important to have an overall view of the whole system at any given moment. Hence, the use of a manager process that will identify the availability status of the different hosts is essential. The dynamic nature of the available resources provided by condor create the need of a centralized mechanism of accepting and forwarding requests depending on different needs. This is done using an RTSP proxy. The architecture we use is illustrated in Fig.4.1. The main components are a VoD Server, a proxy server, a master server, the video clients and the condor hosts. A client makes a request for some content to the proxy using a standard DESCRIBE RTSP command. The manager using an internal protocol assigns a transcoding job to one of the available transcoders and then notifies the proxy about its selection while the transcoder is preparing to starting the transcoding and streaming. Therefore, the proxy replies to the client's RTSP request with a Code *302*. The proxy's response includes a URL which is the address of the transcoder that was assigned to serve this specific client. As a result, the client is being redirected to the selected by the manager transcoder. Finally, the client using standard RTSP negotiation as explained in 2.5.1, establishes a session with the transcoder and the transcoded content is being served by the condor host over RTP.

All the communication between the client and the proxy or the transcoders is done using standard RTSP commands which meets our initial requirements. The protocol that the manager uses to control and give commands on the transcoders is a simple protocol described in 4.4.

An alternative and probably faster scenario would have been a direct connection between the transcoder and the client right after the manager's choice concerning the host that will run the job. However, this would be impossible by using RTSP and would require the use of a non-standard protocol and therefore a modified client.

4.3.1 VoD Server

The VoD Server is a host that stores all the content that will be available to the clients. The transcoders request the content from the VoD Server which is served as a unicast stream. Therefore, for each client there is one stream from the VoD Server to one of the transcoders. It can be easily duplicated for efficiency and fault tolerance.

4.3.2 Proxy

The proxy server is the first point of contact that the clients have, therefore it must be locatable. For example, its IP address has to be known to the clients. The proxy's responsibility is to redirect a client to the condor host that has been assigned to execute the transcoding job that is required in order to serve the specific client. In our architecture the proxy server is integrated with a condor host manager which is explained below in Section 4.4.

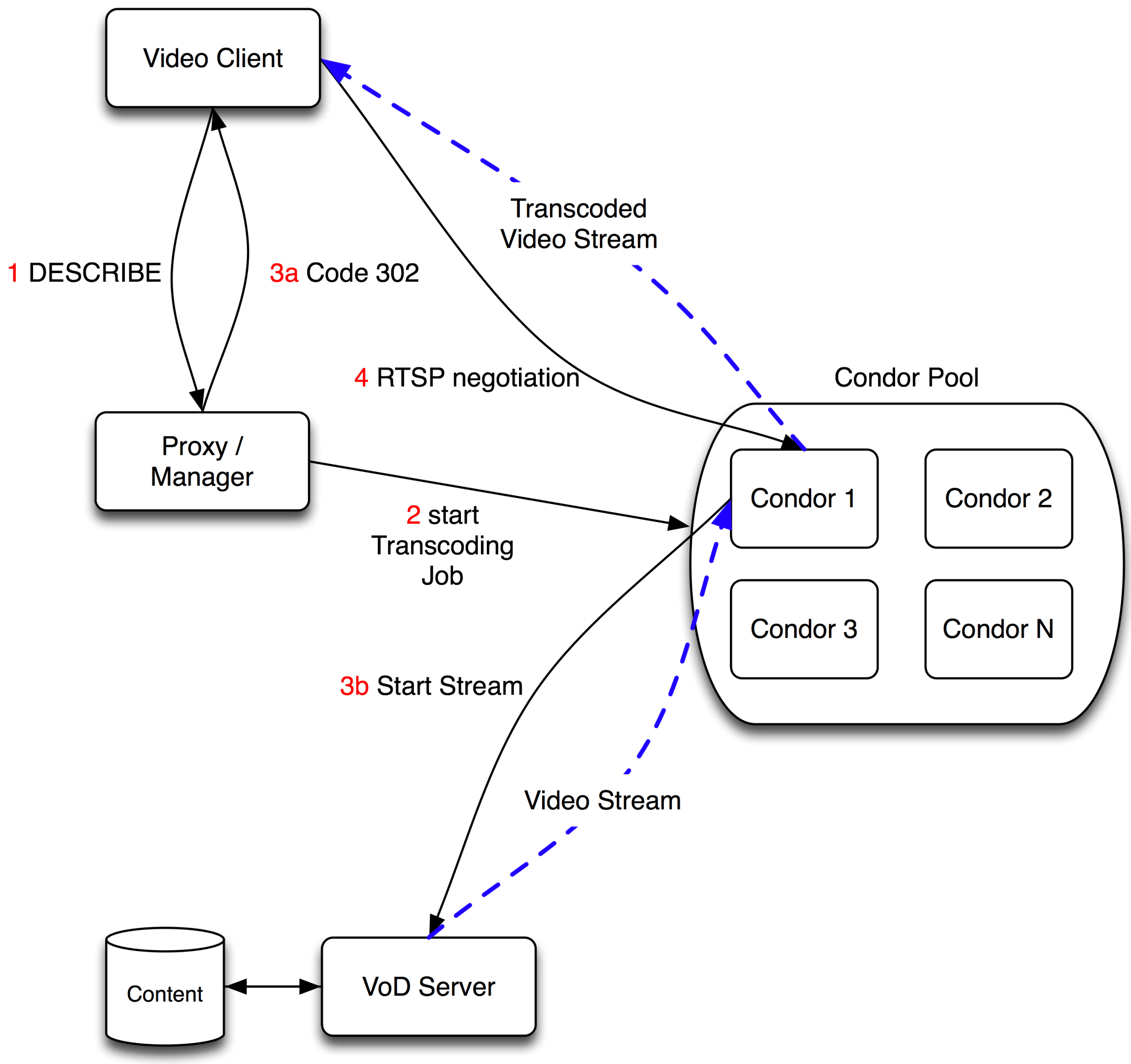

Figure 4.1: Video Condor Architecture

4.3.3 Clients

Any client that supports the RTSP protocol as described in the RFC [43] can be a client in our application.

4.4 Management of Resources

Condor Hosts

The Condor hosts are the resources that are going to be used for transcoding. Due to the opportunistic nature of the resources, its vital to be able to identify the current status of each host. Four different states are used:

- *Unavailable*: The host is either occupied by someone else using it for its primary purpose or the condor job hasn't been executed yet.
- *Available*: The condor job is running and the host is ready to begin a transcoding job.
- *Busy*: The host is occupied running a transcoding job and streaming the content to a client.
- *Suspended*: The host became occupied while it was running a transcoding job and serving a client.

The manager using a polling mechanism identifies the status of the hosts and maintains and updates this information. Unfortunately condor does not have support for event notifications and therefore the only way to get information is by polling.

Host Policy

As explained in Section 3.3, when a condor job is submitted the user has no control concerning the time that this job will start being executed by a condor host. Condor decides when a job is going to start being executed according to the policies defined on its settings. This would be disastrous for a mobile video application because it would add a huge overhead to the start of each stream resulting to long waiting time for the stream to start. Thus, instead of launching a transcoding job right after the proxy receives a request, a *pre-provisioning* policy is used. This policy allows us to significantly reduce the starting time of a transcoding job and therefore maintaining the QoE level high. When the application starts, it automatically maintains a specific number of hosts waiting to accept transcoding jobs. These hosts are idle hosts that run a condor job that does nothing more than waiting for an order, using our internal protocol, to start a transcoding job. A manager is needed in order to have an overall view of the hosts and to be able to assign transcoding jobs according to the incoming requests.

Manager

The manager is responsible for monitoring a number of condor hosts that are ready to run a transcoding job and give the command for a transcoding job to start in any of them. It regularly polls the condor hosts to check their status. When the application begins, a number is specified and the manager submits equal number of jobs to condor and automatically starts polling condor for the job status. Hosts are identified as *Unavailable*. When condor decides to execute the jobs, these running jobs are considered transcoders ready to be used by our platform and therefore hosts are marked as *Available*. Hence, manager's main responsibility is to keep track of all the changes that happen to the transcoders. When a new request is received from a client on the proxy, the manager selects an available host and sends the information that is needed in order to start the transcoding job while the proxy redirects the client to that host. Once a transcoding job is running on a host, this host is considered *Busy* and therefore not available to run more jobs until its done. When a job get suspended due to the host being occupied, the manager is responsible for finding another available host to resume the transcoding while it marks the first host as *Suspended*.

Sequence of Events

The behavior of the system can be seen in a sequence diagram in Fig.4.3. The manager submits the necessary condor jobs and begins polling the hosts to check the status. As a result the manager always has the overall view of the resources regarding their availability. When a client requests a stream from the proxy, the manager chooses one available host and passes the information needed from the host to start the transcoding job. When the transcoder becomes ready to serve a client after receiving an order from the manager, it notifies the manager and the proxy redirects the client to that host. The client uses RTSP to initiate the multimedia session and eventually receive the transcoded stream. The events marked as "Playback Request", "Redirect to Host 1" and "Streaming Done" refer to RTSP commands which can be seen in detail in Fig.4.7. The "Playback Request" is the RTSP session establishment negotiation, the "Redirect to Host 1" is the proxy's reply with Code *302* and the URL of the transcoder whereas the "Streaming Done" refers to the TEARDOWN command issued at the end of the stream. All the other events on Fig.4.3 correspond to the internal protocol used between the manager and the hosts to identify the different events that occur.

Job Management Protocol

The transcoding job is completely transparent to condor. Condor only knows about the idle job that was submitted to it, and its waiting for a transcoding order. A diagram that illustrates the condor job submitted can be seen in Fig.4.2. When the condor job starts running it creates a TCP socket which will be used by the manager to send its orders. This socket is marked as "A" in Fig.4.2. When the manager sends an order to start the transcoding the process running as a condor job contacts the VoD server using socket "B" and creates a socket "C" which will be used by the client to request the content after it will be redirected by the proxy using the RTSP protocol. After a client's request the transcoding process starts transcoding and streaming using a UDP socket marked as "D".

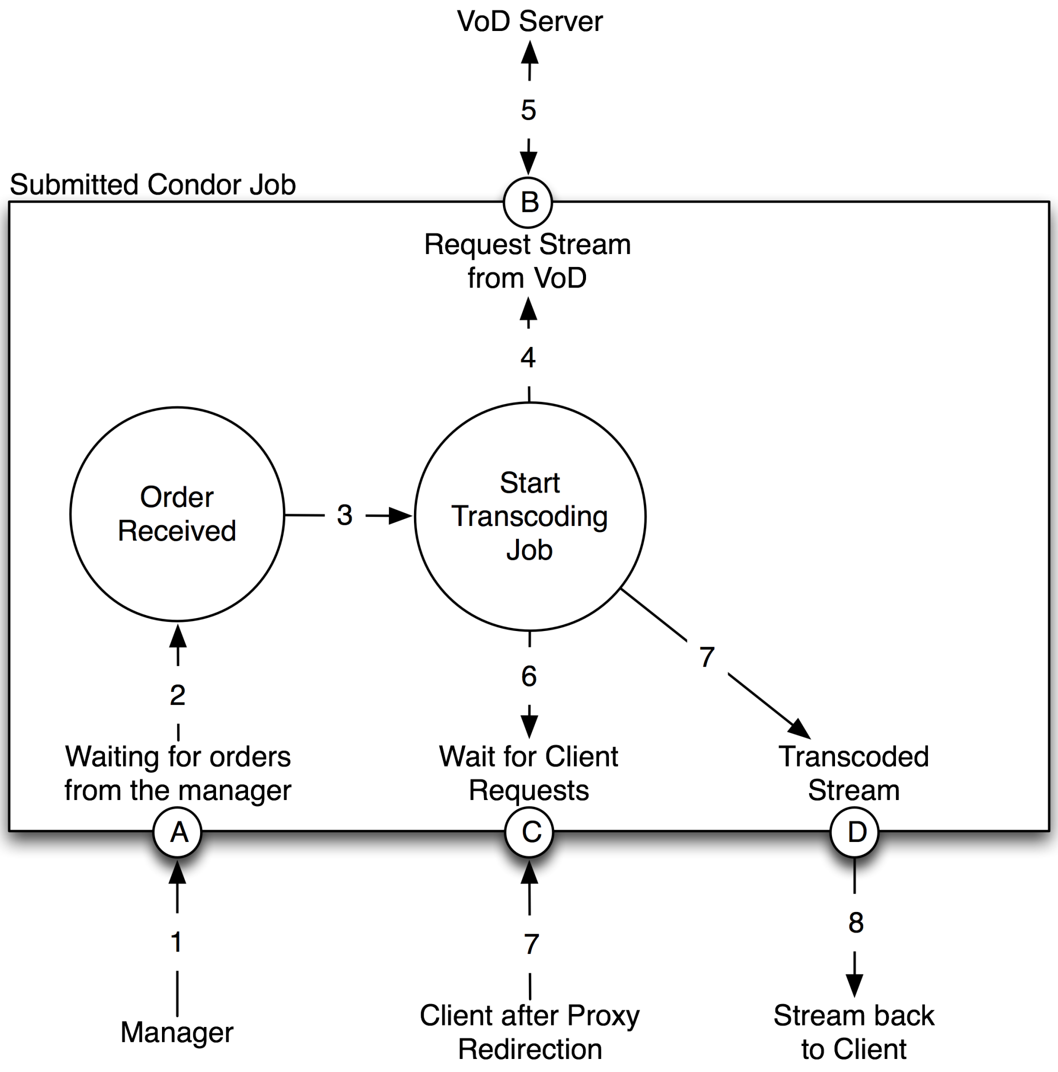

Figure 4.2: Condor Job

The status of a transcoding job can not be seen using the condor API and the JobAds.

Hence, in order to start a transcoding job and to identify when it ends, a simple protocol is used between the manager and the condor hosts. This protocol consists of pre-defined numbers and strings that are parsed from both sides and correspond to specific actions and status updates. This protocol is illustrated in Fig. 4.3.

4.5 Implementation

4.5.1 Proxy/Manager

The manager is designed and implemented to behave as a *problem solver*, as explained in Section 3.6. It is an application built on top of Condor and by using the Condor SOAP API it triggers various events upon certain situations while it maintains a view of the running processes. The status of the running jobs is acquired by polling Condor for the JobAds and checking the property *JobStatus*.

The Manager maintains a job queue which is a list of all the incoming requests and a list of the available hosts that can be used for transcoding. New jobs are added to the job queue by the proxy and for each job in the queue, the manager finds an available host to execute it as seen in Fig. 4.4. As long as there is a request in the Job Queue, the manager is responsible for finding available resources to execute this job. By using the pre-provisioning scheme as explained before we can be sure that there will always be available hosts for running transcoding jobs. When the queue is empty, the manager is only responsible for updating the status of the hosts and rearranging the pool size.

4.5.2 Transcoders/Clients

VLC is used for the RTSP control stream as well as for receiving and rendering the RTP streams. The main problem though is that VLC only supports basic RTSP commands like

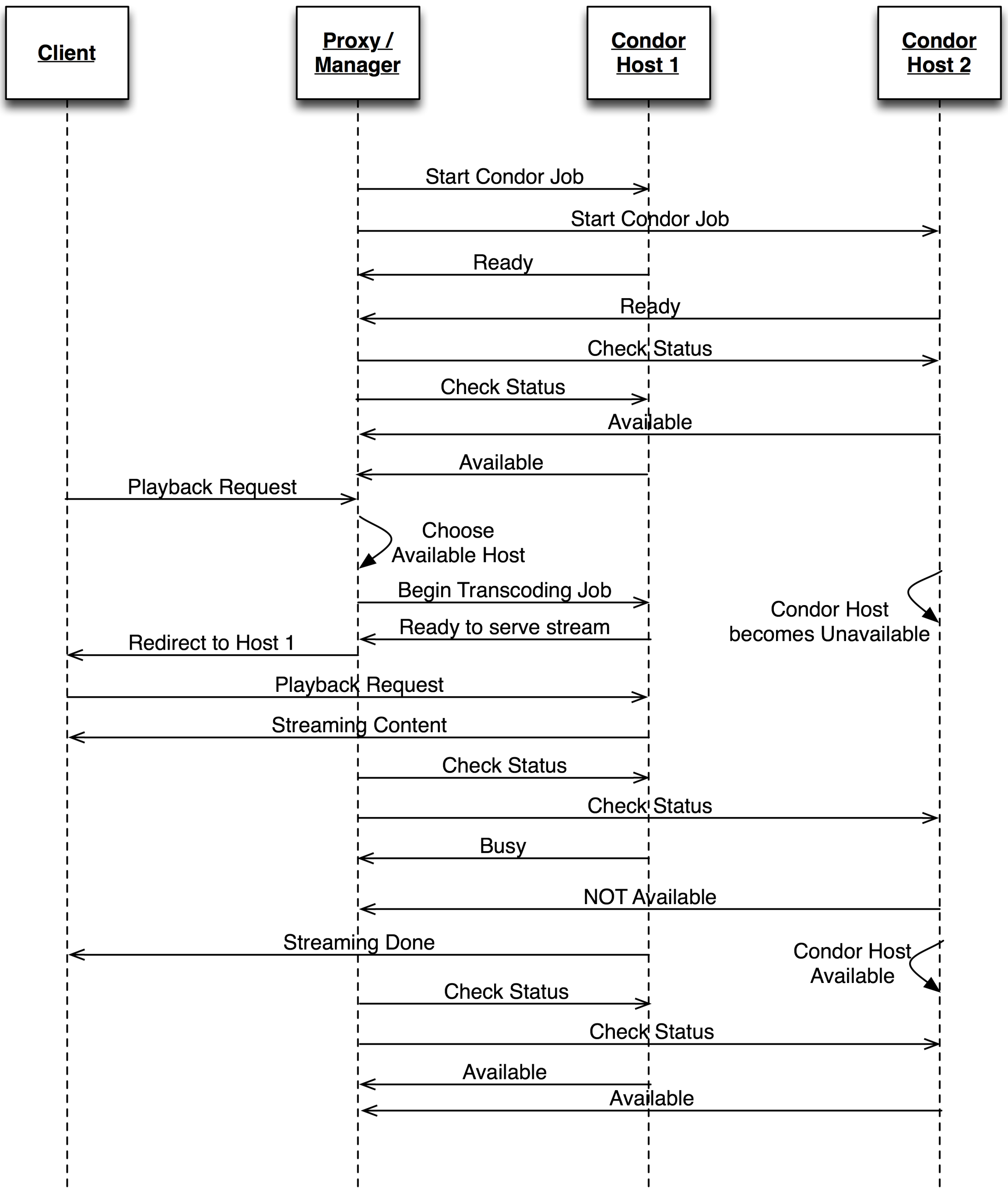

Figure 4.3: Video Condor Sequence Diagram

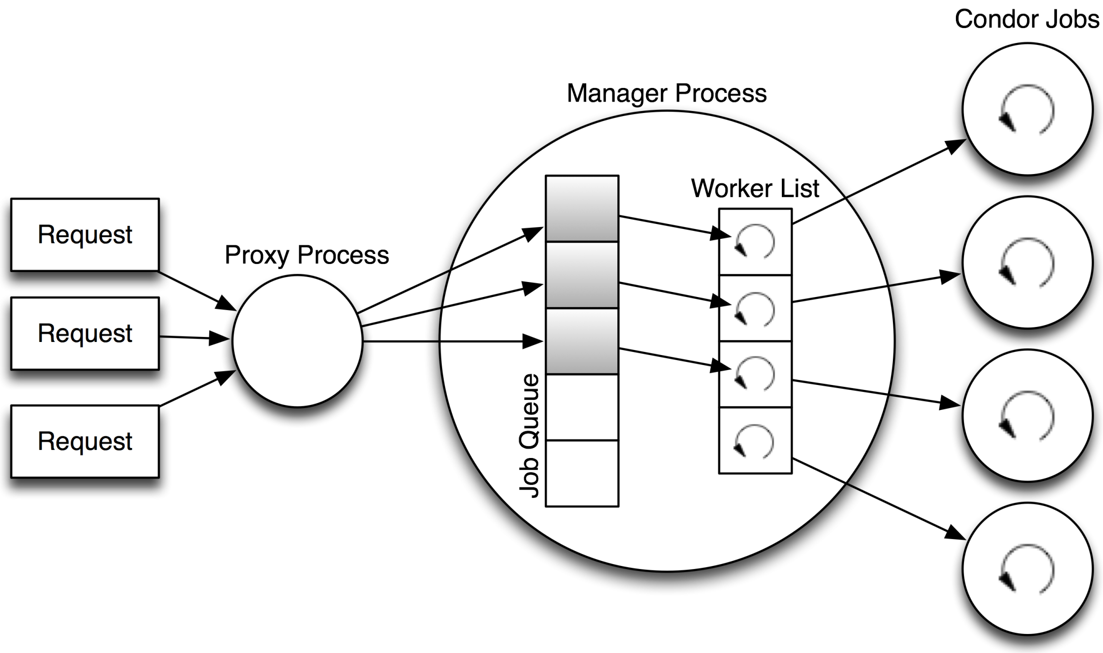

Figure 4.4: Proxy and Manager

OPTIONS, DESCRIBE, SETUP, PLAY, PAUSE and TEARDOWN. Thus, we needed to extend its RTSP functionality to support REDIRECT and different response Codes as described in Section 2.5.1. The REDIRECT command support is highly crucial because its part of the standard use of RTSP protocol. REDIRECT is issued by the streamer and allows an already connected to this streamer client, to resume its current session from another source. This was achieved by using the Python VLC bindings to create a wrapper around a VLC instance. VLC communicates with the local wrapper and the wrapper decides what needs to be done according to the situation and to what the RTSP RFC defines. Therefore, if the wrapper receives an RTSP command or response that is supported by VLC, it just forwards it to the local VLC instance as seen in Fig.4.5. On the other hand, if an unsupported by VLC RTSP command is received, the wrapper perform several actions that will have the desired result, while its external behavior as described by the RTSP RFC as seen in Fig.4.6.

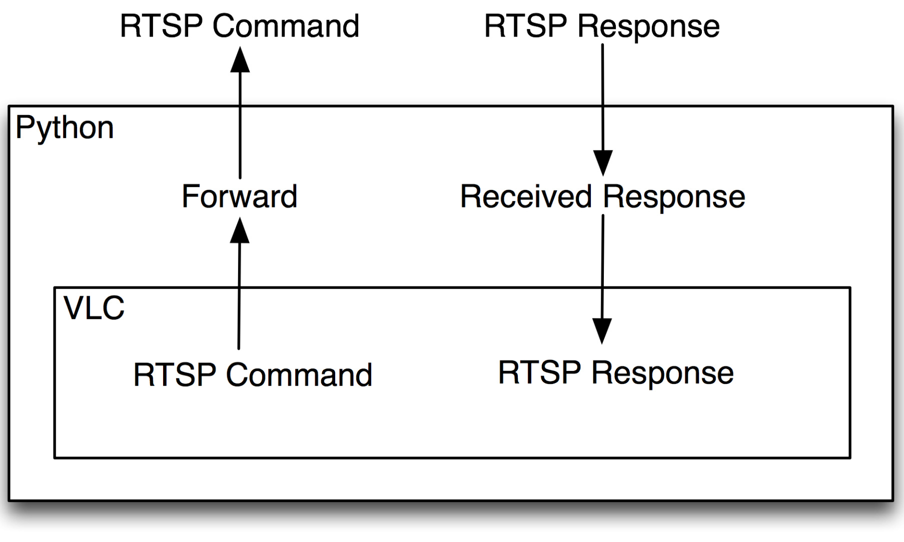

Figure 4.5: Python VLC Wrapper Default Behavior

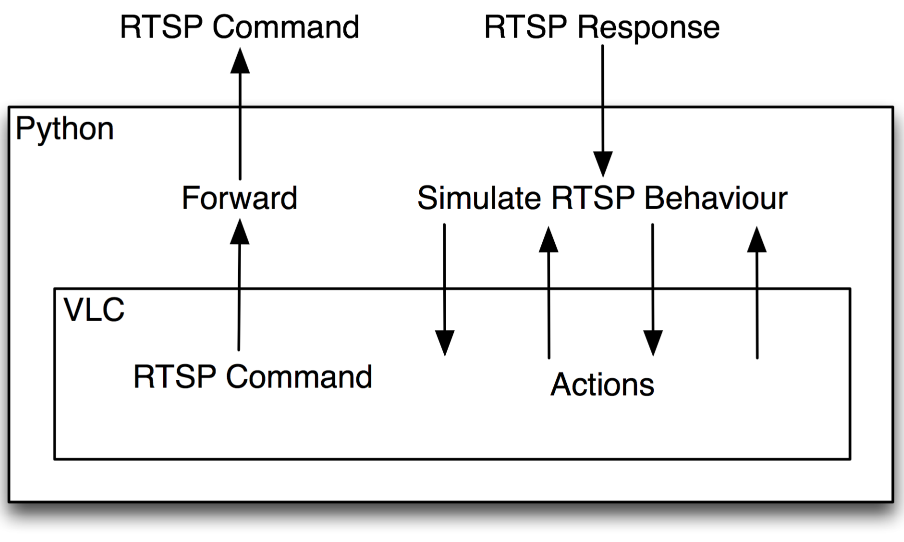

Figure 4.6: Python VLC Wrapper Simulating RTSP Default Behavior

4.5.3 Communication

RTSP

All the communication between the clients, the proxy, and the transcoders is done strictly using the RTSP protocol. The typical RTSP communication between the client and the platform can be seen in Fig.4.7. The steps go as follows:

1. The client sends a DESCRIBE command to the proxy, declaring its intention to view a specific file.
2. The manager finds an available transcoder and starts a transcoding job using the internal protocol. When the transcoder is ready to accept requests it notifies the manager using the internal protocol and the manager sends an RTSP reply with code 302 including a URL with the transcoder's address. Code 302 corresponds to a message saying that the requested resource has temporarily been moved to the location described with the URL. This is a way for the proxy to redirect the client to the assigned by the manager transcoder using RTSP.
3. The client then initializes an RTSP session with the transcoders using the commands OPTIONS, DESCRIBE, SETUP.
4. The client issues a PLAY command and the transcoding and streaming begins.
5. When the stream ends, the client issues a TEARDOWN command, the transcoding job ends and the transcoder is marked as *Available* and is ready to start another job.

However, when a transcoder serving a client becomes occupied from the host's owner, the default behavior of condor, is to send a SIGTERM signal to the transcoding process. This signal is caught by the python wrapper which will then terminate the local VLC instance that does the transcoding job and send a REDIRECT command to notify the client that it must resume the streaming from another transcoder. The client's wrapper

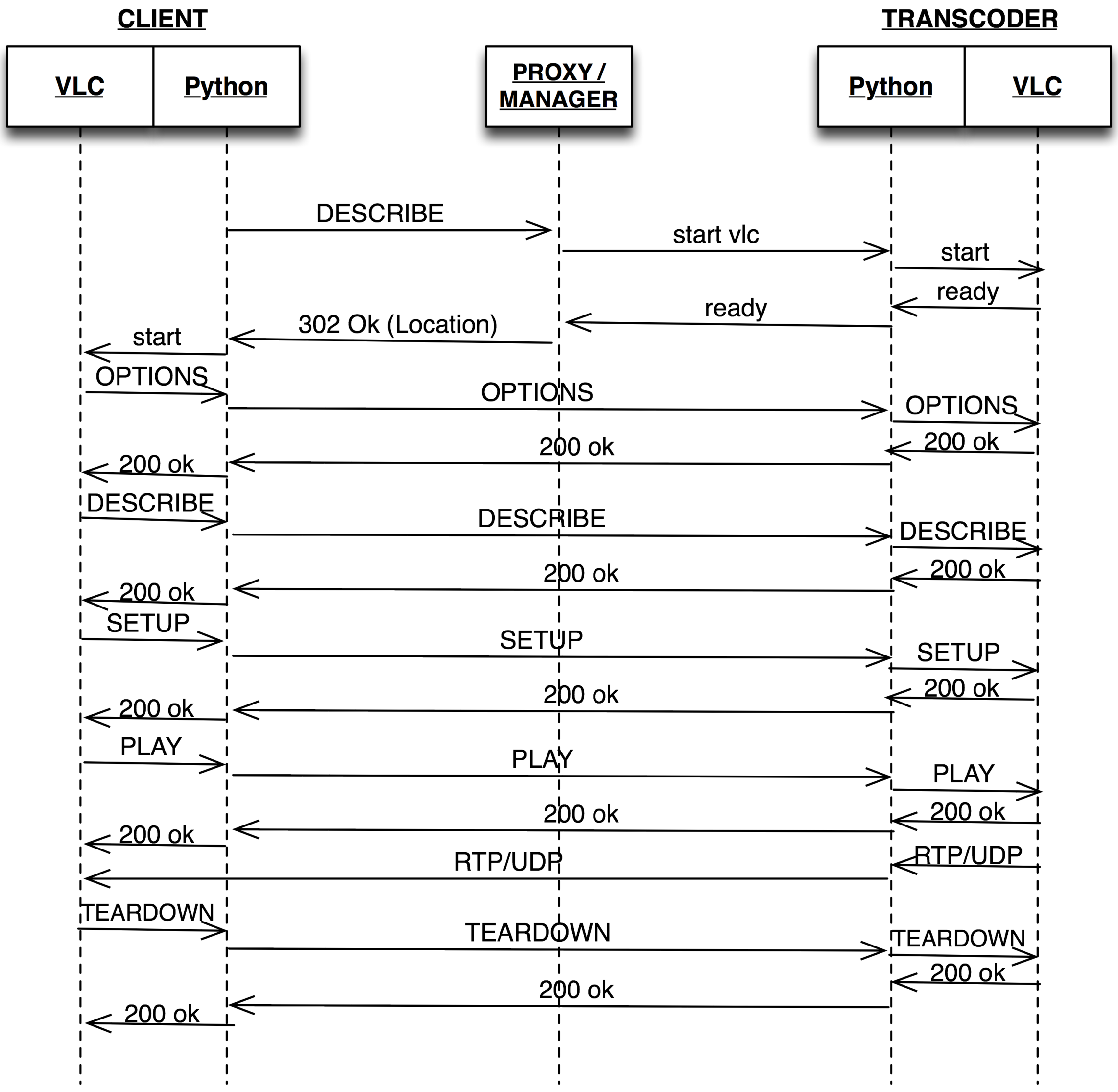

Figure 4.7: RTSP Command Sequence Diagram

will receive the REDIRECT command and terminate the local VLC that receives and renders the stream and will start a new one with the new location. The sequence of the commands is seen in Fig.4.8.

The session begins normally and when the transcoder receives the SIGTERM signal from condor, the REDIRECT command is issued. As explained earlier, VLC doesn't support the REDIRECT command and therefore, the command is identified by the python wrapper which then stops the stream and restarts it from the second that was stopped. The whole procedure is masked to the outside environment, and the VLC instance and python wrapper behave as seen earlier in Fig.4.6.

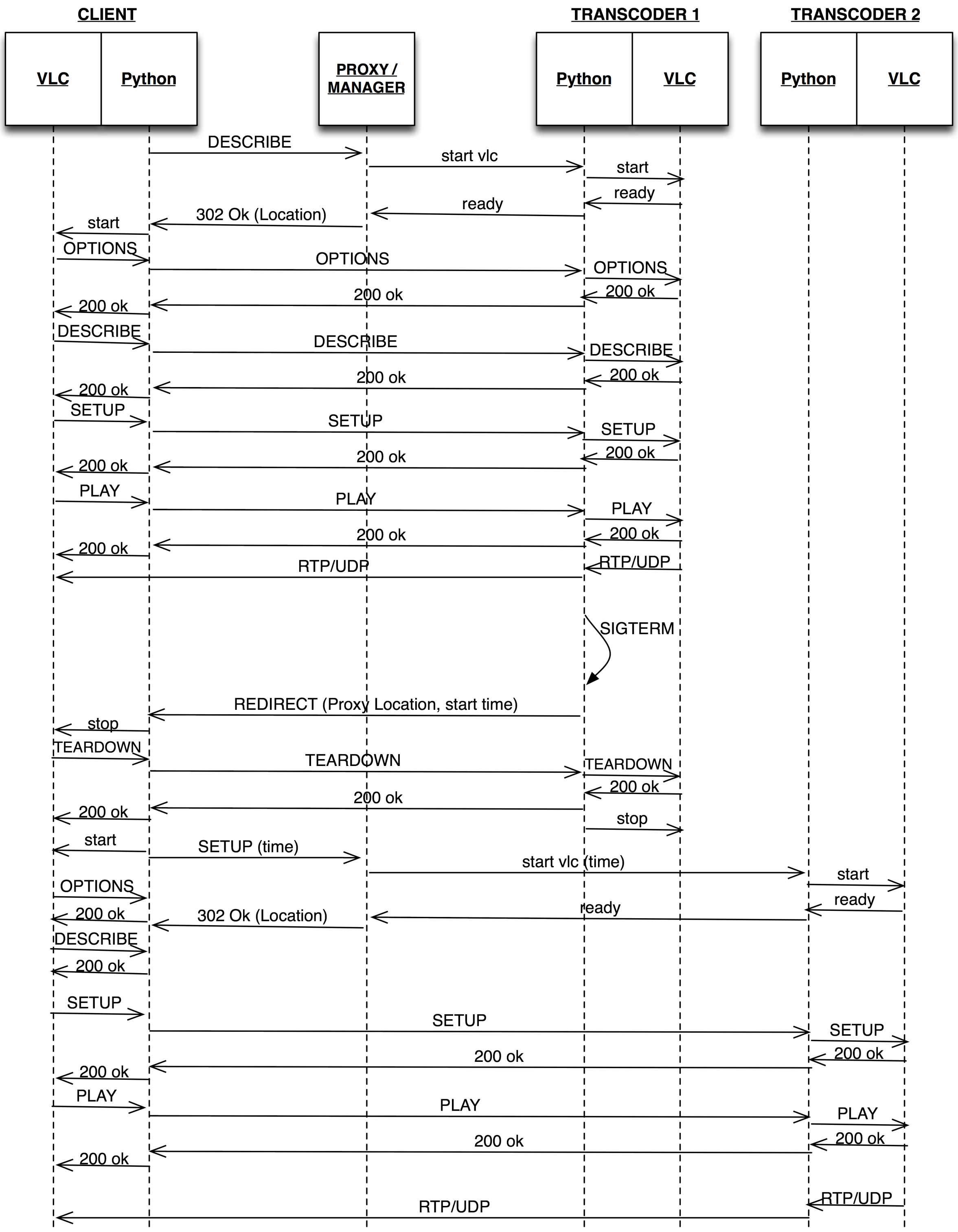

Figure 4.8: RTSP Command Sequence Diagram in Case of a Transcoder Change

Chapter 5

Evaluation

5.1 Noticeable Aspects

Condor was designed for executing batch processes. The human interaction only occurs at the job submission stage. The process' input and output are given in a file on the hard disk. This might not be sufficient for all different kind of jobs though and especially real-time and interactive jobs. In multimedia processing, factors like latency and asynchronous data distribution can play a critical role and affect the result. Thus, if we want to use Condor for multimedia processing in a transcoding on-demand application, we must take into account all the properties of multimedia processing as well as the particularities that the transcoding on-demand method will introduce.

Condor provides the resources that our application uses to perform the transcoding jobs. The opportunistic usage of these resources introduce two different problems: a) additional time to initiate a multimedia stream and b) the possibility of a transcoding job being interrupted.

5.2 Stream Initialization Overhead

Opportunistic resources require a centralized control that will manage their availability and assign the transcoding jobs. Therefore a monitor mechanism had to be implemented which added some extra complexity to the overall system. This mechanism is the manager combined with the proxy as explained in Chapter 4. A very sensitive point in the overall application that critically affects the QoE is the startup time of the stream. The startup time, is the time between the client's request and the video playback on the client's device. In a simplified application the client would send the first RTSP command directly to the transcoder and the only overhead would be the RTSP session establishment time. In our application, a few extra steps are needed before the RTSP session between the client and the transcoder can be established. These steps include the communication with the proxy, the selection of an available transcoder and the redirection of the client to that transcoder. In order to evaluate the performance of our system and calculate the overhead added, three different measurements were taken using: i) pre-transcoded content, ii) a direct transcoding on-demand scheme and iii) Video Condor.

5.2.1 Experiment Setup

The architecture used for the experiments can be seen in Fig. 5.1. All the hosts were in the same LAN keeping the latency amongst them to the minimum. For each experiment the VoD server and the client remained the same whereas the transcoding scheme was changing. In addition to that, the video and audio attributes of the streamed file remained the same to avoid different latencies or buffering policies `that` might have been added by different codecs, bit-rates or resolutions.

i) Pre-transcoded Content

A video file was pre-transcoded and stored on the VoD server. The file was streamed to the client by an intermediate host without transcoding.

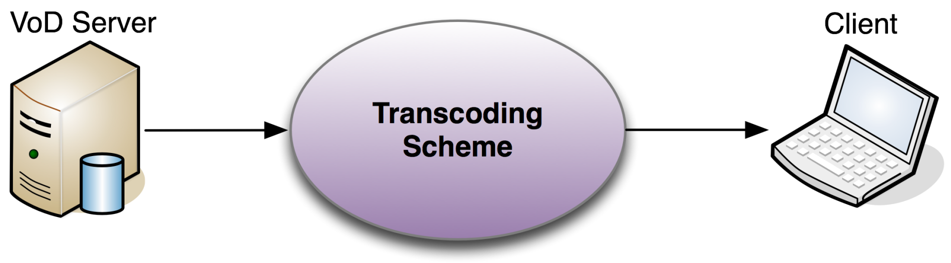

Figure 5.1: Experiments Architecture

ii) Direct Transcoding On-demand

A transcoder was setup in between the VoD server and the client. The client requests the file from the transcoder and the latest retrieves it from the VoD and streams it back to the client transcoding it on the fly using the same attributes (bit-rate, frames per second, codec) with the pre-transcoded one.

iii) Video Condor

Our architecture replaced the transcoder of the previous scenario. The content is streamed from the VoD to one of the transcoders in the condor pool while the client requests the content from the proxy. A condor pool of three hosts was setup as the transcoder pool. The proxy and the manager were on a different host. All of the above hosts were in the same LAN maintaining the latency between them to the minimum.

Condor

Condor version 7.0.1 was used with the default settings and configuration. Condor is very flexible and permits the change of many different properties that would critically affect the performance of Video Condor. However, changing the values of the properties that would improve the performance of our system would result in a non-realistic use of condor and therefore we decided to keep the default setting set by the creators of condor.

For example, condor allows custom defined policies to identify when a host is used by its owner. If the setting for that property is changed to 80% CPU usage, then this will mean that the host will be considered available as long as its CPU utilization is less than 80% which is obviously unrealistic and would result to all the hosts being available to run transcoding jobs. These default settings are the ones that would be used in a real life condor deployment and therefore making our evaluation of Video Condor more realistic.

5.2.2 Session Initialization Time

Due to the fact that a different multimedia players might use different buffering and rendering techniques, we consider the video playback time as the time that the first RTP packet arrives to the client's device. Moreover, we ignore the time needed by the device to send the first RTSP command after the client has chosen the content. In the experiments we measured the time needed between the client's initial RTSP request and the first RTP packet that arrived to the client. The results can be seen in Table 5.1 and in Fig. 5.2.

	Pre-Transcoded	Transcoding	Video Condor
Time (sec)	2.52	2.61	3.92

Table 5.1: Session Initiation Time

The time needed for a normal transcoding on-demand scheme is 2.61 seconds whereas the time for application to establish a multimedia session and receive the first RTP packet is 3.92 seconds. Therefore, the extra overhead that our application introduces is 1.31 seconds. This is the time needed for Video Condor to chose an available transcoder and assign a transcoding job to it.

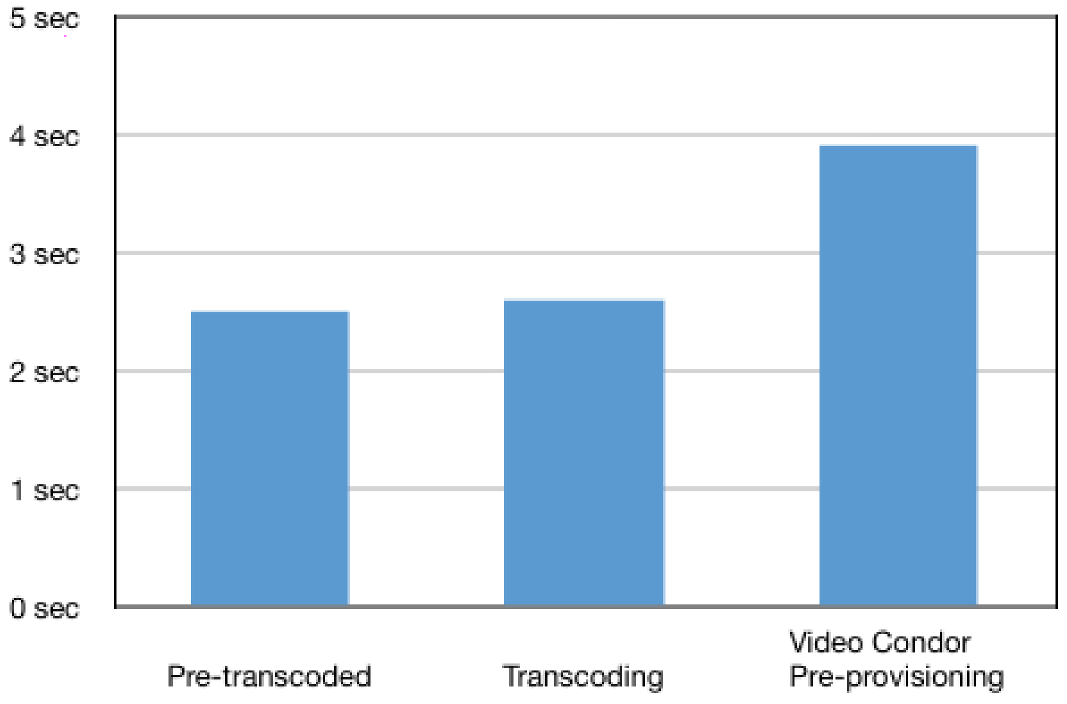

Figure 5.2: Session Initiation Time

5.3 Transcoding Job Interruption

In the context of such a dynamic environment is almost certain that a transcoding job will have to be interrupted due to a host being occupied by its owner. Therefore, the transcoding job has to be resumed by another host with the minimum impact on the QoE. However, due to the limited support of RTSP in VLC our application is not able to make that change transparent. Therefore, when a transcoder is being suspended by condor and another one takes over, this event is visible to the user. By manipulating the buffer size, this change can be made transparent. The video buffer should provide enough data until the new transcoder resumes the transmission.

When a REDIRECT message is sent to the client, it needs to end the current session and resume the stream by starting a new session with another host with the address included in the REDIRECT message. This address will be the proxy's address so that the client can have the address of a new transcoder from the condor pool through the manager as explained in Section 4.4. Therefore, three different events occur: a) the termination of the current session, b) the acquisition of a new transcoder from the proxy and c) the resumption of the stream from the transcoder provided by the proxy (See Fig. 4.8).

In our implementation, all the above events happen synchronously, resulting to the maximum amount of time that will be needed from the client to resume the stream. The time between the last RTP packet received on the client from the initial transcoder and the first RTP packet received on the client from the new transcoder was 4.8 seconds. This means that with the current implementation, if we want to make the transcoder change transparent to the user, the client needs to have a buffer size big enough to playback for 4.8 seconds without receiving new data.

However, an asynchronous implementation of the above events would result to the elimination of the time that the client will have to spend without receiving any RTP packets and therefore maintaining the QoE untouched and making the transition between one transcoder to another, completely transparent to the user. More precisely, the termination of the current session could happen after the proxy informs the client about the location of the new transcoder that will resume the stream. The client will trigger the Video Condor mechanism that is responsible for launching a new transcoding job before terminating the current session. By doing that the client will be able to start the new session before terminating the previous one. This is possible due to the fact that the default time that is given to a condor job to terminate before it becomes evicted on the occupied host is 10 minutes. This asynchronous implementation of the events done on the client in case of a REDIRECT command being received, can eliminate the time that will need to spend without receiving any RTP packets and therefore the change between the transcoders can be done without any influence on the QoE.

5.4 Clients

The use of a wrapper around VLC adds some extra overhead due to the fact that a large number of events that are masked from the external environment and the other parts of the architecture, happen between the local VLC instance and the wrapper. A simple client that would only understand OPTIONS, DESCRIBE, SETUP, PLAY, PAUSE, TEARDOWN as part of its RTSP support will be able to receive streams from Video Condor but will not able to resume the stream after a transcoder being suspended by condor. A client that is fully supported by our application must support the basic RTSP commands such as OPTIONS, DESCRIBE, SETUP, PLAY, PAUSE, TEARDOWN as well as the REDIRECT command and response codes 200, 302 and 404

Chapter 6

Conclusions

Multimedia processing using opportunistic resources is possible using a resource management system such as condor. An overhead is added which mainly corresponds to the opportunistic nature of the resources used and their management. The use of an opportunistic transcoding on-demand scheme can affect the QoE of the service but the overall benefits of such an application are enormous. Fully specialized streams are provided to clients with poor capabilities, using resources that are tolerant to failure while keeping the maintenance and management costs to the minimum.

On the other hand the use of opportunistic resources, introduces two problems a) additional time for the service initialization and b) high probability that the transcoding and streaming job will be interrupted. Video Condor adds 1.31 seconds extra on the time that is needed for a standard fixed transcoder to serve the content. In case a condor host getting occupied during during a transcoding and streaming job, the client will need to have enough data on the video buffer for an average of 4.8 seconds in order to make the transition between transcoders transparent to the user. However, these 4.8 seconds include a sequence of events done synchronously by the client. These events can be implemented to be done asynchronously and therefore the 4.8 seconds can be eliminated and the transcoder change can be completely transparent.

6.1 Future Work

Multiple Transcoding Jobs

Depending on the capabilities of a host that is part of the condor pool, it is possible to execute and serve more than one transcoding job at the same time and therefore making full use of the available resources.

Single Point of Failure

The proxy and the manager could take advantage of condor's mechanisms concerning fault tolerance and run as condor jobs in order to eliminate any single points of failure.

Dynamic Pool Support

Extending the idea of node pre-provisioning, the condor pool that is used for transcoding can be automatically adjustable depending on different factors such as the frequency of the incoming requests. As a result, there will always be enough hosts ready to execute transcoding jobs. A more complicated manager can be implemented in order to gather information about the behavior of the hosts and the clients and then using a statistical analysis on the collected data, decide when a new transcoder should be acquired or when an acquired transcoder should be released. As a result, the number of the gathered resources will be optimized and always have enough resources to serve the incoming requests. Meanwhile during low request rate periods the resources will be left untouched.

Transcoder Change

The use of opportunistic resources for transcoding and streaming purposes introduces the crucial issue of transcoding jobs being interrupted and resumed. Minimize the visual effect of a transcoder change by manipulating the RTSP buffer size or using an asynchronous implementation as described in Section 5.3.

Appendix A

Abbreviations

Short Term	Expanded Term
MP3	MPEG-1 Audio Layer 3
IPTV	Internet Protocol Television
QoS	Quality of Service
Wifi	Wireless Fidelity
3G	Third Generation of Mobile Phone Standards and Technology
NMM	Network-Integrated Multimedia Middleware
VLC	VideoLAN Client
API	Application Programming Interface
DRMAA	Distributed Resource Management Application API
GGF	Global Grid Forum
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
ASCII	American Standard Code for Information Interchange
RPC	Remote Procedure Call
XML	Extensible Markup Language
RTSP	Real Time Streaming Protocol
IETF	Internet Engineering Task Force
RTP	Real-time Transport Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4

Short Term	**Expanded Term**
IPv6	Internet Protocol version 6
GUI	Graphical User Interface
QoE	Quality of Experience
LAN	Local Area Network

Bibliography

[1] Amazon, inc. http://www.amazon.com.

[2] Amazon s3 webservices. http://aws.amazon.com/s3.

[3] Break.com. http://www.break.com.

[4] Condor Project Homepage. http://www.cs.wisc.edu/condor.

[5] Distributed resource management application api. http://www.ogf.org/documents/GFD.22.pdf.

[6] FFmpeg project. http://ffmpeg.mplayerhq.hu/.

[7] Google app engine. http://code.google.com/appengine/.

[8] Google, inc. http://www.google.com.

[9] Google video. http://www.video.google.com.

[10] Java programming language. http://www.java.com.

[11] live555 project. http://www.live555.com/.

[12] livemedia library. http://www.live555.com/liveMedia.

[13] Metacafe. http://www.metacafe.com.

[14] Network-integrated multimedia middleware. http://www.networkmultimedia.org/.

[15] Predictor@home:. `http://predictor.scripps.edu`.

[16] Python programming language. `http://www.python.org`.

[17] Videolan - vlc media player. `http://www.videolan.org/`.

[18] World wide web consurtium - web standards. `http://www.w3.org/TR/di-gloss/`.

[19] Youtube.com. `http://www.youtube.com`.

[20] Seti@Home., 2003. `http://setiathome,ssl.herkeley.edu/`.

[21] D.P. Anderson. Boinc: a system for public-resource computing and storage. *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, Nov. 2004.

[22] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System, 2003.

[23] Inc. Cisco Systems. Video application components and architecture, Sep. 20,2006. `http://www.cisco.com/univercd/cc/td/doc/solution/vobbsols/vob1/vbdig/vbvid1.htm`.

[24] Bo Shen Dongyu Liu, Eric Setton and Songqing Chen. Pat: Peer-assisted transcoding for overlay streaming to heterogeneous devices. In *Proceedings of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2007)*, June 4-5, 2007.

[25] Miron Livny Douglas Thain, Todd Tannenbaum. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):97–101, 2005.

[26] D. Stainforth et al. Climateprediction.net: Design principles for public-resource modeling research. 2002.

[27] L. Goasduff and C. Forsling. Consider revenue models for mobile tv carefully, gartner counsels, Mar. 27,2007. `http://www.gartner.com/it/page.jsp?id=503578`.

[28] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[29] Jiun-Long Huang, Ming-Syan Chen, and Hao-Ping Hung. A qos-aware transcoding proxy using on-demand data broadcasting. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 3:2050–2059 vol.3, March 2004.

[30] Verena Kahmann, Jens Brandt, and Lars Wolf. Collaborative streaming in heterogeneous and dynamic scenarios. *Commun. ACM*, 49(11):58–63, 2006.

[31] Jin Liang and M. Nahrstedt. Supporting quality of service in a non-dedicated opportunistic environment. *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 74–81, April 2004.

[32] M. Litzkow and M. Livny. Experience with the condor distributed batch system. *Experimental Distributed Systems, 1990. Proceedings., IEEE Workshop on*, pages 97–101, Oct 1990.

[33] Wai Yip Lum and F.C.M. Lau. A context-aware decision engine for content adaptation. *Pervasive Computing, IEEE*, 1(3):41–49, 2002.

[34] Zhuoqing Morley Mao, Hoi sheung Wilson So, and Byunghoon Kang. Network support for mobile multimedia using a self-adaptive distributed proxy. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 107–116, New York, NY, USA, 2001. ACM.

[35] Giovanni Novelli, Giuseppe Pappalardo, Corrado Santoro, and Emiliano Tramontana. A grid-based infrastructure to support multimedia content distribution. In

UPGRADE '07: Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks, pages 57–64, New York, NY, USA, 2007. ACM.

[36] R. Raman, M. Livny, and M. Solomon. Matchmaking: distributed resource management for high throughput computing. *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146, Jul 1998.

[37] R. Raman, M. Livny, and M. Solomon. Resource management through multilateral matchmaking. *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, pages 290–291, 2000.

[38] Rajesh Raman. *Matchmaking frameworks for distributed resource management.* PhD thesis, 2000. Supervisor-Miron Livny.

[39] Sumit Roy, Bo Shen, Vijay Sundaram, and Raj Kumar. Application level hand-off support for mobile media transcoding sessions. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 95–104, New York, NY, USA, 2002. ACM.

[40] Sumit Roy, Bo Shen, Vijay Sundaram, and Raj Kumar. Application level hand-off support for mobile media transcoding sessions. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 95–104, New York, NY, USA, 2002. ACM.

[41] P. Salomoni and S. Mirri. A multimedia broker for ubiquitous and accessible rich media content transcoding. *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, pages 186–191, Nov.-3 Dec. 2004.

[42] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[43] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), April 1998.

[44] John R. Smith, Rakesh Mohan, and Chung-Sheng Li. Scalable multimedia delivery for pervasive computing. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 131–140, New York, NY, USA, 1999. ACM.

[45] Jarrod Chapman Sidney P. Elmer Siraj Khaliq Stefan M. Larson Young Min Rhee Michael R. Shirts Christopher D. Snow Eric J. Sorin Bojan Zagrovic Vijay S. Pande, Ian Baker. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing, 2003.

[46] Ye-Kui Wang, Imed Bouazizi, Miska M. Hannuksela, and Igor D. D. Curcio. Mobile video applications and standards. In *MV '07: Proceedings of the international workshop on Workshop on mobile video*, pages 1–6, New York, NY, USA, 2007. ACM.

[47] Dongsong Zhang. Web content adaptation for mobile handheld devices. *Commun. ACM*, 50(2):75–79, 2007.