Distributed Feature Extraction Using Cloud Computing Resources

A Thesis Presented to the Academic Faculty

By

Steven Dalton

A Proposal Presented to the Academic Faculty
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science with Research Option

School of Computer Science
College of Computer Science
Georgia Institute of Technology
Atlanta, Georgia

Acknowledgments

I wish to thank Professor Umakishore Ramachandran, Dr. Rajnish Kumar, Junsuk Shin and the other members of the Embedded Pervasive Lab for getting me excited about research.

Table of Contents

INTRODUCTION

In the past 10 years massive surveillance systems have been deployed over larger areas because of the increased threat level perceived by governments around the world and a decrease in the costs of wireless cameras. In 2006 the United States Department of Homeland Security dedicated $2 billion dollars to state and local governments in order to create networks of surveillance cameras to watch over the public in the streets, shopping centers and airports[1]. The United States is not alone in its interest to deploy these large scale surveillance systems to reduce the threat of terrorist and criminal activity. In London there are more than 200,000 cameras deployed within the city and more than 4 million throughout the entire country[2]. These large camera systems have been cited as being largely ineffective in deterring terrorists' attacks or crime in the areas in which they are deployed. The discrepancy between surveillance and perceived security has been attributed to the amount of personnel required to monitor these large surveillance networks 24 hours a day and the deluge of data presented to the human operators for analysis. For human operators to effectively monitor these massive systems requires the use of computer monitoring systems which can identify which camera streams are interesting based on a set of predefined characteristics.

Computer Monitoring Systems

Computer monitoring systems allow for the automated processing of a large number of data streams collected from a wide variety of data collection devices. The data streams are processed by a software system that extracts the presence of interesting features, such as the presence of a face or a weapon, within each frame. The frames that exhibit this feature can then be categorized according to the threat level, the location or the time in order to create a

priority hierarchy to ensure that the data streams which exhibit the highest degree of time-sensitive information are processed quickly. It has been shown that this data prioritization provides a level of situational awareness in the surveillance system which can filter the overwhelming amount of captured data into a manageable "interesting" set[3]. In order for the computer monitoring systems to be of practical use they must rapidly detect, prioritize and deliver the analyzed data in near real-time. In massive surveillance systems composed of hundreds of thousands of cameras this real-time dependency requires a massive amount of computational resources for data processing, a large and efficient data storage system, a reliable high bandwidth communication network, and the ability to scale according to an increase in either the number of cameras within the network or the number of interesting data streams.

As a example consider the 2009 presidential inauguration in which more than 5,265 cameras were used to monitor a crowd of approximately 1.5 million people[4]. The available network bandwidth limited the number of cameras which could transmit data to the control center and the increase in the amount of data collected for processing presented serious issues which needed to be addressed to secure the area around the National Mall. While the data centers that are used to support these surveillance systems are vast they are usually static in the sense that they cannot dynamically grow to adapt to the sudden influx of data submitted for processing. This static processing resource is a computational form of the same problem that was observed with human operators, the saturation of data degrades the ability to effectively monitor a large area. One solution is to rent additional processing power from utility computing infrastructures that contain a large number of computational resources.

Utility Computing

Utility computing is a model for the sharing of computational resources based on the consumer needs. Instead of purchasing large infrastructures to support a temporary need, such as the 2009 inauguration, the computational resources can be rented in a fashion that is comparable to other utility services, such as electricity or water. The consumer in the utility computing scenario may request the resources as needed to accomplish the task. This provides a way to optimize the resources allocated to meet the task specific requirements. The most notable realization of utility computing in the recent years has been grid computing[5, 6]. The main limitation of grid computing has been the batch oriented processing model that queues tasks that are submitted by all users until the necessary computational resources can be dedicated to accomplish the task. This model does not perform well in situations that require a sense of real-time responsiveness. The promise of expansive computational resources has lead to some modifications to the grid computing system to allow real-time responsiveness[7]. The use of grid computing as a method to deliver computational power on demand has experienced only minimal success because it has been shown that the grid model of computational utility is difficult to integrate into existing systems[8].

The most recent extension to utility computing, that has received a lot of hype as the long awaited solution to many of the utility computing short-comings in the past, has been the idea of cloud computing systems[9]. Cloud computing is a relatively new concept and has been difficult to clearly compare and contrast with the grid computing model[10]. The differences between the cloud and grid computing models lies in the level of abstraction presented to the consumer. In the grid computing model the computational resources are presented at a low-level which allows the user to specify explicit usage guidelines for the entire system while the

cloud computing model presents a computational resource as a narrow instantiation of the specific capability defined system. Examples of current cloud computing systems include Amazon Web Services, Microsoft Azure, and Google AppEngine [11-13].

Proposal

The need to expand the computational resources in a massive surveillance network is clear but traditional means of purchasing new equipment every year or based on event specific needs is wasteful of resources. In this work I will provide evidence in support of utilizing a cloud computing infrastructure to perform computationally intensive feature extraction tasks on data streams. Efficient off-loading of computational tasks to cloud resources will require a minimization of the time needed to expand the cloud resources, an efficient model of communication and a study of the interplay between the in network computational resources and remote resources in the cloud.

The goal of this project is to construct a cloud processing system that will be integrated into the distributed camera surveillance network, ASAP[3]. The ASAP system uses a heterogeneous wireless sensor network to collect and prioritize multiple video data streams. ASAP performs the stream processing using only the in-network computational resources. This limitation on the availability of computational resources has restricted the number of interesting applications that can be implemented. By using cloud resources to move computations out-of-network I hope to facilitate the construction of more computationally complex applications in the ASAP system.

ARCHITECTURE

As noted in [14] Utility Computing infrastructures can be segmented into different classes based on the level of abstraction that is presented to the programmer and the level of resource management. At the highest level of abstraction Google's AppEngine is aimed at web application developers using the standard the Java API or Python API. In AppEngine the applications are expected to be request-reply based and the amount of CPU time per request is limited[15]. At the next level of abstraction is Microsoft's Azure infrastructure which requires that the hosted applications are written using the .NET libraries and compiled to the Common Language Runtime. The programmer does not have explicit knowledge or influence over the operating system or system runtime. The lowest level of abstraction is the Amazon Elastic Compute Cloud (EC2) infrastructure which exposes to the programmer knowledge of the physical system on which the application is hosted. There are no limits on the language the application is written or the operating system. In this study I chose to use Amazon EC2 to avoid any dependency on programming languages and to closely monitor the application performance.

The EC2 servers used in this report are Linux based virtual machines running on the Xen virtualization engine[16]. EC2 computational capabilities are measured in EC2 Compute Units. "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor."[17] This study was performed using small instances which provide

1.7GBs of memory and 1 EC2 Compute Unit. The costs associated with running an EC2 instance are 10 cents per hour and 10 cents per GB of data transferred out of the instance.

EC2 instances boot using an Amazon Machine Image (AMI) which is stored in the Amazon Simple Storage Service (S3). Default AMI's are provided by Amazon and developers are able to build custom AMI's from scratch. Requesting an image requires the transfer of the AMI from S3 to a local server, the local server inflates the file system stored in the image onto the local disk and the Linux system then boots the kernel.

Application Design

The face detection software was constructed using the OpenCV[18] computer vision library. To minimize the face detection time the application was implemented using Haar cascades[19]. The Haar cascades face detection method is invariant to the scale of the image and has shown be an extremely fast and effective. The invariance of the image scale was utilized to reduce the image size at the client which lowers the communication overhead to send images to the web service. The jpeg encoded images captured from the web camera were 320 x 240 pixels which were scaled down to 151 x 100 pixels before sending. Scaling down the image before sending did not result in any significant increase in the total overhead time to perform the face detection operation.

The face detection software was exposed as a web service using gSOAP[20]. gSOAP is a highly optimized automatic web service generation application that minimizes the parsing required to process SOAP encoded objects. The application was exposed as a web service to attain the maximum amount of interoperability across the heterogeneous platforms that are

used in the ASAP system and because many embedded libraries offer native support for HTTP requests.

The AMIs that are provided by Amazon are approximately 1GB in size and the average startup time from request to instance availability is 2 to 3 minutes. To reduce to time required to scale up the EC2 application a custom AMI was constructed using TTYLinux[21]. TTYLinux is a minimal implementation of the Linux operating system that implements only the core facilities that are necessary for a system to be operational. The base TTYLinux AMI image was taken from the Eucalyptus project[22], a cloud computing infrastructure software package. The statically compiled face detection web service application was added to the TTYLinux AMI which increased the size of the AMI from 5MB to 30MB.

RESULTS

A single small EC2 instance using the custom TTYLinux AMI was requested, the EC2 instance was configured to repeatedly connect to my local machine when the instance was available. This was done to measure to total time from request to availability of the EC2 instance. After 32 trials the average availability time was 25 seconds which agrees with the results of researchers in the Eucalyptus project[22].

Figure 1 shows the system setup that was used to conduct the performance measures for this report. The client machine specifications were: Intel Core 2 Duo 1.3 GHz processor and 2GBs of memory. The HTTP Axis camera interface was used to capture 15 frames per second.

Figure 1: This figure depicts the system setup. The client machine is connected on a wireless network to a web camera. The measured throughput between the client machine and the cloud instance is 140 kbps.

The top graph in Figure 2 shows the measurements that were collected from the client overlaid with the time per frame the server spent processing each frame. These initial measurements reflect the sending, receiving and processing time when the 320 x 240 pixel image is sent to the face detection web service with no scaling. As can be seen the communication overhead dominates the application performance and processing requires roughly 1 second per frame.

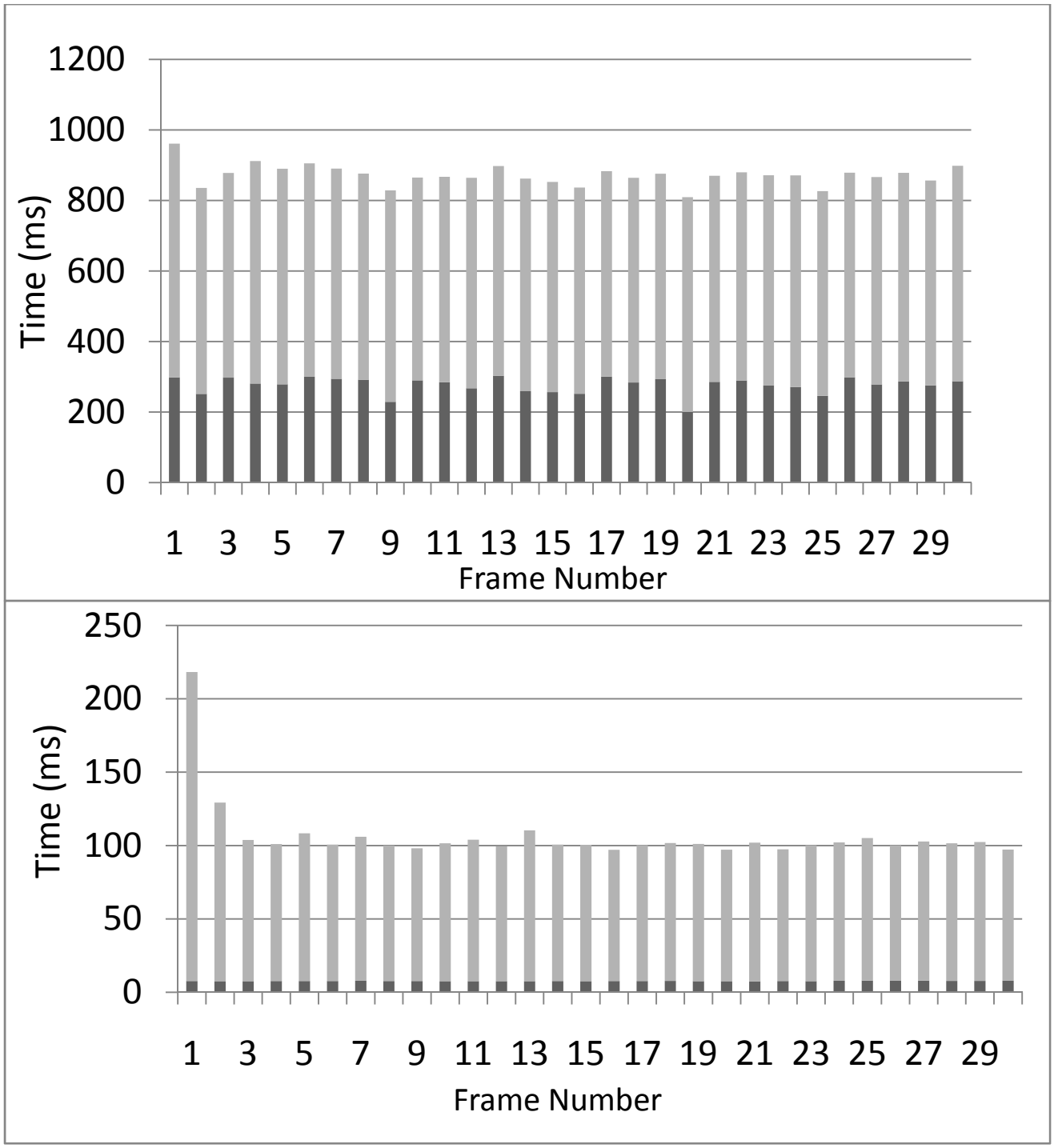

Figure 2: Each graph depicts the total time required per request for an image processing sequence. The light gray bars represent the communication time while the darker gray bars represent the time spent on the server side processing the image. The top graph shows the sequence using the 320 x 240 pixel image and the bottom shows the performance using the reduced 151 x 100 pixel image.

For the surveillance system to be of practical use it must be able to process at least 10 to 15 frames per second. Additionally in the top graph of Figure 2 the average time for the Haar cascades algorithm to perform the face detection is 300ms. This large processing time can be attributed to the size of the image and the amount of time required for the algorithm to decompose the image into rectangular regions.

The bottom graph of Figure 2 shows the results of a similar image sequence with the images scaled down on the client side to 151 x 100 pixels. The communication dominates this sequence even more than the previous trial but in this case the reduction in the image size

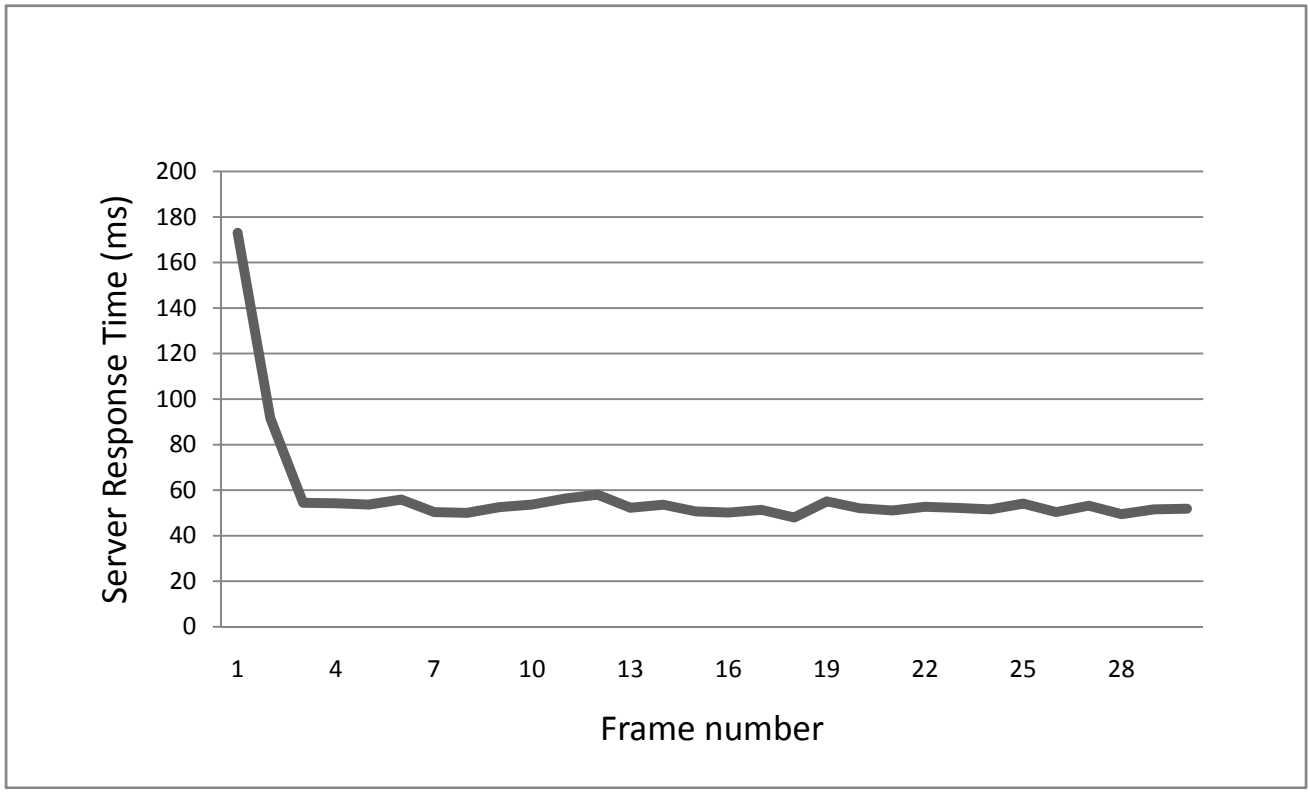

Figure 3: This image processing sequence shows that the overhead of building the TCP connection adds an additional 100ms to the image processing. After the connection is established the average time to process an image falls to 57ms.

reduces the average cost of the communication by 400ms. The reduction in the image size provides an extremely large decrease in the amount of time required to perform the face detection algorithm. The time to perform the face detection drops from an average of 300ms to 15ms. Choosing to reduce the image to 151 x 100 pixels was the result of experiments with an image containing 4 faces. Reducing the image past 151 x 100 pixels resulted in a false negative detection of faces within that image.

Figure 3 shows the results of a single image sequence were it is apparent that the time to build the TCP connection adds an overhead to the initial detection sequence. After the TCP connection is established the remaining image frames have and average total detection time of 57ms. By establishing the TCP connection to the EC2 instance prior to sending the first image I was able to process 16 frames per second.

CONCLUSION

Future work will focus on integrating cloud computing into the ASAP distributed camera network. Off-loading the computationally intensive tasks onto the cloud infrastructure facilitates the implementation of more complex analysis of the data collected in the ASAP sensor network. Moving computations into the cloud will also allow the use of the local in-network resources to provide an even greater level of responsiveness for real-time situations that exhibit a high priority level.

This study has attempted to show that moving computations into the cloud is feasible and easily allows for the processing of 10 to 15 frames per second. Integrating cloud services into the ASAP system will require more research regarding the most effective way to predict the need for more computational resources in the cloud. The 20 to 30 second delay in scaling the cloud resources presents a serious problem in regards to the real-time responsiveness of the ASAP system. Another interesting area is the use of selective filters implemented at the clients to reduce the number of images which are sent to the cloud instance. Simple and fast methods to selectively choose images that are sent for processing are necessary to ensure that using the cloud remains a cost effective solution.

REFERENCES

[1] D. o. H. Security, *Budget-in-Brief Fiscal Year 2006*, 2006.

[2] EPIC. "More Cities Deploy Camera Surveillance Systems with Federal Grant Money," http://epic.org/privacy/surveillance/spotlight/0505/#footnote2.

[3] J. Shin, R. Kumar, D. Mohapatra *et al.*, "ASAP: A Camera Sensor Network for Situation Awareness," in OPODIS, 2007, pp. 31-47.

[4] J. A. Chamot. "Law Enforcement Monitored Numerous Inauguration Security Cameras with New System," March 20, 2009; http://www.govtech.com/dc/articles/616262.

[5] I. Foster, and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing,* vol. 11, no. 2, pp. 115-128, 1997.

[6] C. Kesselman, and I. Foster, *The Grid: Blueprint for a New Computing Infrastructure*: {Morgan Kaufmann Publishers}, 1998.

[7] B. Agarwalla, N. Ahmed, D. B. Hilley *et al.*, "Streamline: a Scheduling Heuristic for Streaming Applications on the Grid," *CERCS Technical Reports*, Georgia Institute of Technology, 2005.

[8] G. V. M. Evoy, and B. Schulze, "Using clouds to address grid limitations," in Proceedings of the 6th international workshop on Middleware for grid computing, Leuven, Belgium, 2008.

[9] L. Siegele, "Let it rise: A special report on corporate IT," 2008.

[10] L. M. Vaquero, L. Rodero-Merino, J. Caceres *et al.*, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.,* vol. 39, no. 1, pp. 50-55, 2009.

[11] "Google AppEngine," http://code.google.com/appengine/.

[12] "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2/.

[13] "Microsoft Azure," http://www.microsoft.com/azure/.

[14] M. Armbrust, A. Fox, R. Griffith *et al.*, *Above the Clouds: A Berkeley View of Cloud Computing,* UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[15] "Google App Engine Quotas," http://code.google.com/appengine/docs/quotas.html#Free_Changes.

[16] "XenSource Inc. Xen.," http://www.xensource.com/.

[17] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/instance-types/.

[18] Intel, "OpenCV Open Source Computer Vision Library."

[19] P. Viola, and M. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision,* vol. 57, no. 2, pp. 137-154, 2004.

[20] R. A. V. Engelen, "Pushing the SOAP Envelope with web services for scientific computing," *In proceedings of the International Conference on Web Services (ICWS), pages 346–352, Las Vegas*, 2003, pp. 346-352.

[21] "Ttylinux home page," http://www.minimalinux.org/ttylinux/.

[22] D. Nurmi, R. Wolski, C. Grzegorczyk *et al.*, *Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems*, University of California, Santa Barbara, 2008.