7-2008

L-encoder: Video Transcoding in the Logistical Network

Harold Thomas Gonzales
University of Tennessee, Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Harold Thomas Gonzales entitled "L-encoder: Video Transcoding in the Logistical Network." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Micah Beck, Major Professor

We have read this thesis and recommend its acceptance:

David Straight, Jian Huang

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Harold Thomas Gonzales entitled "L-encoder: Video Transcoding in the Logistical Network". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Micah Beck, Major Professor

We have read this thesis
and recommend its acceptance:

David Straight

Jian Huang

Accepted for the Council:

Carolyn R. Hodges, Vice Provost
and Dean of the Graduate School

(Original signatures are on file with official student records.)

L-encoder: Video Transcoding in the Logistical Network

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Harold Thomas Gonzales

July 2008

Dedication

To my parents, who gave me life and my first computer.

Acknowledgments

I would like to acknowledge first the members of my committee for taking the time to serve. I would like to acknowledge my advisor, Dr. Micah Beck for providing me an outstanding example of passion and dedication in computer science research. His example as a teacher and a leader will accompany me for a long time. I would also like to thank Dr. Terry Moore for his lessons in communication and collaboration and for always having an open door for complaints and grumbles of graduate students. I must thank my fellow graduate student Chris Sellers who has been my logistical networking guru. His work ethic is unparalleled and his patience admirable. I would like to thank all of my instructors in the Computer Science department who have made me feel welcome and continued to inspire me over my six years in Knoxville.

I must thank the many friends in Knoxville who have supported me throughout my time at UT. I would especially like to thank Tom Anderson and Eric Moore, who have never complained about my computational ravings over the years. I thank also Jon Sykes and Jerod Hollyfield for ensuring I will never have an acting career. I also thank all the regulars at the Leaf and Ale, whose wit, insight, and wisdom prove that a cigar is never just a cigar.

Last I want to thank my family who have always believed in me. Without you I could never have gotten this far.

Abstract

Transcoding, the transformation of digital information from one encoding format to another, is a prominent operation in the realm of digital video and audio. This process of changing the encoding formats of multimedia files and streams is now a common task for many users due in part to the prevalence of portable media players and consumer electronic devices for digital media, which place constraints on the file and stream formats they can play. The transcoding process is often both data and compute intensive due to both the large data requirements of modern media formats from sources such as HDTV and the computational complexity of changing encoding formats. There is motivation then to create a network transcoding service that would allow users to leverage the computational and storage resources within a network against this task.

The Internet Backplane Protocol (IBP) and the Network Functional Unit form the backbone of a network computational paradigm called Logistical Network Computing (LoNC), which outlines the design of scalable computational services within a network. A proof-of-concept transcoding service, L-encoder, was created within this paradigm based on the open source mencoder program. L-encoder was designed to allow users to offload computation onto shared computing and storage resources and to allow for parallel processing to achieve improved performance. Experiments were conducted to determine the performance improvements due to parallelization, and an analysis was undertaken to assess the feasibility of deploying the L-encoder service in a large-scale network infrastructure as well as the potential for future research in this area.

Contents

List of Figures

Chapter 1

Introduction

Video transcoding, the process of transforming encoded video data from one encoding scheme to another, is a well studied problem in digital multimeida. The problem has come to increasing prominence with the prevalence of mulitmedia enabled personal devices such as personal media players, video enabled cell phones, and personal video recorders (PVRs). These consumer devices often have specific requirements for the encoding of audio and video streams they support, and, as a result, the need for transcoding solutions for individual users and content distribution infrastructures has increased. The prevalence of High Definition Television (HDTV) has also led to increased focus on transcoding due to the large requirements for bandwidth in transmission and storage space for recording of content.

On the infrasturcutre side, developments have primarily focused on solutions involving specialized, dedicated hardware such as the TI DaVinci chip [4]. These large productions systems have been used to create large distribution networks for streaming on-demand multimedia services such as the Verizon VCast mobile TV service [2]. On the user side, development has focused primarily on software based approaches. There are many open source solutions that provide end-users with transcoding suites capable of handling the demands of modern multimedia formats. These software based solutions, however, come at a high cost in terms of resource needs. These include both the substantial computational cost of the transcoding operations, and the storage cost for housing both the original data in its original encoding and the newly produced, transcoded data. This research explores

the posibility of a network computational solution that would allow users to leverage resources within the network against the resource requirements of the transcoding process. It describes a proof-of-concept network transcoding service, L-encoder, the technologies underlying it, its efficacy as both a simple offload service, and the potential gains it can provide the user by exploiting parallel execution. This document begins with an introduction to the technologies underlying L-Encoder such as Logistical Networking and the Mencoder suite. It then describes the approach to developing both the server and client software, results from parallel execution tests, and an analysis of the potential for such a service to be deployed in a research testbed environment.

1.1 Logistical Networking

Logistical Networking is an approach to the coscheduling of shared resources within network computer systems based on a conventional logistics model, which deals with the coscheduling of transportation, storage and processing resources in industrial systems [10]. Logistical networking seeks to provide scalable interfaces to the fundamental resources underlying network systems: bandwidth, storage, and computation. It is comprised of several core technologies which provide these interfaces. A diagram of the logistical storage stack can be seen in figure 6.1. The network architecture in logistical networking is based on the same layering principles as the TCP/IP stack, where more general, lower level services are implemented in the lower layers. In this case the first logistical layer comprises the Internet Backplane Protocol, which is the best-effort storage service for exposing storage resources. Upon this are built higher level abstractions which provide additional functionality. The Logistical Backbone (L-Bone) is a directory service for IBP depots. It allows users to search for depots based on a set of criteria such as available resources and location. The Network Functional unit which provides the computational aspect of logistical networking can be thought of as sitting along side the IBP layer. In this sense IBP is the fundamental storage interface within the logistical network, while the NFU is the fundamental compuational interface. To understand the role these technologies play in creating a transcoding service within the logistical network, it is necessary to examine each one in more detail.

1.1.1 Internet Backplane Protocol

The Internet Backplane Protocol provides a scalable, best-effort, end-to-end storage service within a computer network [5]. Storage servers, known as depots, allow users to request allocations of storage. These allocations are awarded on a best-effort basis and thus can be denied due to resource constraints. The storage allocations granted are of limited duration and can expire at any time. This service allows for clients to store large ammounts of data on publicly shared resources, thus making it possible for users to upload files that they intend to transcode into the network where they are then operated upon by a network computation service, the NFU.

1.1.2 Network Functional Unit

The Network Functional unit extends the Logistical Networking philosophy of abstracting resources within a network system to computational resources [9]. The application of this computational aspect to data stored within IBP depots is known as Logistical Network Computation. The NFU operates on data based on a set of operations known as NFU operations or NFU ops. NFU operations can be described as an abstract transformation of state stored within the network. NFU operations take data stored within IBP allocations as input and produce output which is in turn stored within IBP allocations. Thus the data interface to NFU operations is provided with the same best-effort conditions as all other IBP allocations. This model also allows the NFU to maximize its independence from issues such as operating system and underlying hardware platform. Thus the NFU strives to provide a maximally generic network computation service that can be made publicly available.

The exposure of computational resources to clients in the network requires consideration of issues such as availability, security, scalability, and management. The computational resources provided by the NFU are allocated on a best-effort basis. No assurances are given as to the avaibility of computational resources and their at present exists no reservation scheme in contrast to many popular grid computing services. This model allows the NFU to remain flexible and places the burdens of scheduling and managing computation at the endpoints of a computaitonal transfer. Currently there are multiple implementations of

the NFU and no unified standard exists for addressing security concerns. Previous research into this problem has focused upon limiting permitted modes of computation within the NFU, such as restricting which, if any, system calls a given operation is allowed to perform. There have also been efforts in limiting access to resources such as memory and limiting the runtime of programs to serve as a ward against denial of service attacks. In this model, as with many current network services, the potential for service disruption and denial is still a possibility. In the area of management, each NFU operation must be assigned a unique opcode which a client program uses to invoke the desired operation. At present there is no globally managed directory for such operations, and no resource discovery service exists to acertain whether a given NFU supports a given operation and if it has been assigned the expected opcode. With these considerations in mind it is still possible for the NFU to provide computational services within the network.

Previous work involving image processing has shown the potential for the NFU as a shared computing resource [6]. With these logistical networking tools in place, it was conceived that an NFU based transcoding service could be developed. The development of such a codebase from scratch is prohibitive. Additionally, from a user interface point of view, it would be desirable for users familiar with a popular transcoding package to be able to use this new service with a minimal learning curve. It was decided that for this transcoding service an open source, actively developed transcoding suite would be selected and modified to execute within the NFU.

1.1.3 Mencoder

Mencoder is an open source video transcoding package developed as part of the MPlayer project. It includes support for many modern encoding formats both through its native codec support and through libavcodec which is part of the ffmpeg project [3]. It has a large active user base and is supported on multiple operating systems including Windows, Linux, and Mac. These factors all contributed to the selection of mencoder as the codebase for a logistical network computing transcoding service known as L-encoder. The approach to the L-encoder design is covered in the next chapter.

Chapter 2

Approach

In modifying mencoder to create a logistical network transcoding service, there were many factors that needed to be taken into consideration. Adapting mencoder would mean adding the capability to interface with IBP stored data within the network. The current version of the LoCI NFU, which was used for the development of lencoder, passes the capabilites used by an NFU op as a memory map. This abstraction of pages in virtual memory allows for the operation to access capabilities as memory areas, divorcing this access from both the IBP protocol and the underlying file system. This also meant that modifications were necessary to the mencoder codebase to allow for the use of these memory maps.

Both MPlayer and mencoder use a generic datatype stream to interface with input and output data. These stream types have been used to allow for support of streaming media, physical media such as DVDs, and also files in networks via FTP. For the NFU operation a new streamtype was created to handle both input and output streams through the NFU mmap interface.

Additionally, the NFU paradigm frowns upon the idea of temporary files or any other sort of data associated with NFU operations coming from a source other than the depot (ie, the local file system). Workarounds were created to remove the situations in which mencoder would attempt to access the local file system except for the loading of shared libraries and other data associated with the program loading rather than during its execution.

In addition to the modifications to mencoder itself, a client program had to be written to invoke the remote NFU operation and to allow the upload and download of data during

the transcoding process. This was accomplished by a C client program using the NFU and IBP libraries from the 1.4 version of the LoCI IBP depot. The client program handled the task of uploading the input file to the IBP depot, and also uploading other pieces of information necessary for proper execution such as the command line arguements. To maintain compatibility and exploit familiarity, the command line arguments to the L-encoder client were the same as to the mencoder program with two exceptions. The first being that a depot must be specified as the first argument and the second being that the port on which it is accepting connections must be specified as the second argument. This scheme assumes that the user is aware of which depots are providing the L-encoder op, and that they have all properly assigned the operation the proper opcode.

Once the proof-of-concept encoder and client were completed, tests were devised to determine the ability of the program to operate within the network and the ability to exploit the inherent parallelism of multiple depot resources within the network.

Chapter 3

Performance Testing

In order to test the potential for L-encoder as a network transcoding service, tests were devised to examine both its efficacy in a practical example and also its potential for parallel execution. For this test the transcoding of a recorded HDTV stream was chosen. The selection considered several factors. The increased prevalence of HDTV has placed large demand on bandwidth for transmission of streams and on storage devices such as PVRs for its archival for future viewing. Transcoding the stream into one of similar quality, but greatly reduced size, would then seem an ideal task for a transcoding suite. Also, the fact that HDTV is encoded as a transport stream, which includes both video and audio data within individual packets, means that archived data can be easily subdivided into smaller segments without the need for a program to split based upon a container format. The tests would then involve transcoding these HDTV streams into a popular video codec. The Xvid codec is an MPEG-4 standard compliant video codec known for its production of high quality video at relatively low bitrate [1]. This is one reason that Xvid is a popular choice for the encoding of captured HDTV content. For this test the HDTV source streams would be a 1080i streams of varying file sizes each approximately 100M, 200M, 400M, 800M and 1600M. The actual file sizes and their playback length are displayed in the following table.

The relatively large file size for a small amount of content illustrates another motivation for a network offload service. The video would be transcoded into 2000kbps Xvid format, while the audio stream would be unmodified, which is a dependable way to ensure audio sync will not be damaged during the transcoding process or require human intervention to

Table 3.1: File Sizes in Bytes and Lengths in Seconds

Size	Length
102257430	32.72s
204514860	65.44s
409029908	130.88s
818059816	261.77s
1636119820	523.56s

maintain the sync.

3.1 Mencoder testing

For comparison purposes, the parallel test of L-encoder was compared to a parallel execution of mencoder. Mencoder has no native support for parallelism, thus a test was devised that would allow for a similar mode of execution between the mencoder test and the test of L-encoder's parallel potential. For this test the file would be housed and broken into the desired number of segments on a client machine running a test automation script. Each segement of the file would then be transfered to remote hosts via SCP. The mecoder command would then be invoked via SSH on the segments on the remote machine and the result transfered back to the client machine via SCP. The time from invocation to completion of the final segment download would then be recorded for analysis. The tests were conducted using the Cetus lab in the Computer Science department at the University of Tennessee. The Cetus machines are Dell OptiPlex GX280s with Pentium 4 Processor 540 (3.2 Ghz) processors and 1GB of RAM. The testing schedule was as follows:

Tests were conducted based on number of segments. These ranged from 1 segment (the whole file) to 20 segments.
Tests were repeated for each of the five previously mentioned file sizes.
Tests were repeated five times for each file and number of segments pair.

The results of the five runs for each file size and number of segments were collected. Averages were taken for each file size and number of segments pair. The average runtimes

are depicted in figure 6.2.

Another useful means of visualizing parallelization gains is by a metric called speedup.

$$speedup = \frac{Uniprocessortime}{TimewithNprocessors} \tag{3.1}$$

The speedup in this case represents relative speedup, since in the single segment case, the network transfer is still performed. This means that even in the case of only one host being used for computation the algorithm is still "parallel" rather than a true uniprocessor time, which would only involve executing mencoder on a local file. The speedup results for the mencoder test can be seen in figure 6.3.

The graph indicates that the parallelization of mencoder falls short of the ideal speedup, which is a linear speedup where speedup is equal to the number of hosts used. Although it falls short of ideal speedup, these results do show a significant reduction in the time necessary to transcode files. The results of these tests are used as a benchmark against which the tests of L-encoder shall be judged.

3.2 L-encoder Testing

The tests of L-encoder were designed to mimic as closely as possible the parallel tests of mencoder. This is not only a convenience for testing procedure but represents a parallel model for L-encoder execution in the most simple case. In the test only one file segment is allocated per depot. The L-encoder NFU op is then called by an instance of the L-encoder client program for each depot containing a file segment. In this way both the execution of mencoder in parallel and the L-encoder test follow as close to an identical parallel execution as possible. The average execution times for the L-encoder tests can be seen in figure 6.4.

As with the mencoder tests, we can see that the addition of hosts in this test causes an effective reduction in average runtime. As was the case with mencoder the speedup for the L-encoder test was also calculated and graphed. This can be seen in figure 6.5.

In the case of several files in the L-encoder test, the speedup appears to be significantly inferior to that of mencoder operating under the same conditions. This is most striking for the three file sizes where performance falls off dramatically at the eleven hosts datapoint.

The reality behind this performance falloff appears to be much simpler than a flaw in the L-encoder design. Due to time constraints testing had to be performed on a public computer lab with no control over the load on the machines at any given time. During the mencoder test this was not problematic, since none of the hosts had significant jobs running on them for long periods of time. During the L-encoder test, however, this was not the case. In order to speed up testing cycle, the data points for the L-encoder test up to ten hosts were performed on two subsets of the data simultaneously. Since these are the slower half of the tests, this contributed to great time savings in the testing schedule. The tests for more than ten hosts were all conducted by the same manager script and were run at a time where other jobs were present on the lab machines. The only test that could be run before these other jobs started, was the 100M test, whose performance is clearly superior. Far from invalidating the results of this test, this phenomenon demonstrates two important concepts in terms of a network computation service. The first is that the test of L-encoder and mencoder for parallel potential is a simple case with a single job running on each machine. The second is a more interesting issue of whether parallel gains can still be made in an environment where multiple NFU operations are contending for the same resources. This test shows potential in this area, since although the speedup decreased at the eleven host data point, it then continued to increase, albeit slower than the mencoder test, after that point. The graph in figure 6.6 displays a simple test involving multiple L-encoder clients transcoding the 400M file on ten depots. The single client data point is the average from the L-encoder parallel test. The others were compiled in a subsequent test, so the extreme slope from the first data point to the second is most likely an exaggeration of the expected difference.

The graph in figure 6.7 illustrates two important points about the issue of multiple NFU operations being run simultaneously by the same depot. First it shows that, as would be expected, the runtime for the individual clients does decrease with each client added; however, the global throughput in terms of number of files transcoded in a given time, actually increases somewhat with multiple clients. This increase in overall throughput is advantageous in a network computing service, as it allows for the shared use of resources to accomplish more global work while slowing individual progress. One test with 25 clients yielded a total runtime of 3956 which is a fairly graceful degradation for that number of

clients. Graphs of the ratio of total execution time to number of clients can be found in figures 6.8 and 6.9.

It is evident that this graceful degradation cannot be expected in all situations. The oversubscription of the depot resource will eventually decrease both individual performance and global throughput of NFU operations. Expanding this concept to the network at large, we see the possibility for the oversubscription of shared computational resources across the network creating a computationally congested network. In this case not only will individual clients suffer, but an attempt to increase parallelism to alleviate this slowdown could actually increase overall computation time in a phenomenon analogous to network congestion. In these cases it would be advantageous for the client program to implement scheduling capable of identifying congestion and taking a response with the potential to ease the problem. This idea of computational congestion control has the potential to be a complex scheduling problem. A discussion of this issue and potential approaches is found in the next chapter.

Chapter 4

Analysis

An important aspect of a proof-of-concept services such as L-encoder is the potential for its development into a production service. To this end an analysis was undertaken to examine the various elements of the L-encoder service which may require examination before such a service were deployed into a testbed infrastructure.

4.1 Security Issues

The discussion of a network computational service would be incomplete without considerations of computer security. In this case it is necessary to examine potential security issues from several vantage points.

The initial concern must be for the security of the NFU itself. Assuming that the computation perfomed in the NFU is an arbitrary execution of code provides a worst case scenario for security concerns. Since administrators of depots must be willing to at least assign an opcode to a given NFU operation and in many cases must also compile the given operation for the depot, in most cases this means that some cursory examination of underlying execution will be performed. However, in designing a network architecture that has the potential to be publicly available, our analysis must begin with the idea that arbitrary code can be executed within an NFU. This assumption is quite alarming and demands that restrictions be placed upon the execution behavior of given NFU operations.

4.1.1 Restrictions Placed Upon Program Behavior

Initial ideas concerning the restrictions placed upon NFU operations in order to limit their potential malicious impact have attempted to isolate NFU operations as much as possible from the underlying system. Efforts were made to restrict or eliminate the ability of the NFU op to make system calls, thus limiting the scope of the operation's influence and its potential for abuse. In the case of L-encoder the complete elimination of system calls is impractical because of its need for dynamic memory. The mencoder package contains malloc calls that both require a size of memory that is unknown till runtime, and are fairly ubiquitous regardless of codec. A potential alternative of using IBP allocations for these buffers is, therefore, an unreasonable burden for L-encoder's initial creation and would greatly complicate the work of any maintenance programmer. The restriction of certain system calls, however, has the potential to be implemented without any change to the code. Preventing the NFU operation from opening files or sockets in this case would result in an increase in security without necessitating any changes to the program's design.

4.1.2 Restrictions Placed Upon Resource Consumption

Even if the L-encoder operation can be reasonably hardened to meet security concerns, additional problems can arise in the amount of resources the operation can consume over a given time. There are multiple concerns associated with NFU op resource consumption. First, there is the issue of load on the depot affecting the underlying IBP service. Especially in the case of multiple running operations this could negatively impact the quality and availability of the IBP service for the depot. NFU operations could also include a large number of large parameters, which could affect the quality and availability of the IBP service. In addition operations that run for extremely long or indefinite amounts of time could affect a denial of computation attack against the NFU, rendering the service unusable.

Initial efforts into ameliorating these problems focused on the use of resource limits for NFU processes. This could include limiting the CPU execution time, amount of memory available, etc. In the case of L-encoder this type of solution poses a challenge. Due to the plethora of encoding formats and their relative computational complexity, it is difficult to estimate the amount of CPU time necessary to complete a given transcoding

operation. The same issue presents itself with the dynamic memory requirements that each codec may have, posing problems with the ability to make reasonable assumptions about necessary resources to complete operations. Additionally, the client program has no information about the particular hardware that the NFU operation will be using. This information could drastically effect the estimation of necessary resources for operation completion. An end-to-end argument would suggest that given reasonably consistent NFU performance, it should be the duty of the client program to ensure that computation is dispersed and scheduled so that operations may complete under the given constraints. This scheduling algorithm would most likely require a significant amount of detail as to the specific resource restraints of an NFU, and at the present time no method exists for ascertaining this information from the depot.

It should be stressed at this point that the NFU is an experimental platform. This provides a certain degree of optimism that future study into resource consumption issues may produce a limitation scheme such that a client scheduler could be created that could still produce performance gains using parallelism, but also be aware of the potential impact of the operation within the network.

4.2 Fault Tolerance

As with any network service, it is prudent to examine the fault tolerance of L-Encoder. Currently the L-encoder client operates on a fail-stop model. If the NFU operation terminates before completion, an error is returned and all computational work to that point is lost. In the case of a program error, this is not of great concern, since there was no chance for the operation to complete, but this situation is undesirable especially in the case of large files with a long transcoding operation where the depot revokes the computational resource. The management of a parallel encoding job could in this case keep a list of available depots and simply reassign that portion of the encoding task to another depot as it becomes available.

A checkpointing algorithm could be introduced that would allow for the resumption of a partially completed job, but this would require several serious alterations to the NFU operation's execution. For instance there would have to be additional IBP allocations made

to hold the necessary checkpoint information. Also, the memory maps used to interface with the capabilites would have to be explicitly flushed at time of checkpoint, to ensure that data written to the capability would match that in virtual memory. A failure during the flush operation could then produce an unrecoverable error, which would need to be detected and dealt with. There is also the issue of migrating data from one depot to another and the typical issues of checkpoint recovery in migrating schemes, such as resetting the pointers to the memory maps if that virtual address is no longer available.

The issue of fault tolerance could also be ameliorated with a more refined scheduler algorithm. A scheme using a dynamically-ranked pool of depots and a two-level priority queue has been implemented and tested when applied to distributed visualization [7]. This scheme focused on the reassignment of failed tasks to depots that have historically better performance. This sort of scheme could potentially be applied to L-encoder tasks, though future work will have to be undertaken to implement such a scheme and ascertain its efficacy in this case.

4.3 Management Issues

The deployability of a network service is conditioned not only by complications in development but also in terms of management of a deployed system. L-encoder for instance must be compiled or packaged as a binary for any architecture upon which an NFU may run. This could present complications in terms of software infrastructure management and version homogeneity. Additionally, the client assumes that the NFU op has a unique opcode, but no directory service for such opcodes currently exists. This architecture, therefore, places these burdens on individual NFU administrators to both maintain the operation and to make the availability of the operation known to potential users. A resource discover protocol such as the L-Bone could be augmented to include information about the NFU operations that depots provide and their version. This would allow users to dynamically discover these resources and remove from the administrator the burden of advertising the service. Additionally a register of NFU opcodes could be kept, with a reasonable reservation procedure, which would allow for easier management of the multiple NFU operations that may be provided. Such a scheme would allow for both the flexibility for administrators to

dedicate heterogeneous hardware resources to the IBP and NFU services and an increased confidence in the user community that information about available NFU operations would be readily accessible and up to date.

4.4 Congestion Control

As introduced in the previous section, the issue of congestion control can become a significant issue in network systems. In this case the congestion is of a computational nature. The presence of overladen depots could cause both decreases in individual performance and overall throughput. The example in the previous chapter was a simplified one, considering that the workloads and their distribution among the depots were both homogeneous. A more general problem involves the identification of congestion points at certain depots in an environment where work is unevenly distributed and then implementing a control scheme that will allow that congestion to be eased.

4.4.1 Congestion Detection

The detection of a congestion point is the first step to alleviating it. In this case we examine both an attempt to detect congestion on the depot itself and a congestion detection scheme that operates at endpoints.

4.4.2 A Depot Based Approach

In an end-to-end congestion detection scheme, the only information about the state of congestion at a depot is gathered by the endpoint. There are network congestion control methods that eschew this purely end-to-end approach and incorporate the ability for routers to send explicit congestion messages. In the NFU as it is now, there exists no means of signaling observed congestion to a client attempting to invoke an operation. In this case the idea of congestion could be a naive metric such as the number of active operations or something more sophisticated such as a metric based upon the consumption of specific resources such as memory and CPU load. This metric also could be more than just a binary measure of congested or not congested. In such a system the client function, which is called on the client side to invoke the operation, could have an additional parameter which

specified the maximum congestion that it would tolerate from a depot. An appropriate error code could be returned and allow the client program to attempt to invoke the operation on a less congested depot. This scenario where a client voluntarily gives up on an attempt to invoke is useful, but the depot should also have a threshold of congestion above which the invocation is not only discouraged but explicitly denied. This is analogous to the dropping of packets in networks. Unfortunately the NFU has no such functionality at the moment, so it is also important that end-to-end congestion detection and control approaches be investigated.

4.4.3 An End-to-End Approach

In the case of L-encoder the only current means of gathering information about congestion is to monitor the execution times of operations on that depot. This means that measuring a level of congestion is both inherently an end-point concern and can only be assessed in term of the relative completion times of similar operations on different depots. This of itself is not truly a measure of congestion considering that the difference in execution time may simply be a consequence of underlying hardware resources. However, if we conceptualize congestion control as finding the highest capacity "path" for computation within the network, then these two concepts can be said to have the same practical significance.

The notion of execution time as a congestion measure requires that in order to sample the congestion state of a depot, the application must first allocate it a segment of the compute task. In a case where congestion at a node is high, this will mean a lengthy execution time. A potential response to this is to assign a small task to nodes initially in order to determine their relative congestion and then assign tasks based on those results. This, however, can have drawbacks in the non-congested case where the network cost is being paid for relatively small compute segments resulting in slowstart. In the case of L-encoder those costs could potentially be made back by using the information gained to make a closer to optimal choice in the assignment of tasks to network resources. An examination of a dynamic pipelining scheme with a similar premise was recently undertaken [8]. In this case an examination of a distributed image processing application was performed. Tasks were assigned to depots via a pipelining scheme. In the initialization phase a new task

was only assigned to a depot once it had completed its previously assigned task. Then, based on an analysis of the depot's throughput, additional tasks could be assigned based on observed performance. There is potential for a similar approach within L-encoder. An initial small task could be assigned to each depot and then the number of tasks and the size of the task assigned could increase if adequate performance is still observed. This notion of adequate performance must be experimentally determined and may vary depending on the type of encoding operation being performed. The efficacy of such a scheme and any potential slowstart effect would also require experimental evaluation.

4.4.4 Backoff

The previous approach involving gradual increase in task size and number assignments to depots describes one way that a congestion control algorithm could increase the flow of computation when congestion is not present. Another imporant element of a congestion control scheme is how to respond when congestion is present and scale back flow appropriately. Important considerations in designing the backoff scheme will be what operations to consider and at what schedule they should be used. In the case where multiple transcoding operations are being run on a depot that is appearing congested, an effective response may be to simply not assign additional tasks until all those assigned to the depot complete. Another potential backoff techinque would be to decrease the number of depots currently in use, and assign new tasks only to the best performing subset. This method could potentially allow time for the congested node to clear up. This introduces the problem of when to attempt to scale up the number of depots currently in use and how many to remove during congestion. Assuming that a ranked list of depots is kept based on observed performance the removal of the poorest performing and then the addition of other depots could be managed by a system analogous to network congestion control. In analog to the sliding window approach, the number of active depots in the list could slide and shrink based on observed performance. Experiments would need to be conducted to determine which, if any potential sliding window style algorithm would yield desirable results.

The importance of congestion control in a network computational service such as L-encoder cannot be understated. The ideas presented here are designed to lay inroads for

the development of such a client side congestion control approach. When combined with potential scheduler developments in terms of fault tolerance, this client scheduler, it is conjectured, should be able to achieve parallel gains in a wide area testbed system.

Chapter 5

Future Research

The L-encoder NFU was intended as a proof-of-concept for a transcoding service within the logistical network0. The initial results from parallel testing show potential for useful service with performance gains for potential users. It will require additional research effort, however, to adapt L-encoder to the point where it is viable for deployment in a testbed system. There are several interesting applications which could motivate this future development.

5.1 Simple Offload Service

The current test results for L-encoder show great potential for parallel processing gains especially in the realm of HDTV transcoding. The archival of HDTV data for posterity will become increasingly important as more and more broadcasts make the switch to the high resolution formats offered by this standard. The large storage requirements for such files means that an archival service of raw HDTV data seems impractical when compared to a transcoded file of similar video and audio quality. The use of an L-encoder like service could provide the capability to make such archival possible using a heterogeneous hardware system that is available to multiple archival projects. Additionally such a service provided by a content distributor to its clients could reduce the storage burden on PVRs allowing for the archival of previously recorded content without sacrificing the ability to make new recordings due to the large file sizes involved.

5.2 IBPvoHD

IBPvo is an experimental service of LoCI. It combines a video capture service with an online reservation scheme creating a network available video recording service [11]. Its operation is depicted in the figure 6.10. The current IBPvo service operates on conventional television capture. The proposed upgrade to IBPvoHD would involve the capture of HDTV content via a firewire connection from a cable box. In this case there would be an intermediate step between the recording of content and its final upload to the network in which the HDTV content would be transcoded to another format. This format could be general or user defined. In this case the IBPvo scheme would not only be updated to allow for the capture of HDTV source but also it would allow for the transcoding program to potentially create output that could be used in portable media devices or consumer electronic devices for playback. In this scheme the user would be able to get the full benefits of both a PVR and a network transcoding service without having to expend any significant computational resources and having to download a file much smaller than the original captured HDTV content.

5.3 Streaming Media Transcoding

The ability to create a stream of transcoded data from an input source and serve it to multiple recipients is an application domain with many interesting new developments. The previously mentioned Verizon Vcast Mobile TV service currently serves transcoded streams of live TV broadcasts at decreased bitrate and framerate. Such a streaming service for internet enabled mobile devices such as the Linux based Nokia n800 and n810 would provide an interesting research testbed in the area of mobile video delivery.

Chapter 6

Conclusion

The development of L-encoder was motivated by the prevalence of transcoding as a heavy-weight operation with potential for network offload and parallelization. The NFU paradigm provided an available, best-effort platform on which to base such a service, and IBP provided a storage resource to facilitate computation within the network. The comparison runs of parallel execution between mencoder and L-encoder show that the speedup gains made by L-encoder are comparable to those in the parallel mencoder case without the necessity of private resources or an account scheme. The deployment challenges for L-encoder are many, but with the potential utility of such as scheme it is conceivable that continued research into this problem could yield a large scale system capable of serving the transcoding needs of a large user community with diverse applications and areas of interest.

Appendix

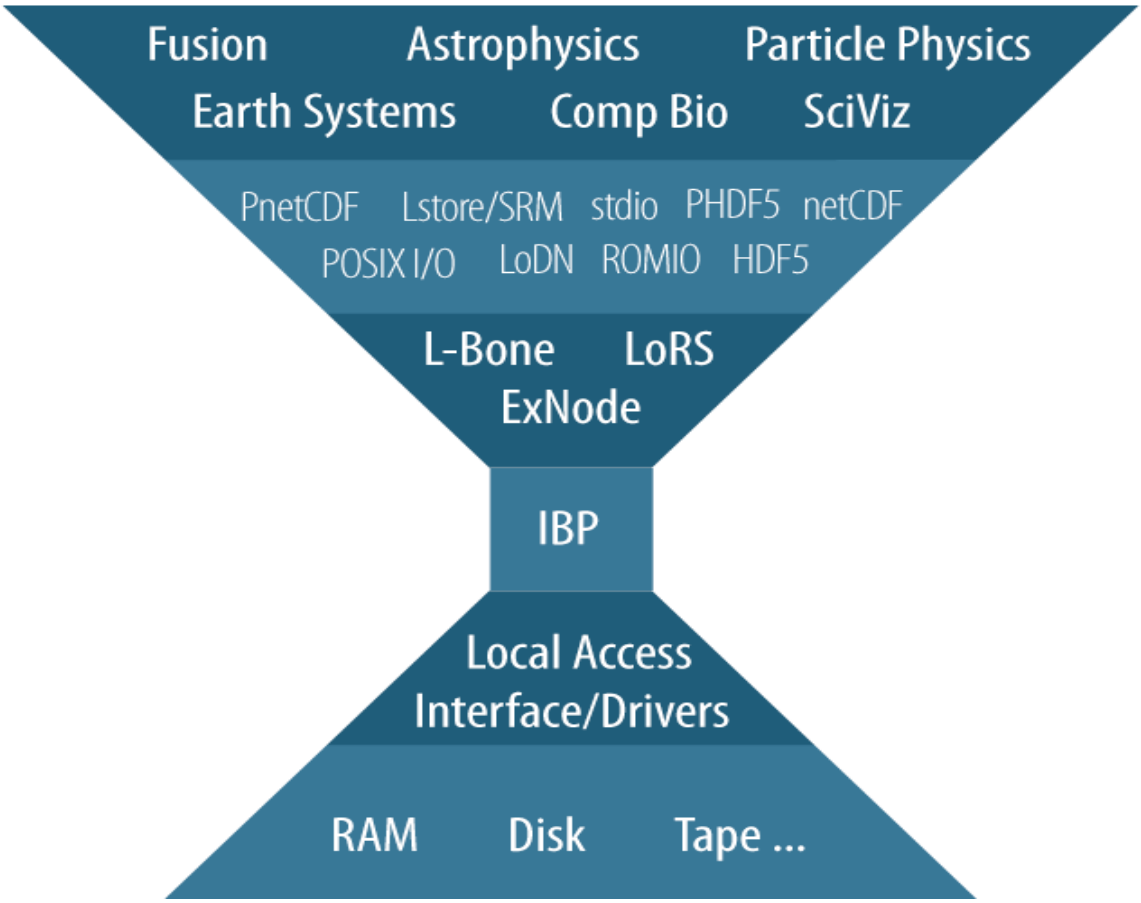

Figure 6.1: The logistical networking storage stack with IBP at its center.

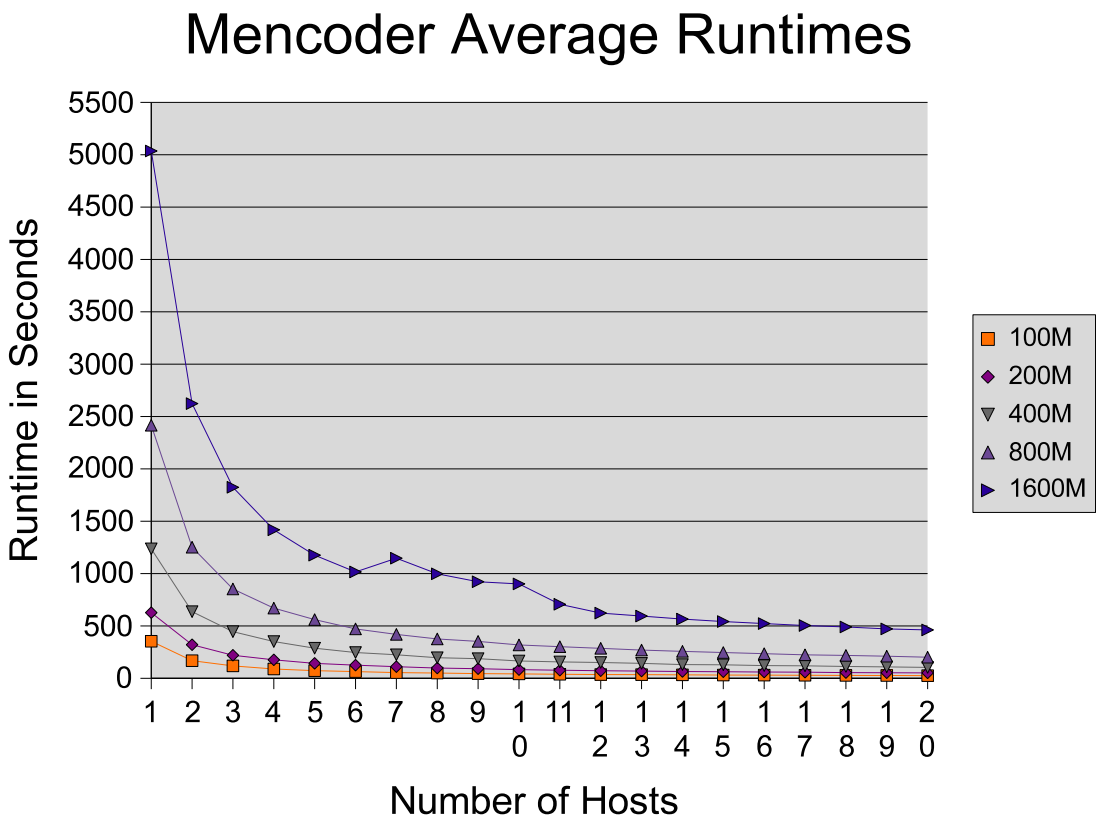

Figure 6.2: This graph shows the average runtime of mencoder on the 5 test files with the file divided into the given number of segments.

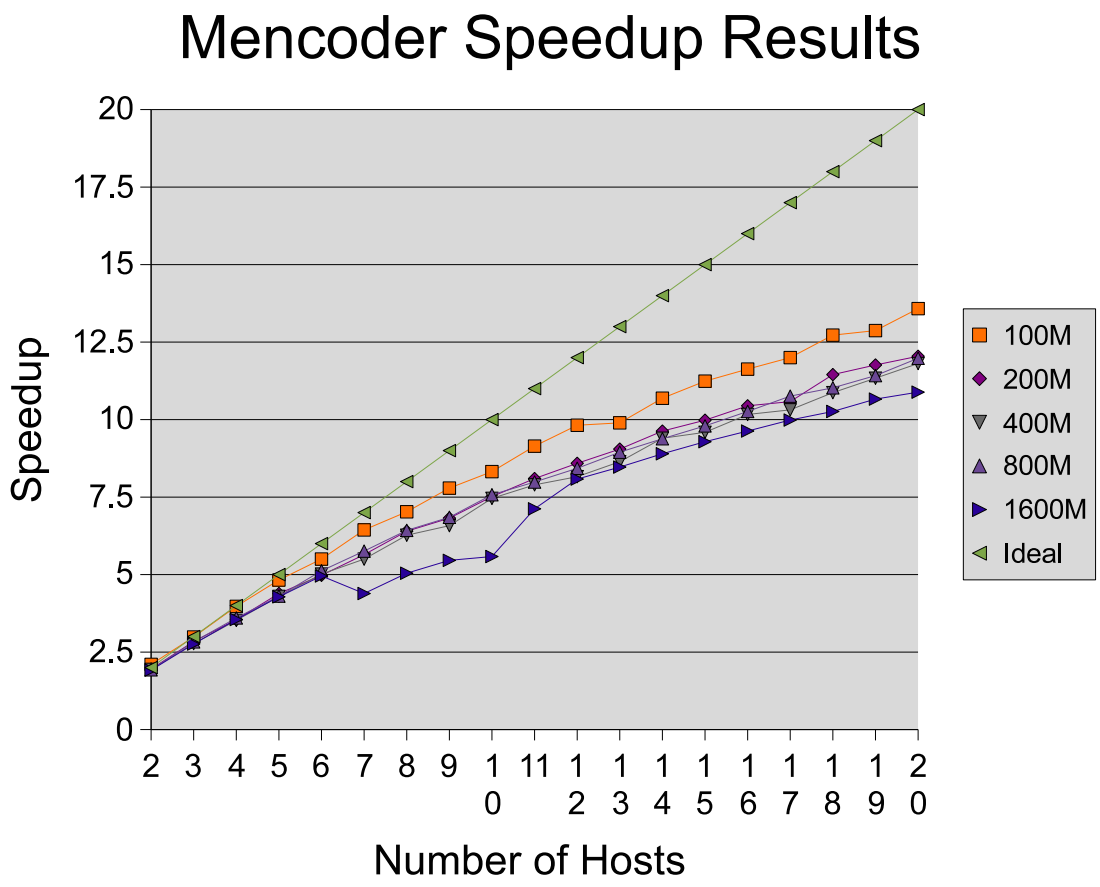

Figure 6.3: This graph shows the speedup of mencoder on each of the 5 files with the given number of segments.

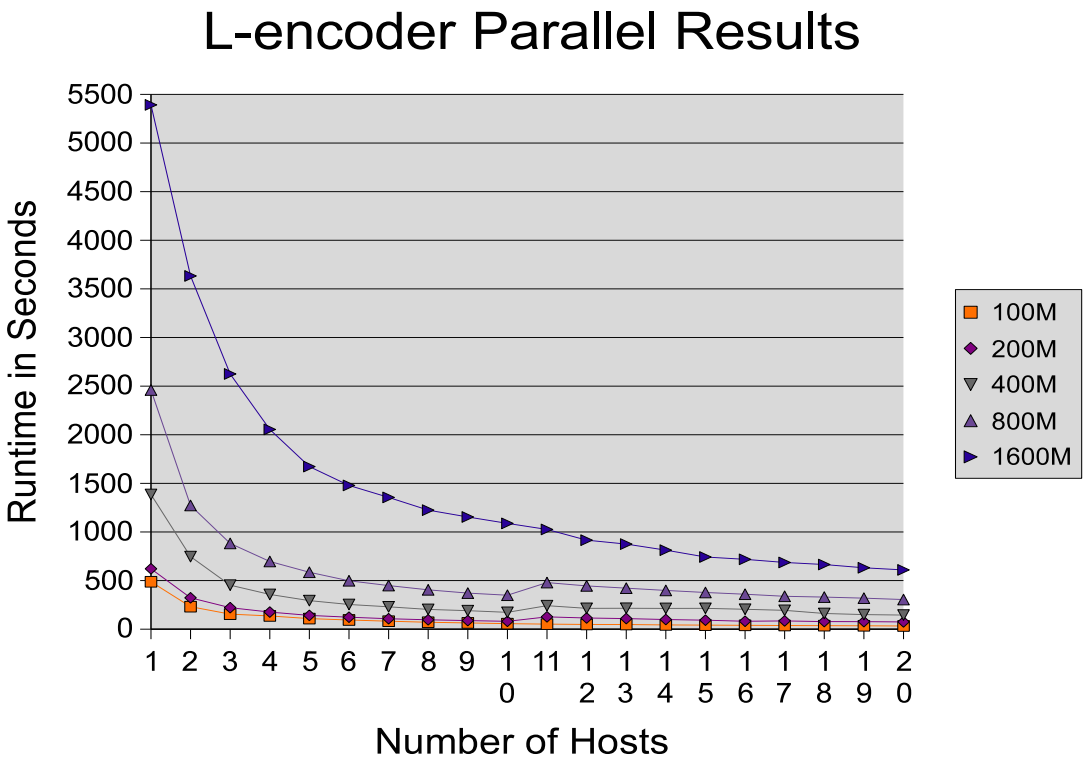

Figure 6.4: This graph shows the average runtime of L-encoder on the 5 test files with the file divided into the given number of segments.

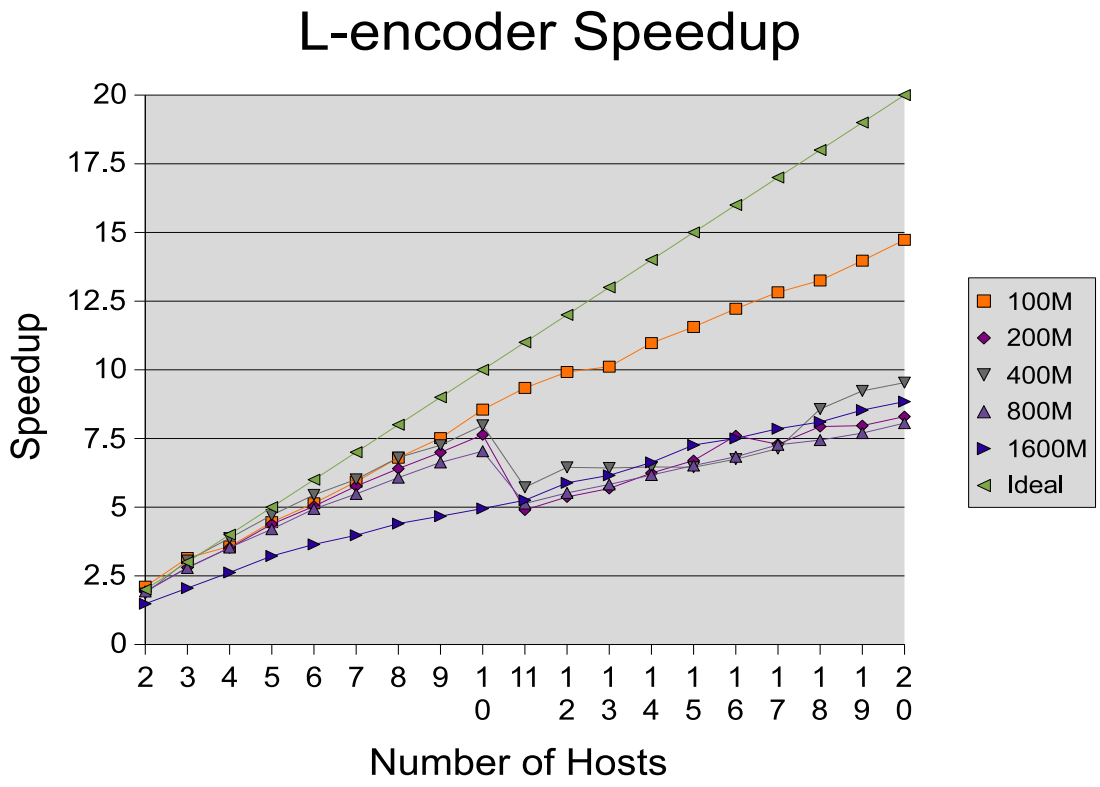

Figure 6.5: This graph shows the speedup of L-encoder on the 5 test files with the file divided into the given number of segments.

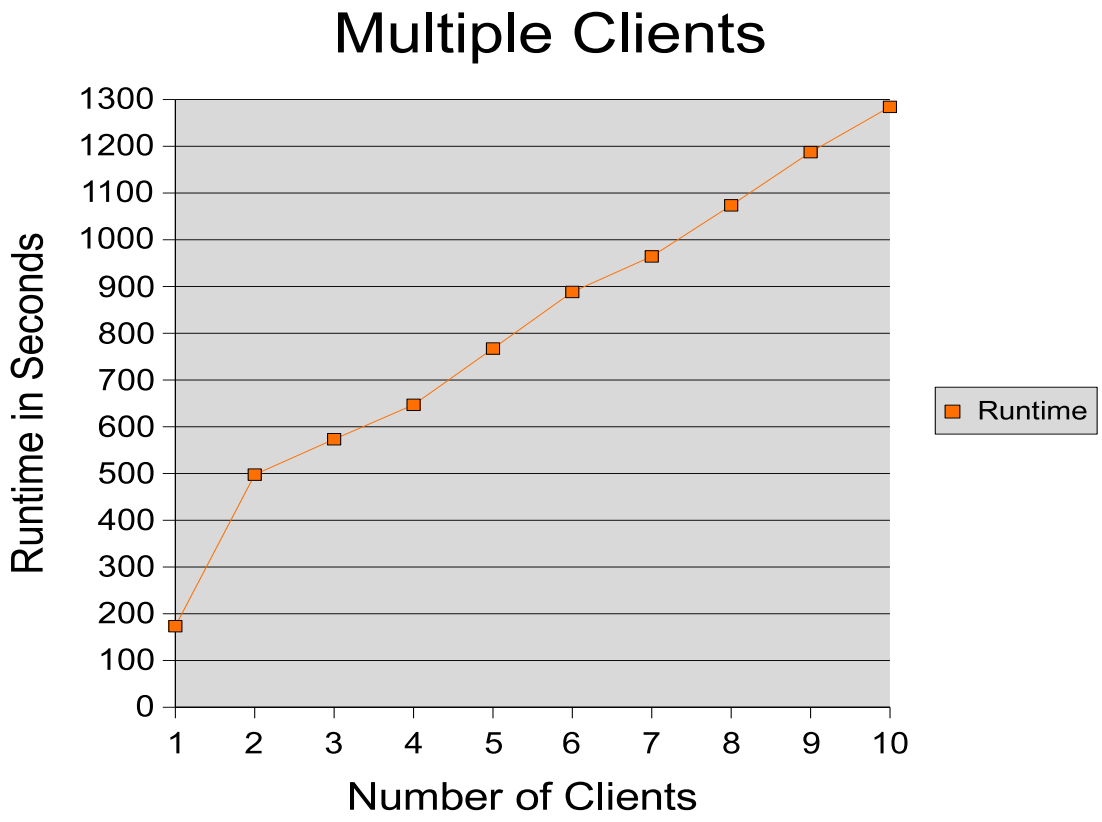

Figure 6.6: This graph runtimes of L-encoder clients when n clients were run simultaneously

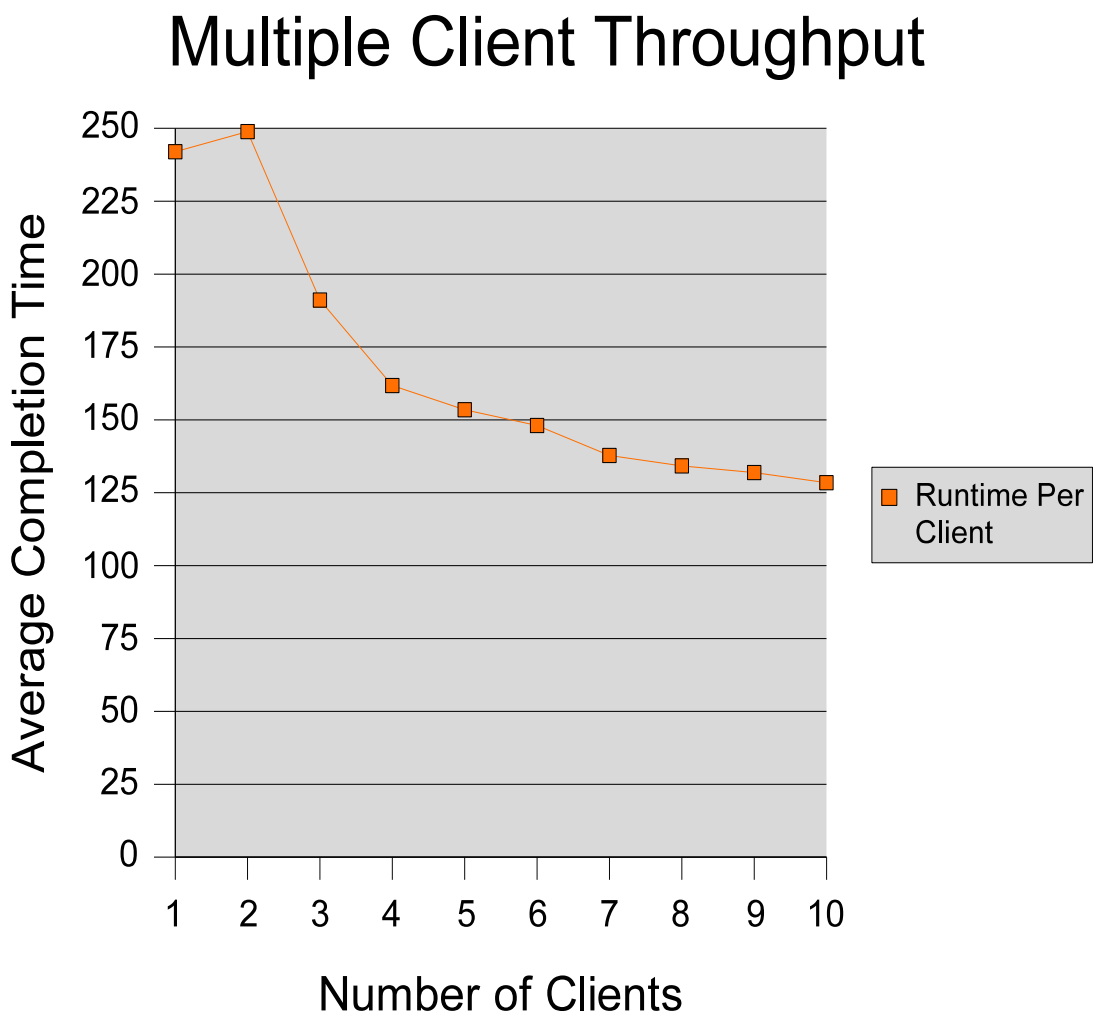

Figure 6.7: This graph shows the average completion time from the multiple client test in chapter 3.

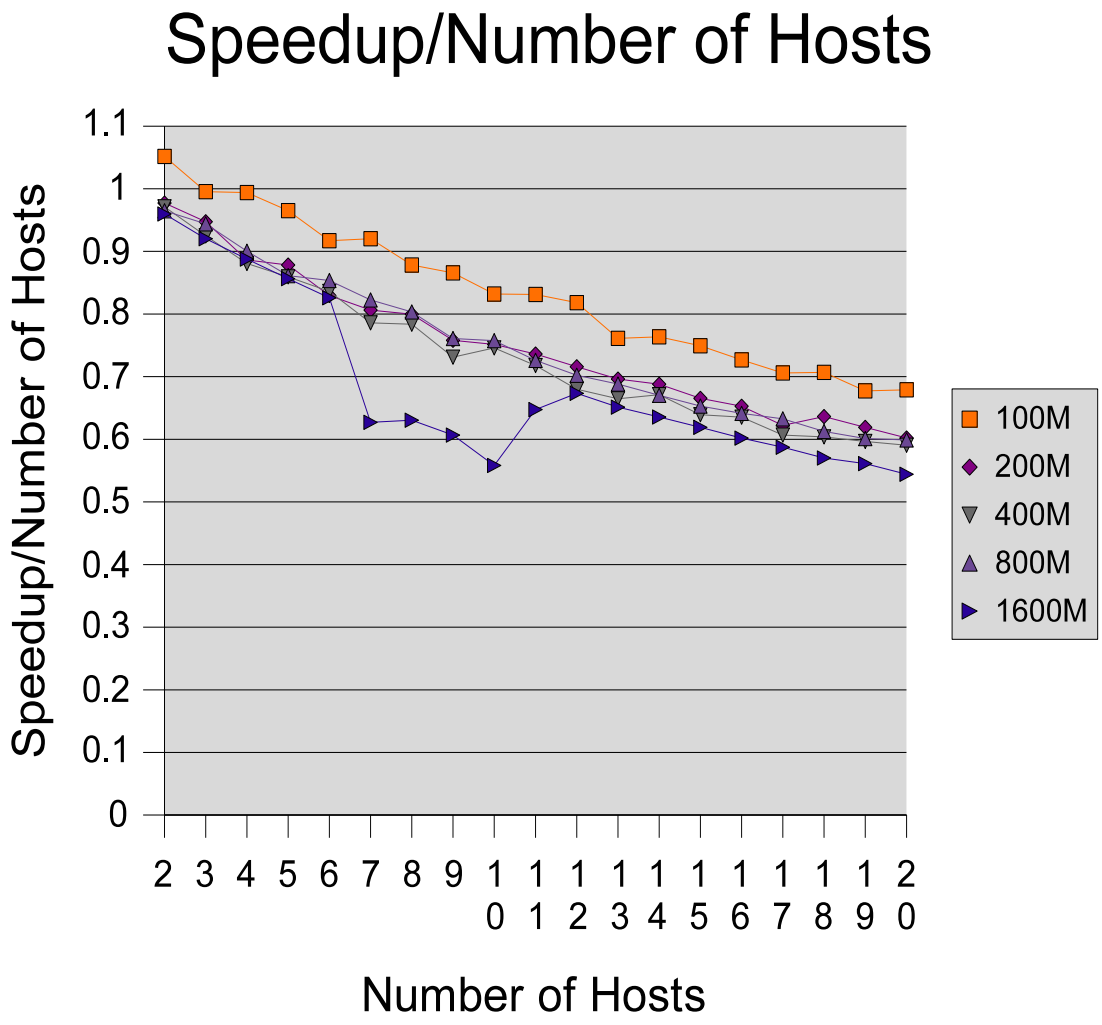

Figure 6.8: This graph shows the ratio of mencoder speedup to number of hosts

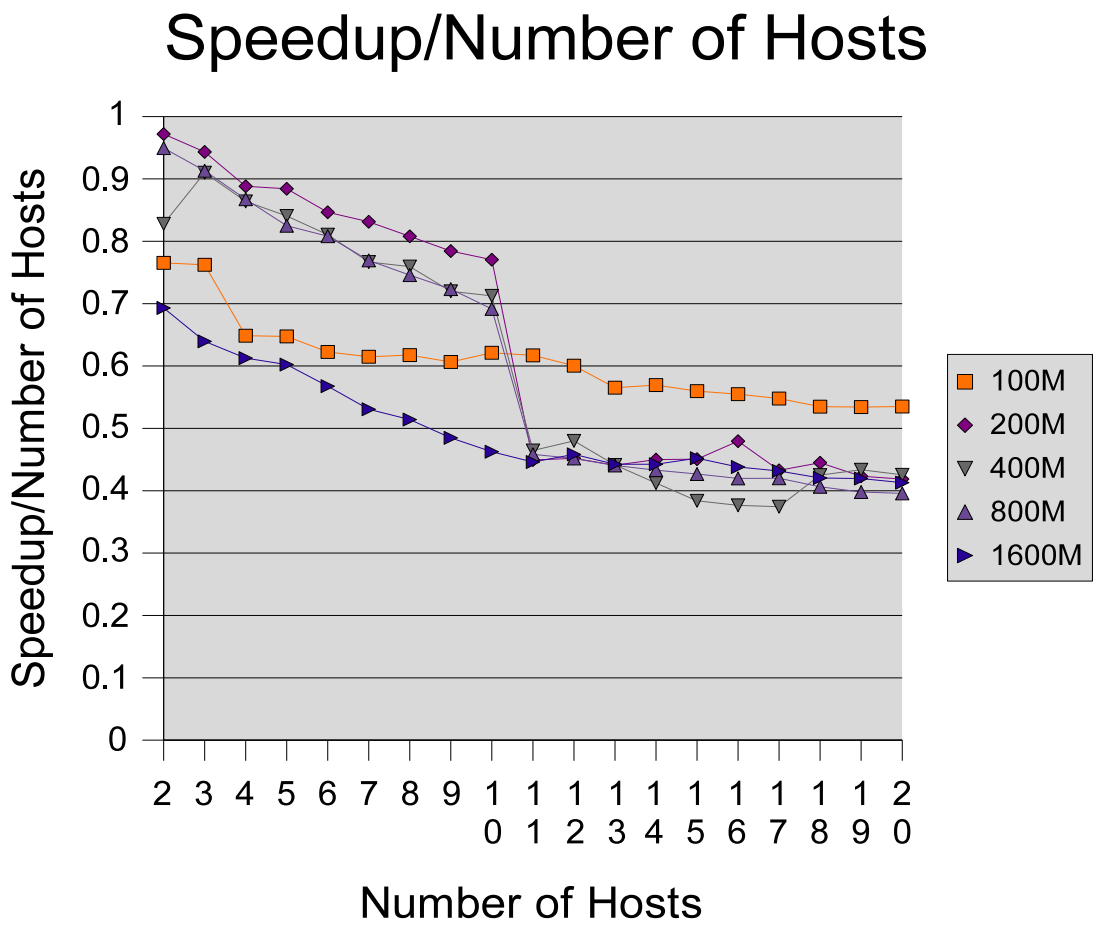

Figure 6.9: This graph shows the ratio of L-encoder speedup to number of hosts

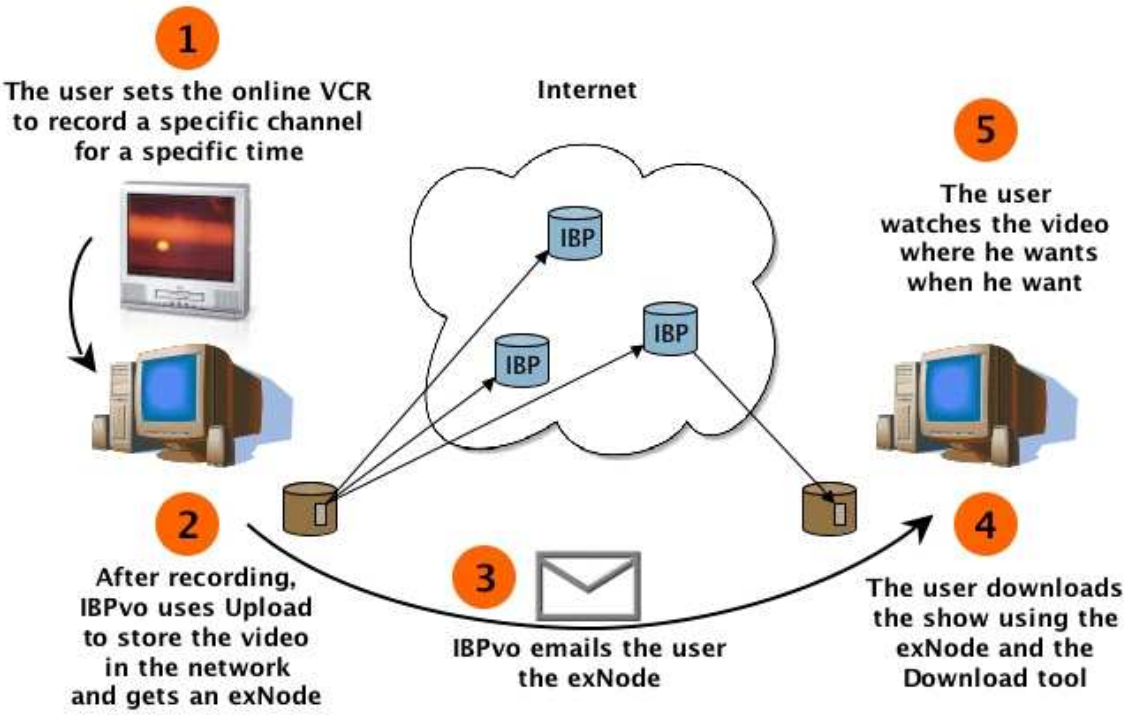

Figure 6.10: This figure depicts the operation of IBPvo.

Bibliography

Bibliography

[1] http://www.xvid.org/.

[2] http://products.vzw.com/, July 2008.

[3] http://www.mplayerhq.hu, July 2008.

[4] http://www.ti.com/, July 2008.

[5] Graham Fagg Terry Moore James S. Plank Martin Swany Rich Wolski Alessandro Bassi, Micah Beck. The internet backplane protocol: A study in resource sharing. In *Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, Berlin, Germany, May 2002.

[6] Jian Huang Huadong Liu, Micah Beck. Dynamic co-scheduling of distributed computation and replication. In *Proc. of IEEE/ACM CCGrid*, Singapore, May 2006.

[7] Jian Huang Micah Beck Qishi Wu Terry Moore Jinzhu Gao, Huadong Liu and James Kohl. Time-critical distributed visualization with fault tolerance. In *Proc. of Eurographics Symposium on Parallel Graphics and Visualization*, Crete, Greece, April 2008.

[8] Huadong Liu. *Scalable, Data-Intensive Network Computation.* PhD thesis, University of Tennessee, Knoxville, TN USA, April 2008.

[9] James S. Plank Micah Beck, Terry Moore. An end-to-end approach to globally scalable programmable networking. In *Workshop on Future Directions in Network Architecture (FDNA'03)*, Karlsruhe, Germany, August 2003.

[10] James S. Plank Martin Swany Micah Beck, Terry Moore. Logistical networking: Sharing more than the wires. *Active Middleware Services*, 2000.

[11] Terry Moore Micah Beck. Logistical networking for digital video on internet2. In *Fall 2003 Internet2 Member Meeting*, Indianapolis, IN, USA, October 2003.

Vita

Harold Gonzales was born in Ogden Utah on March 6, 1984. He attended the South Carolina Governor's School for Science and Mathematics before coming to the University of Tennessee to begin undergraduate work in Computer science in 2002. In 2006 he was awarded a Bachelor of Science degree in Computer Science and decided to stay in Knoxville to begin his graduate study. In 2007 he joined LoCI as a graduate research assistant. He received his MS in computer science from the University of Tennessee in Summer of 2008. He plans to attend the University of Colorado at Boulder to pursue a Doctorate in Computer Science.