Distributed Image Processing for Automated Lecture Capture Post-Production

Craig Thompson, Christopher Brooks, Jim Greer
Department of Computer Science, University of Saskatchewan
110 Science Place
Saskatoon, SK, Canada
{cdt830 , cab938}@mail.usask.ca, greer@cs.usask.ca

ABSTRACT

This paper describes a low-cost distributed computing approach for the post-processing of videos. It has been implemented to support the creation of composite videos from various video sources, and supports image manipulation on individual frames of the video. The principle deployment scenario for this work has been to support the automatic creation of output video based on traditional face-to-face lectures. Experimental results suggest that this approach can lead to significant speedups with a reduction in operating costs by leveraging existing underutilized equipment.

1. INTRODUCTION

1.1 Motivation

Creating novel and engaging online educational content is a time consuming and expensive task. Instructors have to be content experts, graphic designers, instructional designers, and pedagogy experts. Large educational institutions such as universities may have the demand for hundreds of online (which includes fully online as well as blended learning) courses to be offered in a given term. Many institutions[1] provide distance education either partially or exclusively through high cost television networks, where significant resources are devoted to media technicians and directors to achieve studio-quality results. Despite the abundance of spare computing resources on campuses, dedicated equipment for capture and broadcast is often purchased, and the number of media-rich distance education courses is typically minimal.

To reduce the costs of offering online courses and to enhance student learning experiences in traditional face-to-face courses we are in the process of building and deploying a multi-camera lecture capture system. This system, called Recollect[2], provides for automatic template-driven post-production of videos based on a variable number of video streams captured in the classroom. These streams are decomposed into images, manipulated (cropped, merged, etc.), and recombined into single device-targeted output video with minimal technician interaction. Instead of competing with selective but high quality television recording of lectures, we aim to support wide scale deployment of online content from lower quality fixed cameras in the classroom.

Post-processing of video, even if the processing is automatic, can be a time consuming task. In our initial deployments we have focused on creating both high resolution and low resolution videos for the web in Flash Video, as well as H.264 videos targeted at mobile devices (such as iPods). We do this using (usually) three video streams; an NTSC source of the professor, and NTSC source of the audience, and a VGA source from the digital projector. Keeping processing costs under control is of principle concern both from a research and production stand point. To do this, we have begun to leverage the mass of unused computing resources within the institution instead of buying dedicated hardware. We see volunteer computing approaches to be particularly salient in higher education, where masses of modern computer labs sit mostly idle during non-working hours.

1.2 Technical Approach

Frame-based processing of videos is a naive and slow technique for video manipulation in comparison to in-stream approaches, but it allows for codec independent toolkits to be used for image effects. This increases the variety and availability of post-processing actions that can be applied to videos, and is extremely useful in rapidly deploying new device-specific output templates. The main cost of frame-based processing in addition to high performance hardware is the extreme amount of time required to process each frame, as well as the time required to decompose and recompose the video from input and output sources respectively. In our current implementation, post-production takes roughly ten times the length of the input videos being processed using a reasonable desktop PC, and roughly three times the length of the input video on a high performance server[3].

Volunteer computing is not new; two of the best know volunteer computing projects are SETI@home, and Folding@home, a radio frequency analysis project and protean folding project respectively. By making use of computers volunteered by individuals, or in our case, available institutional resources, researchers can make use of tremendous computing power at little or no cost [4]. Our general approach (detailed in section 2) has been to devote banks of machines in undergraduate laboratories to processing pieces of video when they are idle. Other approaches we are investigating include heterogeneous machines (e.g. a

[1] In [1] we conducted a survey to identify the attitudes of different stakeholders in higher education with respect to course casting systems. A significant number of respondents indicated that televised courses were the principle method of media-rich distance education.

[2] http://www.recollect.ca

[3] Rates for the desktop are for Intel Core 2 Duo machines running at 2.5 GHz under vmware virtualization (explained later), and for the dedicated server are for dual Intel dual core Xeons running at 2.4 GHz.

978-0-7695-3454-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ISM.2008.44

mixture of laboratory machines and dedicated servers) as well as non-idle machines that may have excess capacity.

2. EXPERIMENTAL DESIGN

Our initial experimentations in this project have involved a swarm of eight modern desktop PCs in an undergraduate computing laboratory under low load. Each computer had a lightweight virtual machine installed in it running in the background polling a centralized server for video to process. An important consideration in our experiments has been to limit the amount of network traffic on wide area networks (at least across institution subnets). In particular, we wish to minimize the network requirements on the server machine, as it could be dealing with potentially hundreds of client computers, but may not be connected to a high speed network (it may be located in the classroom). To do this, we have used the bittorrent protocol to transfer data from the capture server to the client computers. Bittorrent has been shown to make excellent use of peer upload bandwidth, thus reducing the bandwidth load on the server greatly. Furthermore, the bittorrent protocol scales well, and the amount of data uploaded by the first seed is nearly independent of swarm size [3]. For all of our tests, we ensured the machines had no initial load, and that they were located on the same hardware switch. Future investigations will examine how load averages (e.g. non heterogeneous machines) and network topologies effect processing performance.

Once distributed, lecture streams are broken up longitudinally into processing pieces, where each piece of work represents one minute worth of recorded lecture. Clients request work from the capture server, which responds with a torrent file for the whole of the lecture. Once a client has completed downloading the lecture it will request instructions on the time piece it should process and details about how that piece should be processed.

We intentionally make clients download all of the raw data for the lecture before they begin to process the video. This has several advantages; first, given that we are using a student lab for our processing, a machine could be removed from the client swarm at any given time. If all clients have the full data set, it is trivial to reassign a job to another client, as just the instructions need to be transmitted and not the data. We use a first in first out method of job scheduling, and re-queue unfinished pieces to the work queue. Starvation is not an issue, as all pieces must be completed before the lecture is completed (thus priorities of pieces processed doesn't affect the final output).

By distributing the data before processing, rather than as the instructions are assigned, we can easily adapt to the rates at which clients can receive and process data. Although with a bittorrent-based distribution method the server does send more than one full copy of the data set, we have the option to limit the server to seed one copy only if wide area network resources were especially limited.

Once a client computer has completed downloading the data set it requests details about the work piece to be processed. While an in-depth discussion of how the video is processed is out of the scope of this work, the general steps of the process are:

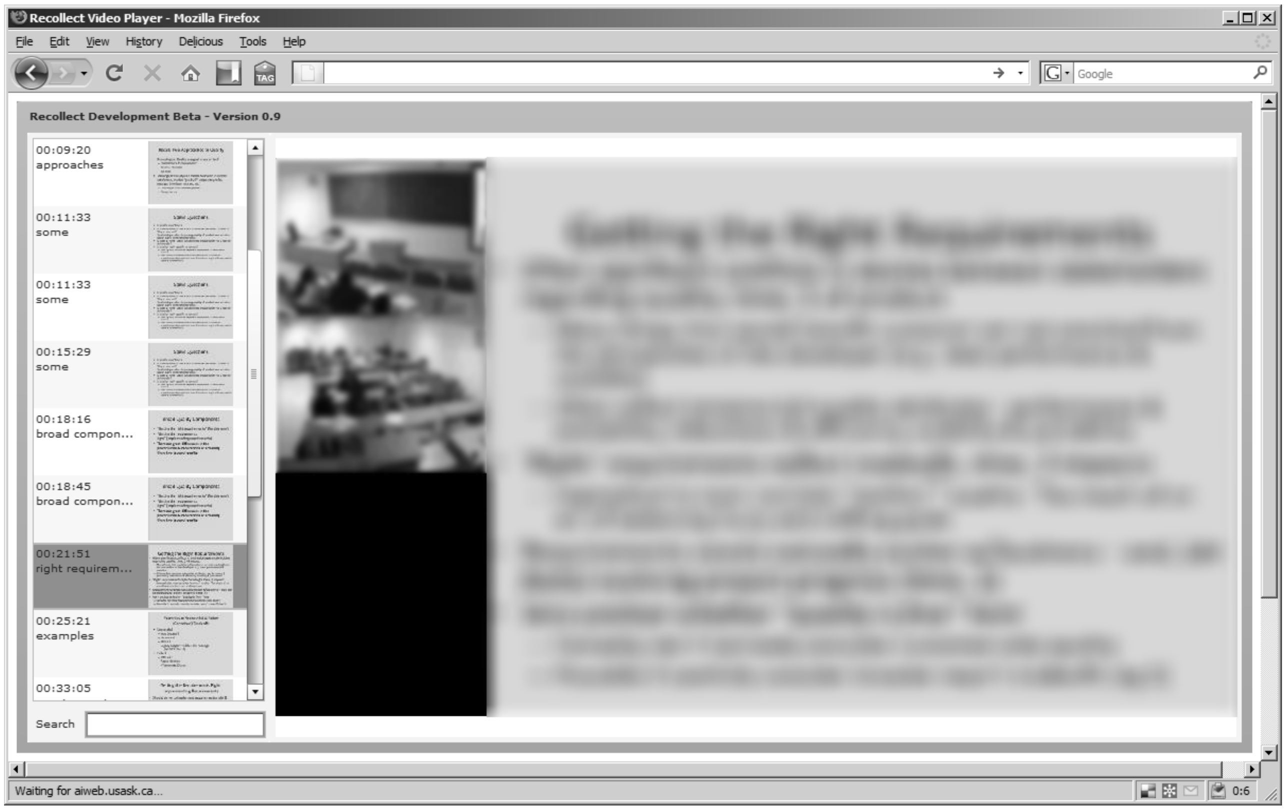
Recollect Video Player - Mozilla Firefox
File Edit View History Delicious Tools Help
Google
Recollect Development Beta - Version 0.9
00:09:20 approaches
00:11:33 some
00:11:33 some
00:15:29 some
00:18:16 broad compon...
00:18:45 broad compon...
00:21:51 right requirem...
00:25:21 examples
00:33:05
Search
Waiting for aiweb.usask.ca...
0:6

Figure 1 An example of the Recollect playback component, showing the composition of three videos into one stream on the right hand size. Video content has been blurred for anonymity purposes in publication.

1. Client decomposes a time slice from each video stream into a set of image files.
2. Each frame undergoes a series of image manipulations (e.g. cropping) using the ImageMagick[4] toolkit, and then is combined across input streams into a single mosaic representing a single output frame, as shown in figure 1.
3. Final frames are combined into a video segment (depending on the intended output format), and posted via a REST-based web service to the capture server.

We use a direct host-to-host transmission method for returning the completed job chunks, because there is no need for the other peers to have the finishd work, so bittorrent is not useful here. Once the capture server receives each job piece, it combines them into a final output video, and publishes them to various institutional websites.

3. EXPERIMENTAL RESULTS

Figure 2 displays the average load average versus time for the eight client computers used. The load average[5] is the average number of runnable processes in the processor queue over a one minute period. Data processing is broken up into three phases; a data acquisition phase, a job processing phase, and a result transmission phase. The black line in figure 1 shows the mean load time over all clients, and the shaded regions above and below this line show one standard deviation of difference.

During the first fifteen minutes a single server creates a torrent of the original video files and seeds this dataset to the clients, resulting in minimal load averages to clients. As clients finish downloading the data file they continue to seed while they execute a number of scripts to decompose the video into frames, then perform manipulations on the video. Load averages are initially high because of a number of factors; in particular, clients continue to actively seed the data set while they process the video. We hypothesis that the large standard deviation seen at the beginning of the job processing phase comes about because some clients are being taxed both with job processing and torrent seeing, while others have a relatively low load and are just processing a single job.

Each client received roughly ten jobs to process, and each job took roughly ten minutes to complete. The average load of the machines was relatively low throughout, as some portions of the job processing (e.g. decomposition into images) are more taxing than others, but client jobs became staggered (because of the non-uniform way bittorrent distributes data - a discussion of bittorrent protocol issues can be found in [3]).

Processing an 80 minute lecture using a dedicated high performance server takes roughly three hours. By distributing the jobs across existing laboratory infrastructure we are able to reduce the processing time to roughly two hours. The results of our initial study suggest two important results:

1. A large number of clients are not required to improve processing time - even a small number of relatively low powered machines can reduce processing costs by a third.
2. Minimal amount of time is spent in data transfer - in our situation data transfer accounted for less than 25% of the total video processing time. More efficient transfer protocols (such as multicast, if it is enabled) will reduce this overhead further. At the moment, videos were limited in bitrate, and were captured at a rate of roughly 3.3 megabits per second. The total data set used was thus 1.8 gigabytes. We anticipate that the increase in processing speed will lead to higher fidelity video being captured in the future.

With the bulk of time spent being done in the job processing

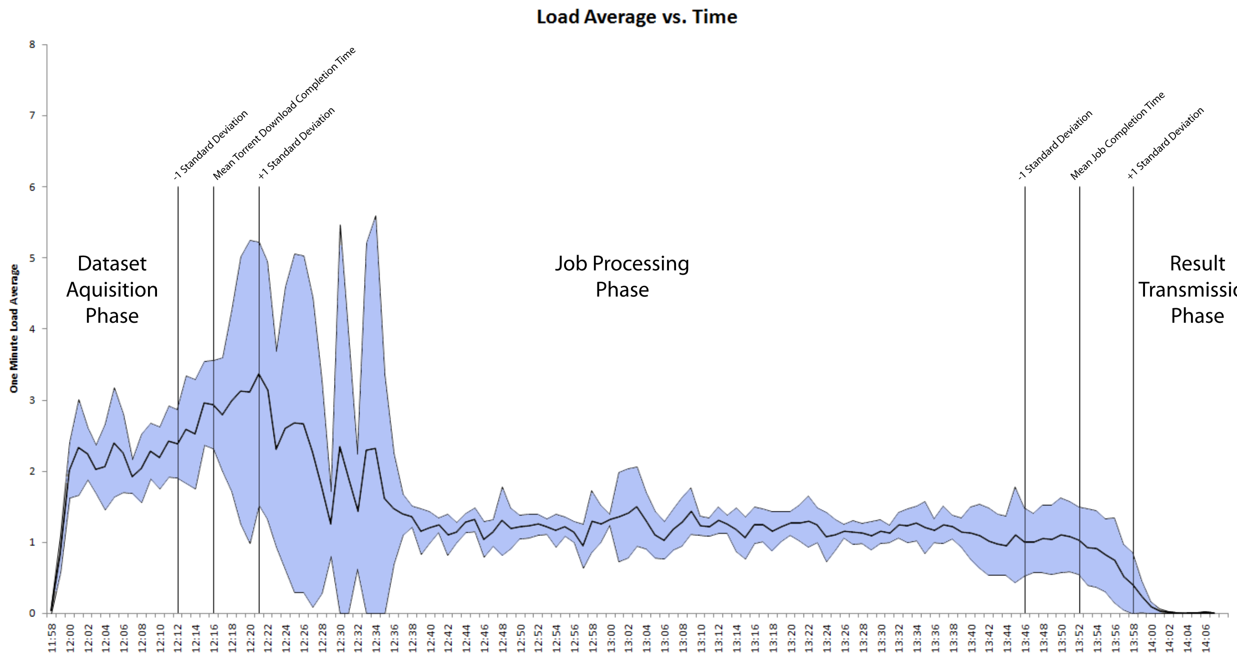

Figure 2 One Minute Load Average (using *uptime*) versus Time

phase, scaling the speed at which a job can be processed (either through hardware speedup or through faster implementation) is still critical in increasing overall performance. However, the distribution of jobs to multiple machines can result in a low cost (no special processing hardware needed) high performance method of video processing.

4. CONCLUSIONS

A significant assumption of our experiment was the relative homogeneity of the distributed processing clients. The computing facilities used for this project consisted of an undergraduate computing laboratory under relatively low load. As such, the computers were in a fairly reliable state, were well maintained, had plenty of spare resources, and had similar hardware architectures. This situation is not typical of the spare resources in higher education institutions, and various times and student workloads impact the amount of resources that are available for volunteer computing. One of our more immediate future goals is to deploy this system in a broader and less controlled environment by including unused computing cycles in research labs, staff desktop machines, and the low-load servers on campus. To accommodate the range of power of client machines we anticipate that we will need to make the job size small enough that even the slowest computers can contribute a piece of data before more powerful machines complete all the other chunks. By distributing the full data set at once, we expect there to be little overhead caused by using very small job sizes. It should be noted that in addition to homogeneity of computing power, we also had a relatively unburdened local network with all of the client computers on a shared switch. Typical higher education institutions often have more complicated network topologies that may require more agile forms of data transfer.

As the number of resources available increases we can increase the number of tasks to meet the available capacity. Perhaps most importantly, we can increase the availability of the lecture capture software itself to fit the demand of instructors on campus. Once we have enough capacity to handle all the lectures we are interested in processing, we can expand the range of tasks performed by each client. For instance, we have a number of optional time consuming server-side tasks involving image analysis on a frame by frame basis. These tasks include movement detection (for automatic cropping around the instructor), event detection (for forming a table of contents of the video), and image detection (for finding special sub-images of interest, such as creative commons copyright logos which could be used for automatic metadata generation).

An interesting result that has come out of studies with how students use this technology (e.g. [1]) is that some students have privacy concerns with appearing on camera. We are investigating the feasibility of mapping positions within a lecture hall, and have students indicate if they wish to remain anonymous. If a student wishes to be anonymous, selective transformations (e.g. blur, pixelization, avatars, etc) could be overlaid on the image to provide increased privacy. Rapid prototyping and deployment of these features is possible because of the frame-based approach implemented.

There are many other interesting manipulations and analyses that can be performed on the screen capture video. One aspect currently under development is to detect transitions from one lecture slide to the next. Knowing the times at which the slides changed in a lecture would enable us to tag the timeline of a video, much like chapter tags on a DVD, allowing students to skip back and forth through the video as they watch. While there has been some work done in this field already [2], we have begun using a supervised learning data mining approach which requires large sets of image characteristics. Generating training data in a timely manner from raw video streams is a consistent issue, and we are intending to use this architecture to help deal with this problem.

Our approach utilizes a central server for the recombining of videos, which is a potential processing bottleneck. This bottleneck is worsened when using multiple pass encoders (e.g. 2 pass H.264 encoding). We are investigating the usefulness of intermediate raw video formats that can be converted and compressed quickly to help minimize this effect. We anticipate that the addition of a small number of high powered client machines to the swarm will eventual require us to provide heterogeneous jobs, and some of these jobs (such as 2 pass encoding of recombined videos) will be allotted to machines with specific hardware characteristics.

5. ACKNOWLEDGMENTS

This work was funded in part by the LORNET NSERC network and the College of Graduate Studies at the University of Saskatchewan. Special thanks goes to Stephen Damm and Greg Logan who developed several of the core components for image manipulation.

6. REFERENCES

[1] Tan, I., Brooks, C., and Trottier, D. (2007) Attitudes Towards Recorded Lectures. Presented at the First International Conference of the new Canadian Network for Innovation in Education (CNIE). Banff, Canada.

[2] Dickson, P., Richards, W. A., and Hanson, A. (2006) Automatic Capture of Significant Points in a Computer Based Presentation. In proceedings of the Eight IEEE International Symposium on Multimedia (ISM '06).

[3] Bharambe, A. R. Herley, C. Padmanabhan, V. N. (2006) Analyzing and Improving a BitTorrent Networks Performance Mechanisms. INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings.

[4] Anderson, D. P. and Fedak, G. 2006. The Computational and Storage Potential of Volunteer Computing. In Proceedings of the Sixth IEEE international Symposium on Cluster Computing and the Grid (May 16 - 19, 2006). CCGRID. IEEE Computer Society, Washington, DC, 73-80. DOI= http://dx.doi.org/10.1109/CCGRID.2006.101