Performance Evaluation of Client-Server Architectures for Large-Scale Image-Processing Applications

Michael D. Beynon, Renato Ferreira, Asmara Afework and Ganti Krishna Mohan
UMIACS and Department of Computer Science
University of Maryland
College Park, MD 20742
{beynon,renato,asmara,gkm}@cs.umd.edu

Introduction

The goal of this study is to conduct a performance evaluation of the Virtual Microscope, a software system employing a client-server architecture to provide a realistic emulation of a high power light microscope. The system is required to provide interactive response times for the standard behavior of a physical microscope. These behaviors include continuously moving the stage and changing magnification and focus. In addition, a software solution can enable new modes of behavior that cannot be achieved with a physical microscope, such as simultaneous viewing and manipulation of one slide by multiple users. The client software runs on an end user's PC or workstation, while the database software for storing, retrieving and processing the microscope image data runs on a high performance parallel computer, potentially at a remote site.

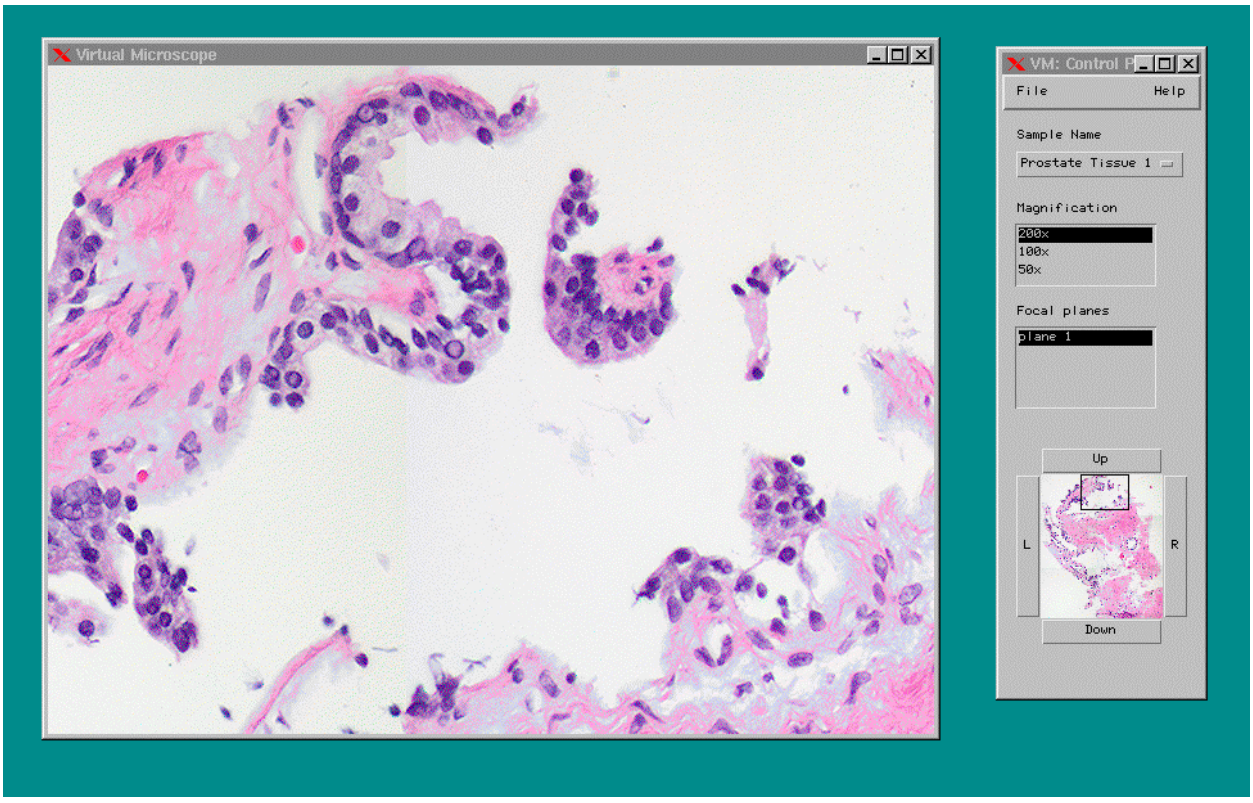

Figure 1 - Virtual Microscope client browsing a slide

The main difficulty in providing the functionality of the Virtual Microscope is dealing with the extremely large quantities of data required to represent a large collection of slides. For example, using the digitizing microscope currently available at Johns Hopkins, a single spot at a magnification of 200X produces a grid of 1000x1000 pixels. We estimate that an array of 50x50 spots is required to cover an entire slide, and each pixel is a three byte RGB

color value. Under this scenario, one slide image requires about 7 Gbytes. However, such an image captures only a single focal plane, and many specimens will require capture of between five and thirty focal planes. Clearly there is an enormous storage requirement, and there are also the attendant difficulties in achieving rapid response time for various types of inquiries into the slide image database.

The Virtual Microscope is representative of a larger class of applications that involve browsing large multidimensional datasets. Similar to database transaction processing, the clients issue small requests for data to the server. The reply consists of data several orders of magnitude larger than the size of the request. Moreover, the amount of data processed by the server in order to produce the response for the client can be much larger than the reply.

For this study, we were interested in determining the number of clients that can be supported at varying quality of service levels. One measure of quality of service for a client is response time. We are also interested in the server response rate (throughput) in MB/s or in TPS, and in determining the bottlenecks in the system in order to help us in improving it. To achieve this end, we designed a model of the system and built a simulation of it.

In the case where several clients are accessing "similar" regions of a dataset, a Proxy can be useful in reducing the load on the server by making a single request to the server when multiple clients issue the same request. Another benefit realized by the use of a proxy is a type of cooperative caching across different clients. From the point of view of a client, the proxy will appear to be a server, and from the point of view of the server, the proxy appears to be a client (albeit with higher demand for data).

System Design

Figure 2 shows the system modules and how they are interconnected. The clients are image browsers with which the users interact. The users are able to select a slide and then navigate through it. After selecting a slide, the user receives a smaller version of the slide to use as an overview. The user chooses to view the data at one of several available magnifications. The user has two ways to move through the slide. They can drag the view box across the overview image. Alternatively, they can use the fine control buttons to move a small amount in any of the four possible directions, which incrementally moves the view box. Requests flow from the clients, in response to users' actions, to the server. When they use the control buttons, the client will only request the data that is not already present, resulting in smaller requests.

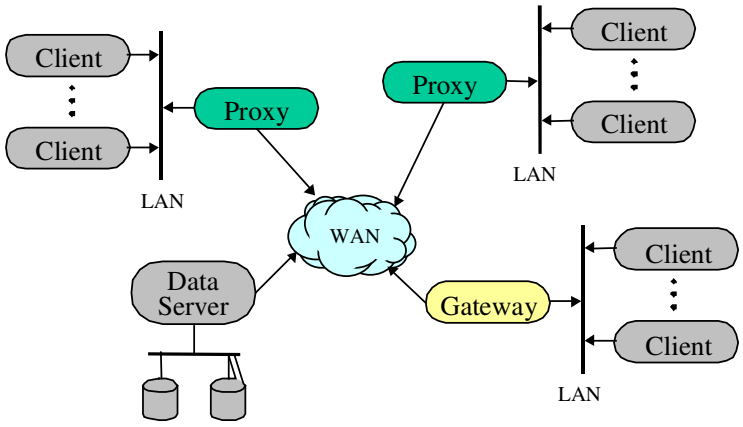

Figure 2 - System Architecture

On their way to the server, requests may go through a proxy. The proxy then makes requests to the server. The proxy maintains a cache of recently accessed data. If the proxy does not have the data cached and the proxy has not already requested the data, a request for it is sent to the server. It notes that the requestor is a consumer of that data and forwards it upon receipt from the server. If another client has already requested the data for the new request and

the response has not yet arrived from the server, it is noted that the data has another consumer and when the response is received, it is forwarded to all consumers. If the proxy has the data cached the request can be answered on the client's local area network, avoiding the wide area connection to the server. We expect that these proxies will play a major role in the scalability of the server, in the sense of the number of clients it can simultaneously support.

The server is divided into a single frontend node that receives all the requests from clients, and a collection of backend nodes running in parallel to retrieve and process the requested data. The frontend accepts requests from clients. The frontend orders the requests in some way and then sends them to the backend. The backend is responsible for efficiently serving image data. In order to produce an image, the data has to be read from disk and an image of the specified magnification must be reconstructed. Once this has been done, the backend sends the data back to the requestor [which can be a client or a proxy]. Figure 3 depicts the path of a request through the various components of the system.

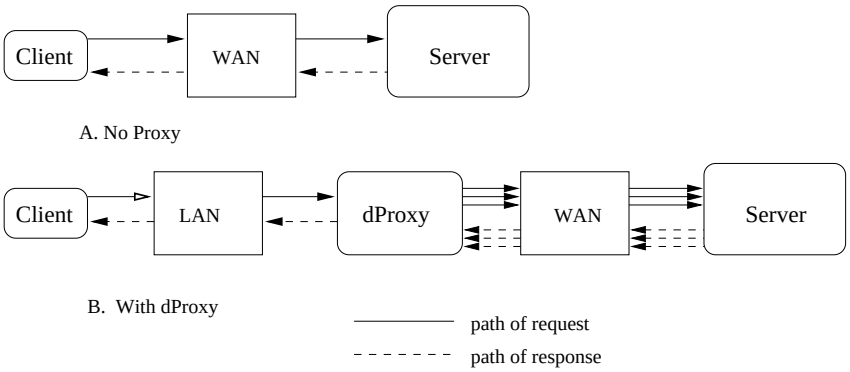

Figure 3 - Request flow through the system

Experimental Design

This section explains the process we used in performing this study. There are many parts to the system under study and the performance of each of these components depends on the others to some extent. The interaction among the components precludes the use of static modeling techniques because they would fail to estimate the dynamic characteristics of the system. So we relied on simulation for our study. The high level procedure we used in conducting this study is as follows:

1. Perform component level characterization studies, and generate performance histograms.
2. Gather user workloads using current Virtual Microscope implementation.
3. Analyze results of component and workload characterizations to find representative distributions.
4. Build an event driven simulation using the distributions.
5. Run simulations to cover factor space.
6. Analyze and present results.

Parameters influencing performance

There is a multitude of factors affecting the performance of the Virtual Microscope. It is easiest to consider them component-wise. They are enumerated in Table 1. Time and complexity constraints prevented all of these from being used as factors in our study (see Table 2).

Client-related parameters

The number of clients in the system will definitely affect the response time seen by each client and the utilization at the proxy and the server. While the client request rate is also an important parameter, we did not vary it for the experiments. Instead, we characterized the request rate displayed in traces taken from the Virtual Microscope and modeled this for our simulation. The size of the data result was not taken to be a parameter. Instead, a client was considered to be capable of generating two types of requests -- large (3 Mbytes) and small (300Kbytes). The processing time for the client to combine the pieces of the answer returned by the server and then to display it is another parameter that was not used as a factor. We merely modeled the current incarnation of the client for this and kept it constant.

Network-related parameters

We instantiated no network parameters. Instead we simply measured network delivery delay on large and small requests on a LAN and a WAN with an ATM interconnect for most of the long-haul. We then used a simple model of this delay in the simulation.

Proxy-related parameters

When we use a proxy, several parameters would impact performance. Those that are most important from a performance perspective, is the cache size, cache replacement policy, and the probability of commonality between clients. The probability of commonality is how likely given client's request has already been requested by a sibling client. Here sibling describes clients that share a common proxy. We both use and do not use a proxy, but most of the proxy parameters described in Table 1 are not varied. We vary *cache hit rate*, which is an abstraction of the other parameters. This was not merely a choice to make the model easier, rather it was required due to the abstractions used for the client workload.

All of the proxy work is in-core. Disk I/O is never used in the current implementation of the proxy. As a result, the majority of the time spent processing requests involves cache lookups and socket communication. To model the proxy, we instrumented the proxy, and benchmarked proxy performance on small and large requests. The distributions for the various control flow paths through the proxy were found to be very consistent and quite nicely modeled by various standard distributions.

Server-related parameters

We realize that complex interactions happen within the server, especially when it is serving multiple requests at the same time. However, due to time constraints and the extreme complexity of these interactions, we have decided to simplify and treat it as a single black box. We benchmarked the service times for large and small requests and then used that as a model for simulation purposes.

Location in System	Name	
Client	Number of clients	Size of data result
	Client request rate (think time)	Processing performed
Proxy	Used?	Probability of commonality
	Cache size	Cache replacement policy
	Processing performed	Overhead
Network	Type of interconnect	Load
Server	Number of Disks	Processing performed
	I/O delays	

Table 1 - System Parameters

Location in System	Name	Levels
Client	Number	1, 2, 4, 8, 10, 20, 30
Proxy	Used	Yes, No
	Cache hit rate	0, 0.25, 0.5, 0.75, 1.0

Table 2 - Factors affecting system performance

Workload Characterization

We characterize the workload as a stream of client generated requests. The request stream is determined by the requests and by the rate at which the client makes requests. A single request can be abstracted as the size and zoom level of the data requested.

Since the Virtual Microscope is currently in the prototype phase, workload characterization is extremely difficult. For example, we cannot examine one of our driving scenarios -- that of a group of individuals browsing the same slide in an educational setting, where we can expect sufficient commonality across clients to justify the use of a Proxy. We were, however, able to capture traces of a trained pathologist using the system. The user examined several slides, thoroughly searching each slide for any abnormality, as they would with a physical microscope. The obvious problem here is that a singleton data point is hardly a case for general arguments. Thus we tried to correct this flaw by abstracting out the pattern of use from the instances we collected.

	8	**4**	**2**	**1**
8	0.65	0.35	0.00	0.00
4	0.02	0.68	0.18	0.12
2	0.00	0.05	0.50	0.45
1	0.15	0.04	0.08	0.73

Table 3 - Probability transition matrix for zoom levels

Zoom	P(3MB)	P(300KB)
8	0.99	0.01
4	0.99	0.01
2	0.92	0.08
1	0.70	0.30

Table 4 - Probability of large and small requests

A client can request data at one of four zoom levels: 1, 2, 4, or 8. Zoom level 1 corresponds to the highest resolution version of the image; level 8 indicates the lowest resolution. We discovered that the previous level heavily influences the zoom level to be chosen next. Table 3 describes the probability matrix used for choosing a zoom level, given the previous zoom level.

Originally, we hypothesized that the zoom level of a given request could be used as a predictor variable for data size of the current request. The theory was that at lower resolutions, the user would not use the fine-grain control buttons, so all requests would be large. At high resolutions, fine-grain control would sometimes be used, resulting in small requests. While this seems to be reasonable, the pathologist who generated our traces never used the fine-grained control. We decided to assume that he was a slightly atypical user and to consider our hypothesis as valid, though the probabilities for creating small requests is quite small, as shown in Table 4.

Based on a cursory examination of the traces, it was determined that zoom level is correlated to the amount of time until the next request. The time between two consecutive requests is termed *think time*. We noted that there was a different distribution for the think times that occurred after each of the zoom levels (see Figure 4).

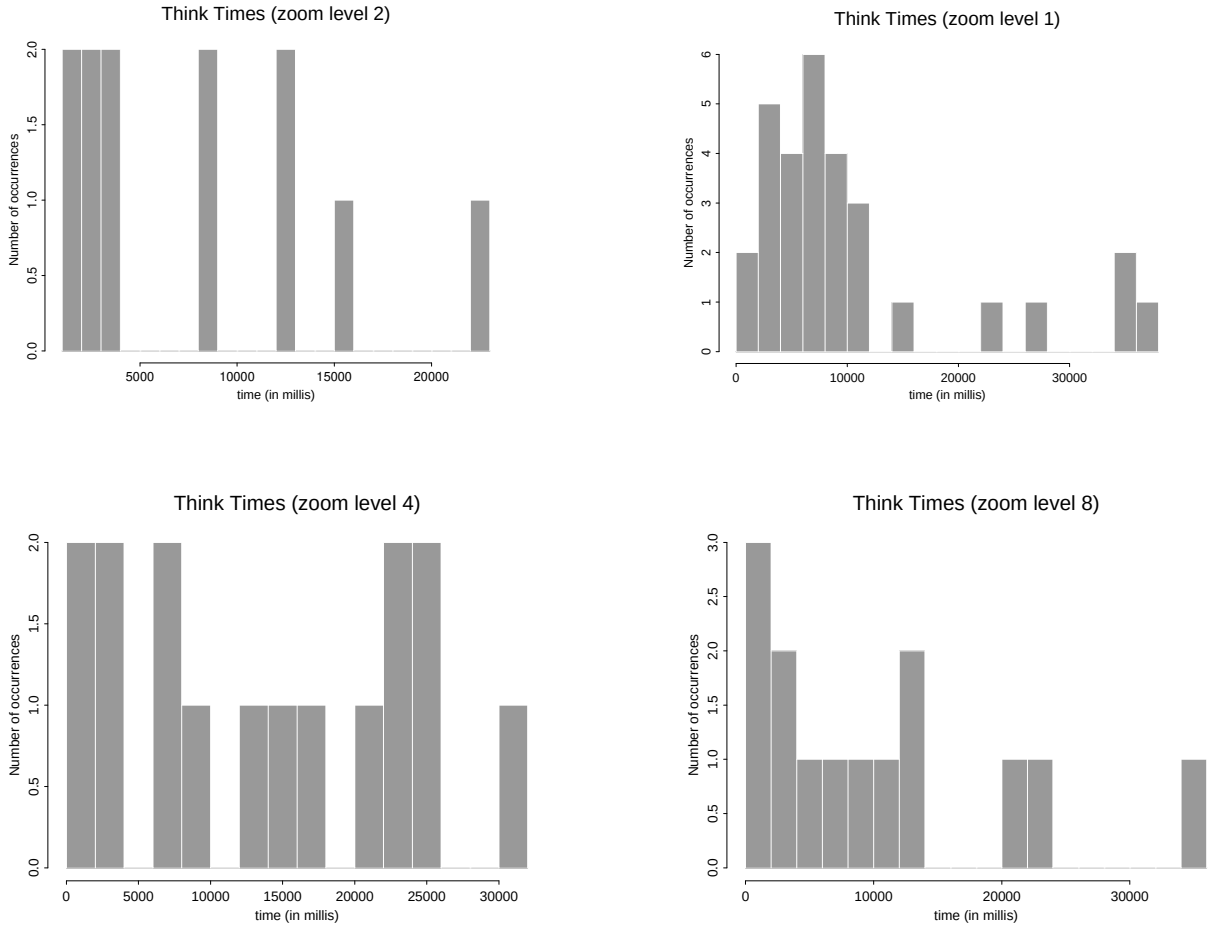

Figure 4 - Client Think Times

Client

In our simulation, the client generates the workload that drives the entire system. Each client produces a stream of requests and their aggregate is the system's workload. The end to end response time is measured from the point of view of the client. In the execution time line of the client we can identify four events of interest.

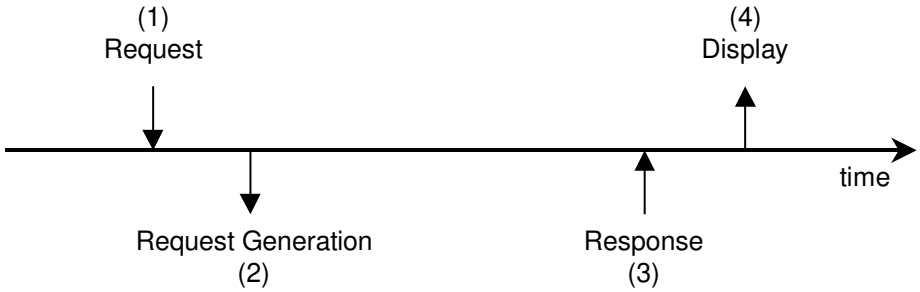

Figure 5 – Event sequence

When a user manipulates the interface to cause a new request to be generated, this is a Request event (1). When the client actually sends a request to a server, then Request Generation (2) has occurred. The Response (3) event happens when the reply is completely received by the client. Finally, the Display (4) event causes the image to be displayed in the client window.

The interval between (1) and (4) is the latency we are interested in measuring. We do not consider the time between (1) and (2), and set it to be zero for the experiments. The interval between (2) and (4) is the service time. Service time is then decomposed into two pieces: (2) to (3), which includes the communication costs, server processing time, and in some cases, the proxy processing time and (3) to (4) which is the time for the client to assemble the server's response and then to display it.

From our traces, we obtained the display delays for the client. It was determined that the display time could be considered normal. It was represented as the normal distribution N(1158.64, 174). A quantile-quantile plot of the display times is shown in Figure 6.

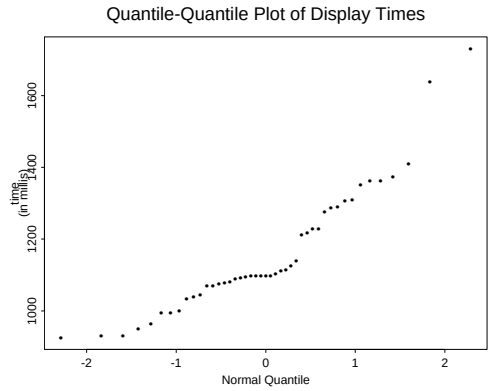

Figure 6 - Quantile-Quantile Plot of Client Display Times

For the experiments, we varied the number of clients in the system. The values considered were 1, 2, 4, 8, 10, 20, and 30 clients. In the simulation, the client arrivals in the system are exponentially distributed. Once in the system a client remains until it has made a fixed number of requests. This was set to be 200 requests, and was chosen to insure the system ran sufficiently long to get more steady state behavior. The clients are initialized to appear as if each had just finished processing a request for the highest zoom level. From then on, the zoom level of the next request is chosen randomly based on the previous request and the probabilities given in Table 3. Once the zoom has been selected the size of the request is assigned randomly assigned based on probabilities in Table 4. Finally, the time between requests is chosen based on the distributions of Figure 4.

Network Experiments

The LAN is the part of the system between the client and the proxy. We assume that any client and proxy that needs to communicate will be on the same local area network (LAN). The wide area network (WAN) is the part of the system between the proxy and the server. The proxy determines if there is any commonality in the requests sent by the clients. If any commonality is found, then the proxy combines these requests before sending them to the server so that the amount of data transfer is minimized.

We decided to conduct experiments to determine the delay of the packets as opposed to obtaining traces and analyzing them because the traces may not be representative of the problem at hand. We want the delay for packets of certain sizes that correspond to request and replies of different sizes of queries.

We are interested in the delay for packets of sizes equal to 100 bytes, 300 KB and 3 MB. A packet of 100 bytes corresponds to the client request, a packet of size 300 KB corresponds to reply for a small query and a packet size of 3 MB corresponds to reply for a large query.

A benchmark program has been written which opens a TCP connection with a server and transfers a packet and waits for the packet to come back from the server, The server just accepts the packet and echoes it. That the packet itself might be fragmented in the process of sending is not taken into consideration because it is the delay for the entire packet that is of interest to us. The total round trip time is measured once the packet comes back. This is done because the clocks at the server and the client may not be synchronized.

LAN

The results for a packet size of 100 bytes are presented in Figure 7. It was observed that the delay takes a value of 975 microseconds with a probability of 0.9 and 1950 microseconds with a probability of 0.1. This experiment was conducted between various pairs of machines. The number of trials for each run is 1000. All the runs were consistent with one another.

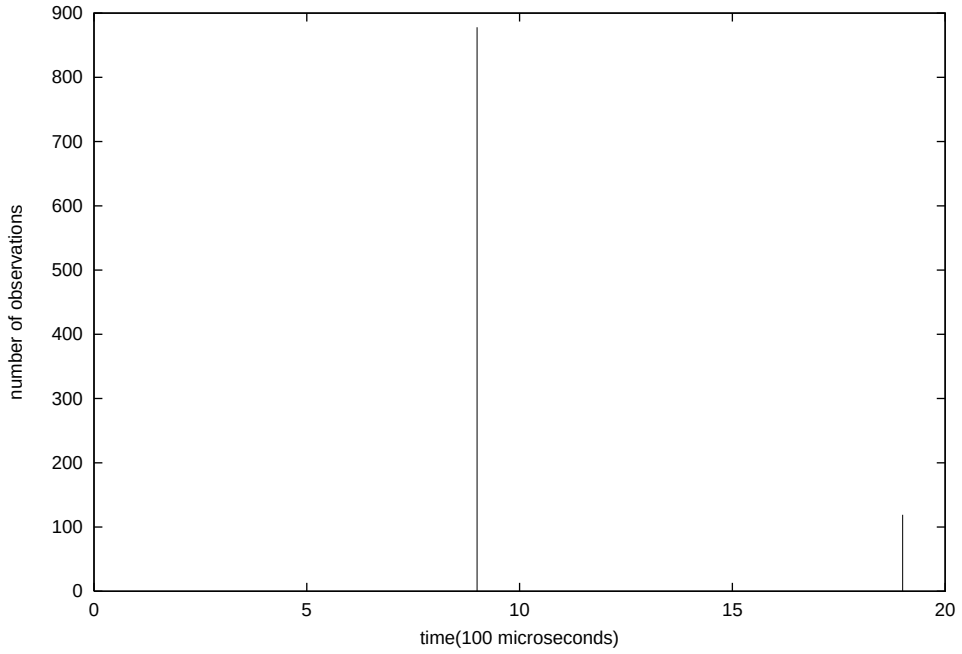

Figure 7 - Roundtrip delay for 100 bytes over LAN

When the packet size was increased to 300 KB, which corresponds to the reply for a small request, it was observed that the delay was a bit more non-deterministic. The delays were following a normal distribution. The number of trials taken is 1000. The number of buckets was 100000. This means that the delay was calculated at the resolution of 100 microseconds. The peak value occurred at a value of 0.9953 seconds. However, the mean of all the observations occurred at a value of 0.94558 seconds. This is because the distribution is slightly skewed towards the left. If this value is chosen as the mean of the normal distribution, the results will be distorted. Hence, the mode is used as the measure of central tendency and the standard deviation is computed taking the difference of the observations with the mode. The standard deviation is equal to 0.005683 seconds when calculated with respect to the mode. So, the confidence interval of the mean at 90% confidence is [0.9950,0.9956] and at 95% confidence it is [0.99495,0.99565]. The results are shown in Figure 8.

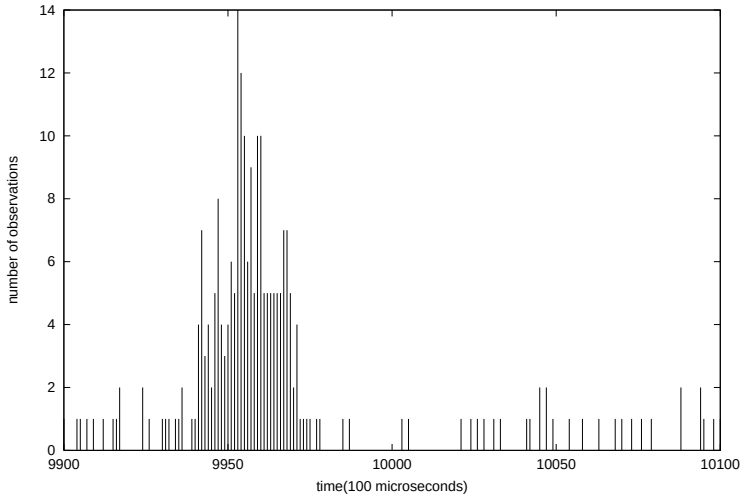

Figure 8 - Roundtrip delay for 300 KB on LAN

When the packet size was further increased to 3 MB, which corresponds to the reply for a large request, the same trend was observed. The number of trials taken is 100, which was reduced from 1000 as in the previous case, due to the time needed to send this much data 1000 times. The peak value occurred at 7.37 seconds whereas the mean was 7.531 seconds. The standard deviation is equal to 0.0307 with respect to the mode. We decided to approximate it to a normal distribution with a mean of 7.37 seconds and standard deviation of 0.0307 seconds. The confidence interval for the mean at 90% confidence is [7.365,7.375] and at 95% confidence it is [7.364,7.376]. The results are shown in Figure 9.

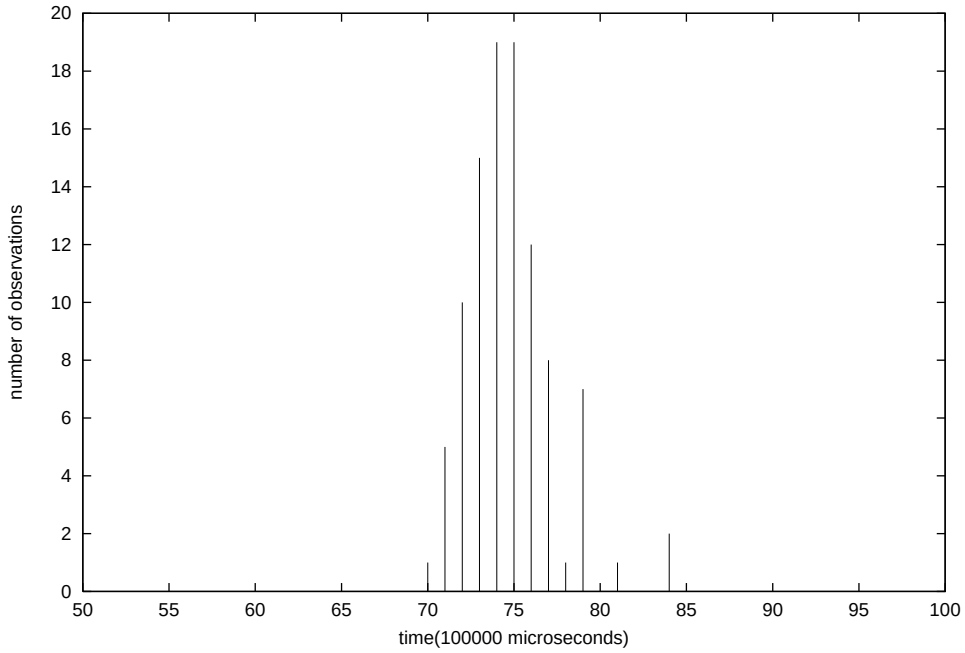

Figure 9 - Roundtrip delay for 3 MB over LAN

WAN

For the WAN, we measured delay between a computer in San Diego and the University of Maryland. This link may not be representative of the actual Internet because the connection is partially a dedicated ATM link between the University of Maryland and the San Diego Supercomputing Center. This was decided to be a necessary assumption. For the magnitude of data we are sending in the scope of the Virtual Microscope, and for the low delays needed with interactive use, anything less than a reliable high-speed network would be unusable.

For a packet size of 100 bytes, the delay was very deterministic and we decided to represent it as a constant value of 0.71 seconds because it had a very small variation. We also expect that the delay of the request packet will not be a major bottleneck in the overall performance. The results for this case are presented in Figure 10.

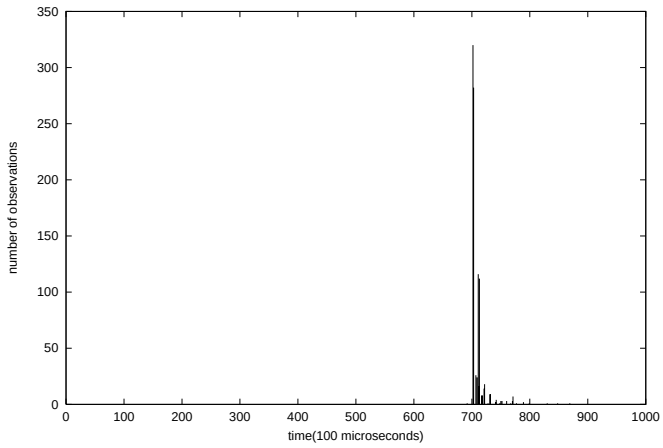

Figure 10 - Roundtrip delay for 100 bytes over WAN

For a packet size of 300KB, the delay was observed to follow a pattern indicated in Figure 11. A large number of packets had a roundtrip delay of around 3.75 seconds, but there were a lot of other packets that had considerably higher delays. If this is modeled by a normal distribution as we did in the LAN, we would not be representing the data accurately. This is because a normal distribution is symmetrical with respect to its mean and the values we obtained are not symmetrically distributed. We could have divided the distribution as two parts, one corresponding to the values near to the mode and the other corresponding to the rest of the portion of the graph. But, the second part of the graph does not seem to follow any known distribution. So, we decided to use the histogram itself in our simulation. We have taken samples from the delay values and for n=1000, 10000 and 100000, the plots look very similar to the original one. The results are for the original measurements are shown in Figure 11. The figures for the sampled data are shown in the appendix.

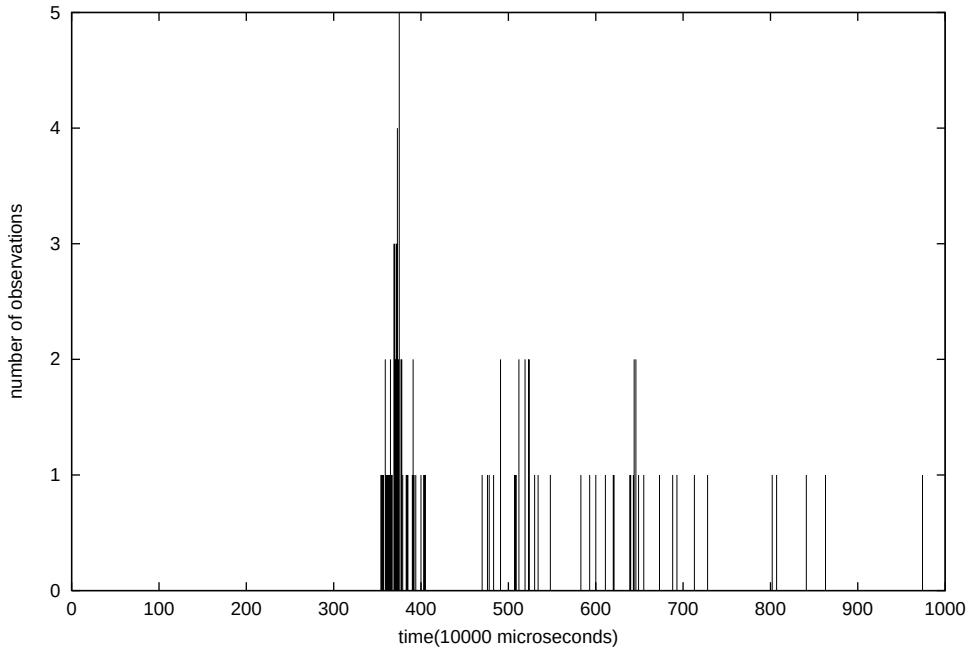

Figure 11 - Roundtrip delay for 300 KB on WAN

For a packet size of 3 MB, the delay was observed to follow a pattern similar to what was observed for the previous case with the peak occurring at a value of 32.9 seconds. We decided to use the histogram in this case also. If we used a normal distribution instead, accuracy would be lost. The results are shown in Figure 12.

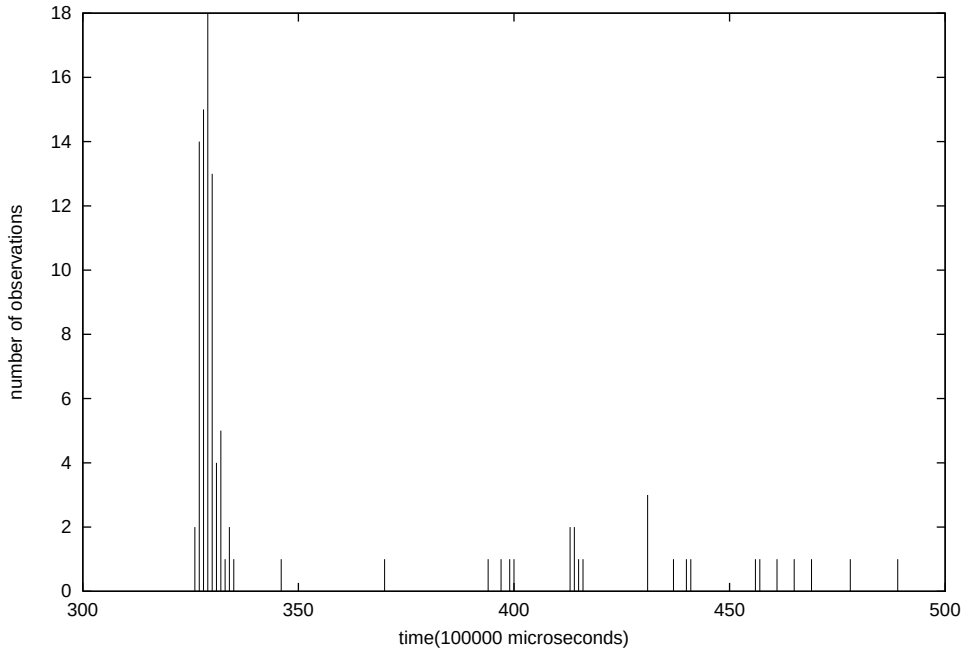

Figure 12 - Roundtrip delay for 3 MB on WAN

Proxy

Of all the components of the Virtual Microscope, the Proxy is the only part not understood enough to have a real implementation currently in use. It represents work in progress, and was a motivating force behind this performance study.

To do an in-depth study of the proxy, we would need to have a better characterization of system workload that included measures of commonality present in the data. The cache policy used is dependent on the actual data requested, so an abstraction of the workload may not work. In addition, commonality is closely tied to the actual data requested in a multi-user situation, thus the limited nature of our traces prevents us from having this level of detail in our characterization. In the cases we use a proxy, we assumed all clients connected to the proxy, and the proxy had the only connection to the server. To limit the scope of the proxy model, the abstractions and assumptions were introduced.

When a client data request arrives at the proxy, it is converted into a set of requests for contiguous blocks of the dataset. The choice of block size is currently assumed to be 256KB each. The requests for blocks needed are sent to the data server, and the subsequent data reply is received by the proxy. When all the blocks needed to satisfy a client data request are collected by the proxy, a data reply is sent to the client. The proxy recognizing outstanding requests for blocks, and not re-requesting it from the server exploits commonality in time (temporal locality). The false sharing that results from blocking the data set, can be thought of as a limited form of pre-fetching. All these aspects are modeled in the proxy using a simple probability based test on a per block basis. A Bernoulli trial is used to determine if a block resulting from a decomposed client data request is found in the proxy's local cache. This probability is the factor *cache hit rate*.

To characterize the proxy for simulation, benchmarks were performed on the instrumented proxy. Several client request streams of small (300KB) and large (3MB) requests were sent to the proxy. Since the proxy is an in-core engine, most of the work involves cache lookups and communication. The current implementation uses extensible hash tables, which exhibits regular behavior, thus making the modeling easier. Table 5 lists the distributions used for various control flow paths of the proxy simulation. Note the benchmarked performance was found to be fairly consistent and well suited to normal distributions.

Description	Distribution Used
Processing a client data request	N(0.000185, 0.000024)
Processing a cache hit (per block)	N(0.000076, 0.000007)
Processing a cache miss (per block)	N(0.000142, 0.000009)
Processing to send hits to client	N(0.289, 0.028)
Processing to receive a server data reply	N(0.204, 0.023)
Processing to send a server data reply to a client	N(0.289, 0.028)

Table 5 – List of modeled Proxy processing distributions

One major caveat in the current proxy implementation is that there is currently no eviction policy in place. The eviction policy would be used in the case where a new block of data has been processed and is ready to be added to the cache, but there is no more room, thus an existing entry must be evicted. Part of the reason for this is because we do not yet understand the behavior of the Virtual Microscope to randomly choose eviction policies to implement. As a result, the processing overhead to receive a server data reply should be larger.

Server

The server is the component responsible for efficiently retrieving the data from the storage system. This component relies on many tricks to improve its performance. Two of the most important ones are data declustering and data clustering. The former consists of the subdivision of the multidimensional dataset into several blocks for which the retrieval is efficient (typically one or a small multiple of a SCSI block). These blocks are then declustered across several disks to which the server has access. The declustering is done in such a way that blocks that are close to each other are placed on separate, so that a larger number of disks are accessed for each query. The concept of distance is user defined, and for the instance of the virtual microscope is the manhattan distance of the rectangular image chunks. The clustering happens on a later step, in which for all the blocks that are declustered to the same disk are again sorted so that they stay close to its neighbors. The goal here is to improve locality for accesses to the same disk.

Also, the server relies on asynchronous I/O, request reordering and other tricks. Request reordering guarantees that data will retrieve in a sequential sweep of the disk, instead of performing random I/O. We have seen about 40% performance improvement on the server due just to this reordering. The advantage of using asynchronous I/O is the overlapping of the I/O with the computation and the communication operations. For the particular case of the microscope very little is gained by that for the computation time is negligible when compared to the I/O time in most cases.

In order to characterize the server we conducted several controlled experiments, varying different input parameters. These parameters are request size and zoom factor. The request size defines the size of the output. The zoom factor, on the other hand, defines how much extra I/O is necessary on the server side, and that is what dominates the service time. The idea of the characterization is to come up with a model for the service time given these two parameters.

We ran 8 different experiments which covers the combination of different input sizes (small = 300KB and large = 3MB) and zoom factors (1, 2, 4, 8). The zoom factor works in the opposite direction as one might think. A zoom of 1 means retrieve full magnification, or whatever is on the disks. If a zoom factor of 2 is selected it means that the program should retrieve twice as much data on each dimension (four times more data overall), and then subsample it to reduce to the output size. So, a higher value for zoom means lower magnification for the image and more work for the server. We ran each of the experiments 100 times in order to achieve some statistical significance to our measures. We also tried to minimize the impact of other aspects like disk cache or workstation load during the tests.

We first measured the times separately and the table above shows the values obtained. Notice that the processing time is negligible compared to the I/O time per request. The communication time measures just the time to copy the data to the protocol stack buffers. The I/O time dominates that time too. Notice that the time to actually propagate the data to the recipient side will be modeled on the network part of the simulation.

Given the results, we decided to model the service time as a linear combination of the input parameters. It seems reasonable that the service time is directly proportional to the size of the output. We obtained roughly that behavior on our measures so we decided to assume that. For the zoom factor, we noted that the service time is also linear on the square of it. The reason for the square is that the zoom is applied in both dimensions of the image.

The server is modeled using the following result:

$$Server_Service_Time = (zoom^2 * 0.0304 + 0.1921) * size / 3MB$$

Simulation

Our simulator was written in C++ and uses the C++SIM package to perform discrete process oriented simulation. The simulator was run on one of the Dec Alpha SMP nodes of the University of Maryland Alpha Farm. Each experiment was run for 100 trials. Most of the data reported will be reflect mean values of the 100 trials. Random numbers were generated from independent streams, and each stream starts with a seed that is 100,000 values away from the last stream's seed. This was done to prevent dependencies that do not exist in the real system to appear in the simulated model. On the above described machine, the full factorial set of runs required >3 hours to complete.

We simulated two configurations based on our reference architecture shown in Figure 2. In both, there is a single data server connected by a WAN. The left side of Figure 13 illustrates the *No Proxy* configuration where a set of clients are connected by a LAN, and must use a gateway to reach the WAN to communicate with the data server. We assume the gateway adds no overhead, as in a hardware router. When we add a proxy in place of the gateway, we have the configuration shown in the right of Figure 13.

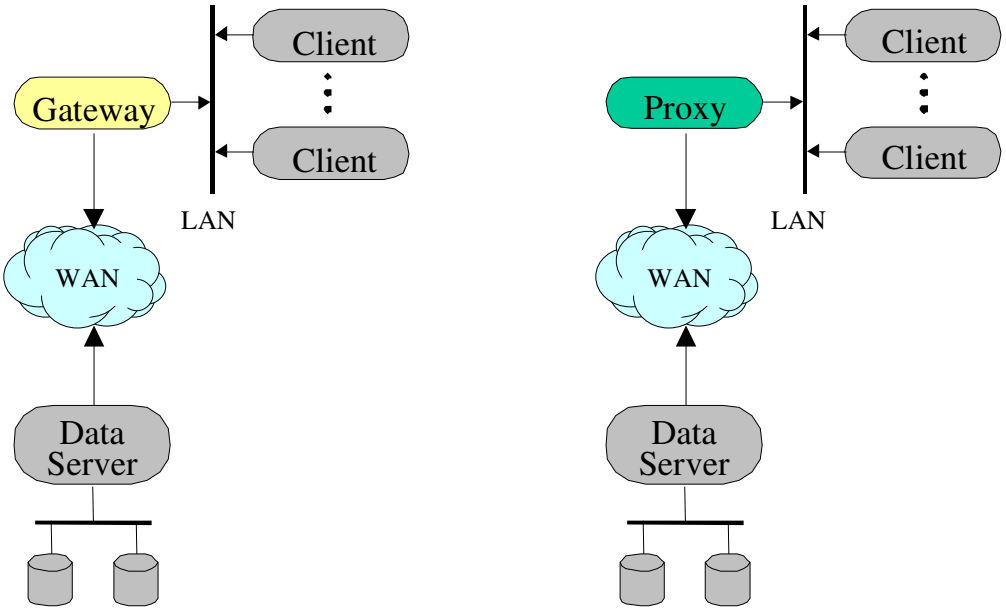

Figure 13 - Simulation Configurations

Results

The outputs of our simulation runs are concerned with a few important metrics. The first order concern is how fast does the system behave for any client. For this, response time is the obvious choice. A second order concern is then to see how well utilized each piece of the system becomes to learn how well the system could perform as the number of clients is scaled.

Response Time

As stated earlier, the response time we care about is the time between the client request and when the client receives all the data. Since sending more data over a network link can take considerably longer than sending less, we consider response time in three ways. In Figure 1, the mean response time across all clients is shown as we scale the number of clients in the system. Several lines are shown in the graph. As a baseline, the system without a proxy is given. In this case, the proxy machine would simply be a gateway machine without the ability to coalesce requests or to do any caching. The remaining plots are for configurations with the proxy turned on, with different cache hit probabilities.

The response times shows on the graph represent the mean response time seen by all clients for a given cache hit rate and number of clients. The response time value for each client used to compute the mean, is the mean response time of all the requests sent by that single client. In general a mean of means may not make sense, but for an overall analysis we found it appropriate. For the No Proxy case, the response time increases as the number of clients is increased almost linearly. This is expected to not be completely true for the real system due to congestion non-linear backoff protocols. Our crude network model only insure that a single entity can be writing to the network at a given point in time. Other than this queuing, no other contentions is considered, thus explaining the linear appearance of the slope. We expect the response time to increase exponentially with the number of clients. The 0% cache hit rate follows the No Proxy baseline very close, as expected. The 25%, 50% and 75% curves all show an

improvement in the response time due to the reduction in WAN communication and reduced server load. It is interesting to note the much larger reduction seen when moving to 75% and 100%, which could indicate that very good cache decisions are needed to get anything near acceptable performance. The 100% curve shows dramatically smaller response time that increases much more slowly to only ~30 seconds for 30 concurrent clients.

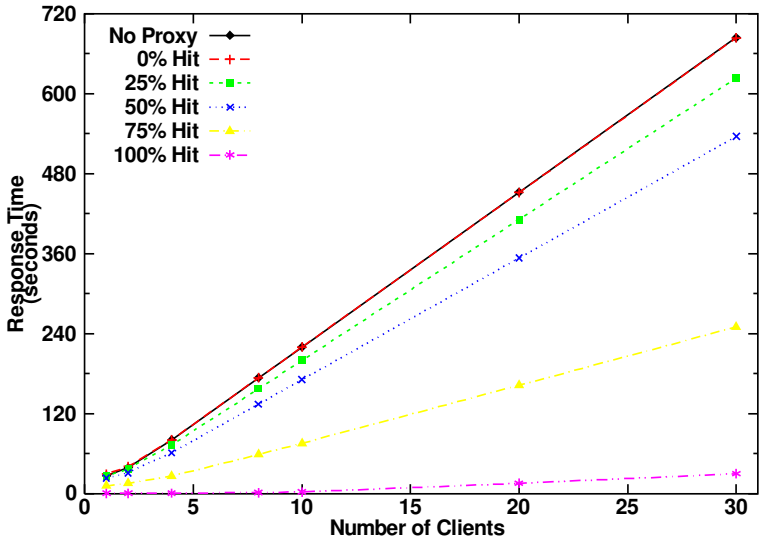

Figure 14 – Overall Client Response Time vs. Number of Clients

Figure 15 and Figure 16 show the same performance metric and data series as the previous graph, but here we examine small (300KB) and large (3MB) data requests respectively. The most striking observation here lies in the relative performance differences between the small and large cases. For the large requests, the 0%, 25% and 50% configurations all follow somewhat close to the baseline No Proxy performance. In contrast, for small requests the 50% cache hit rate response time is approximately half that of the large requests. This is due to the blocking request-response nature of the Virtual Microscope protocol, causing the bandwidth of the link to be in the critical path of every request. Thus the cache policy is more critical for large requests, due to the high latency penalty inherent in contacting the server over the WAN. Interpreted another way, if the protocol was enhanced to perform some sort of pre-fetching of data so that the long haul requests to the server was out of the critical path, it would help most for large queries.

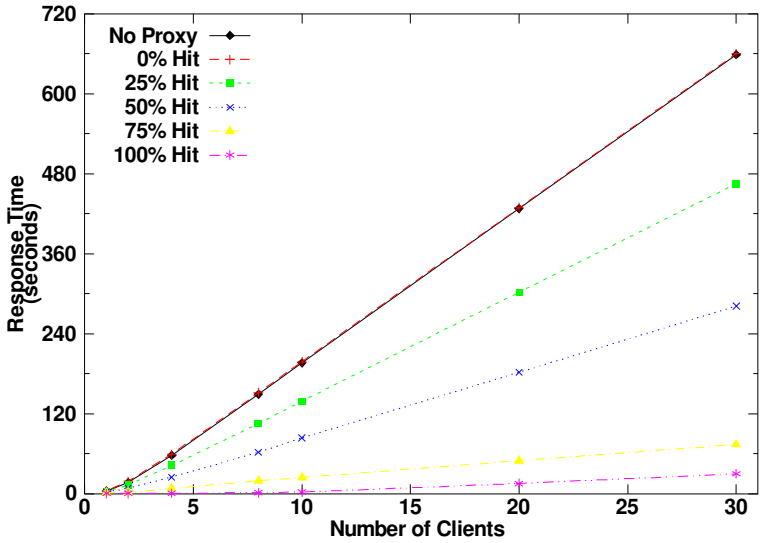

Figure 15 – Small Request (300KB) Client Response Time vs. Number of Clients

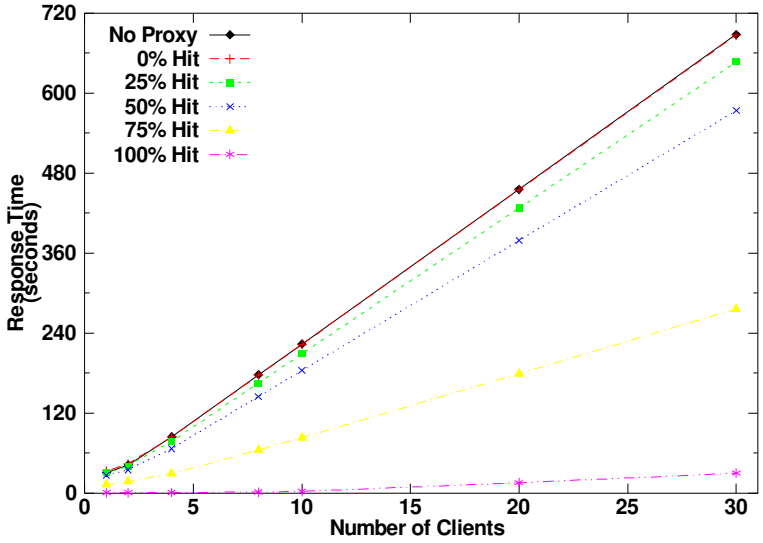

Figure 16 – Big Request (3MB) Client Response Time vs. Number of Clients

Utilization

One of the impacts a proxy can have is to reduce the load seen at the server, by serving data requests from its cache when possible. Indirectly the service time seen by requests that make it to the server is improved, due to the load reduction at the server. Thus a proxy cache hit can not only help the client who requested the data because it is available faster, but the server can better handle the rest of the client requests that miss the proxy cache, and result in server data requests for blocks.

The server utilization, or the percentage of time the server is busy, is shown in Figure 17. Utilization of zero, implies there is no work to do, and a utilization of one means the server was constantly busy. For the case of 100% proxy cache hit, no blocks will ever be requested from the server, and thus the utilization is zero. The baseline curve is the no proxy case. We see the expected curve, where it increases as clients are added to the system, until some

bottleneck is saturated, and then it levels off. Since the maximum utilization seen is about 32.5%, this would suggest that the WAN plays a large role in limiting how fast the server can receive and respond to requests. If the pipe to the server was made larger, perhaps the utilization would reach 70% or 80%, and the server could be considered a bottleneck. This result was surprising, in that we had predicted the server disk I/O as being a major contributor of request latency. The version of the server used for benchmarking was a parallel data server with multiple disks per node. One can conclude that the parallel server seems to have removed the data server from being a bottleneck in our experimental configuration. The 0% hit rate curve *should* illustrate worse behavior than the baseline no proxy case, because all that is happening is the proxy is adding overhead, and also breaking the request into requests for blocks. This is a case where our simulation model is not reflecting the real system, and will be investigated further. The other curves lie between 0% and 100% as expected. The higher the cache hit rate, the fewer the requests that actually make it to the server.

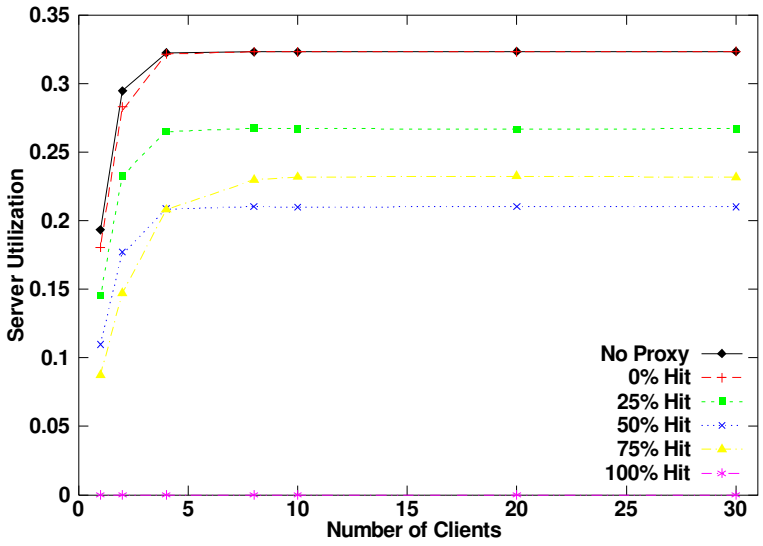

Figure 17 – Server Utilization vs. Number of Clients

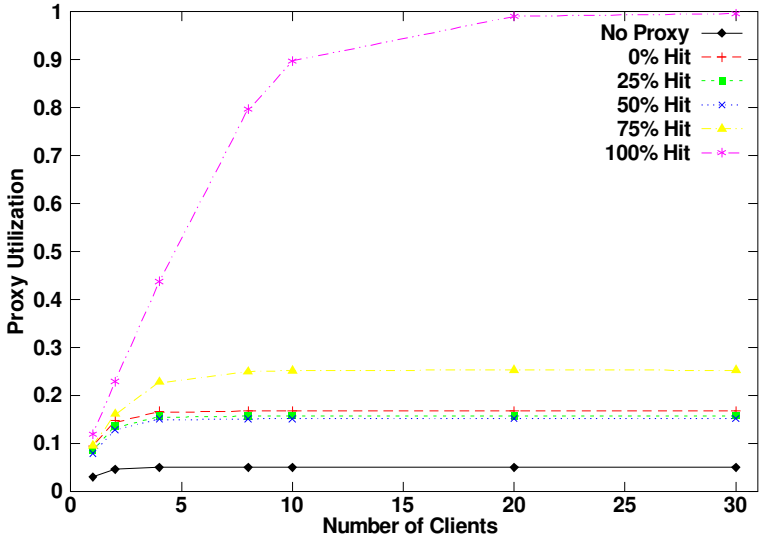

Figure 18 – Proxy Utilization vs. Number of Clients

Proxy utilization follows the expected inverse trend. For low cache hit rates, the proxy does not have much work to do. When the cache hit rate is higher, then the overhead of cache maintenance is increased, and the utilization rises. The main reason we suspect the utilization is very low for most cases except 100% cache hit rate, is due to the missing cache eviction policy in the real proxy implementation. Had this been included, the 0%,25%,50% and 75% curves would all be higher, but still far from the 1.0 mark, since much time is spent waiting on the WAN/server loop. The 100% cache hit rate curve saturates the proxy at 20 clients, and would likely be fewer if the eviction policy was represented. More clients than this would only enqueue at the proxy, and cause increased response time. Initially we had expected the number of clients handled to be higher, so this low number was surprising. An efficient cache lookup scheme based on hash tables was absolutely imperative in supporting as many clients as possible by a single proxy.

Importance of Factors

The next question to examine is now that the performance characteristics of the system are known, what is the factor that contributes most to the performance? The factors we considered for this study is *Number of Clients* and *Cache Hit Rate*. It is easily seen that a high cache hit rate and a low number of clients will correspond to high performance. Inversely, low cache hit rate and high number of clients will cause poor performance. What we want to know is what is the relative importance of the factors when considering one of our performance metrics. The method of analysis used is 2^2 factorial design, and is based on the following model:

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

This style of analysis assigns –1 to the factor level that contributes the most to a performance metric, and 1 to the factor level that contributes the least. The assignments used are listed in Table 6. Note that we chose reasonable levels for the assignments, and ignore the case where a Proxy is not used. In the case of Cache Hit Rate, 100% was merely used to show the extreme case, and is believed to not be attainable. Using 100% would skew this analysis, thus 75% is used as the highest level, and 0% is used as the lowest. We are considering three performance metrics: mean response time seen across all clients (RespTime), mean server utilization (ServerUtil) and mean proxy utilization (ProxyUtil) for all 100 trials of the given configuration. In Table 7 we list the simulated performance metrics values for the extreme cases. Table 8 presents the results of the analysis based on 2^2 factorial design. The main result lies in recognizing that for all three performance metrics we are interested in, the factor that contributes most is Number of Clients (B). Cache hit rate also contributes, but in a lesser manner. Since there are hard performance criteria in an interactive system, the choice of maximum number of clients actively making requests concurrently (30 in this case) has a large impact in which factor comes out looking the most important.

Symbol	Factor	Level –1	Level 1
A	Cache Hit Rate	75%	0%
B	Number of Clients	1	30

Table 6 – Symbol and level assignments for factors

		Response (90% confidence)		
A	B	RespTime (secs)	ServerUtil	ProxyUtil
-1	-1	11.28 ± 3.57	0.0875 ± 0.0277	0.0953 ± 0.0301
1	-1	29.29 ± 9.26	0.181 ± 0.0571	0.0938 ± 0.0297
-1	1	249.92 ± 79.03	0.232 ± 0.0733	0.252 ± 0.0797
1	1	683.71 ± 216.21	0.323 ± 0.102	0.167 ± 0.0529

Table 7 – Measured responses

	Mean Estimate	Variation Explained (%)

Parameter	RespTime (secs)	ServerUtil	ProxyUtil	RespTime	ServerUtil	ProxyUtil
q_0	243.55	0.206	0.15			
q_A	112.95	0.046	-0.022	17.38%	29.31%	11.12%
q_B	223.27	0.072	0.058	67.90%	70.68%	78.53%
q_{AB}	103.95	-0.00063	-0.021	14.72%	0.0054%	10.36%

Table 8 – Mean effects

Conclusion

The performance analysis of the Virtual Microscope represented by this paper, is an initial effort into understanding a broad class of client-server multidimensional image browsing. In this study, we examined an instance of this broad class of applications – the Virtual Microscope. First, we broke the system into discrete pieces, and characterized each piece. The characterization was done by specifically designed benchmark tests that attempt to describe the component's behavior over its domain of intended use. The next major step involved deriving a workload characterization, which was made difficult by the non-existence of many real Virtual Microscope users. Several abstractions and assumptions were made to generate the workload characterization we used to drive the rest of the study.

Given the workload and component models, we built a discrete event simulation to explore the full factorial design space. We then analyzed the simulation behavior in an attempt to better understand the real system. From this analysis we came to several conclusions. When considering the performance of the particular architecture and workload we modeled, along with the performance metrics we cared about, the following can be concluded…

1. The number of clients that we attached to a single proxy made a significant impact on the response time seen by any client. This is a very intuitive result, but turns out to be important enough to be enumerated. Care must be exercised in planning how many clients must be filtered through a single proxy.
2. The cache hit rate at a particular proxy, resulting from the request streams of all the clients connected to it, also impact performance, albeit in a less significant way than the number of clients. Although, if the number of clients is fixed, cache hit rate becomes a major source of improvement. When considering the resulting size of a client query, cache hit rate makes more of a difference with large data results than for small data results.
3. When considering the system as a whole, the major bottleneck appears to be the WAN latency and bandwidth constraints to get to the Data Server. The current parallel implementation of the server sufficiently alleviates any disk I/O problems, such that the WAN is the bottleneck. This became evident by the response time and utilization plots, and was surprising to us.
4. From the workload analysis, we found that the fine control buttons were used less than we expected. We had planned future performance optimizations that targeted frequent use of the fine control buttons, so this was a particularly useful discovery.

Future Work

As stated previously, this study is an initial effort into a more comprehensive study of this class of applications. Several future work directions are planned…

1. Expand the two-factor full factorial analysis to include all data from the 100 trials of each factor level setting. Perform more detailed analysis of the simulation output data, including analysis of variation and confidence intervals for effects. This was omitted only due to lack of time.
2. Gather workload traces from many more users. The Virtual Microscope is currently being deployed at a few research centers as a proof of concept. As people use it, we can gather user traces to learn more how people will use the system.
3. Finish the Proxy implementation, and gather traces when used in a wide area environment, as was used for this study. This validation is sorely needed.

4. As in the last point, real use validation of the studied configurations for the server and client are needed to validate those models as well. Once this is done, detailed conclusions from this performance study cannot be reliably used.
5. Expand the studied architecture configuration to include several Proxy-Client groups all hitting a single server, and some clients hitting the server without a Proxy in between. Additionally, we need to study if a hierarchy of proxies makes a difference. Since there are so many possibilities, it is unclear how to best study this design space.

Appendix

Figures representing the sampled values from the distributions presented in `Figure 11`.

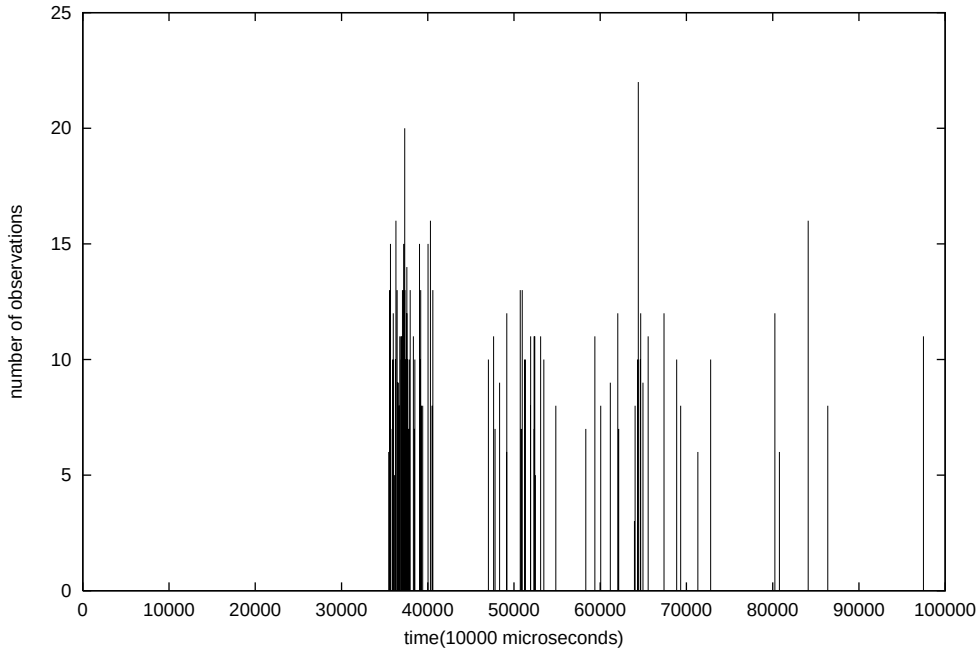

Figure 19 - Sampled values (n=1000) for roundtrip delays for 300 KB on WAN

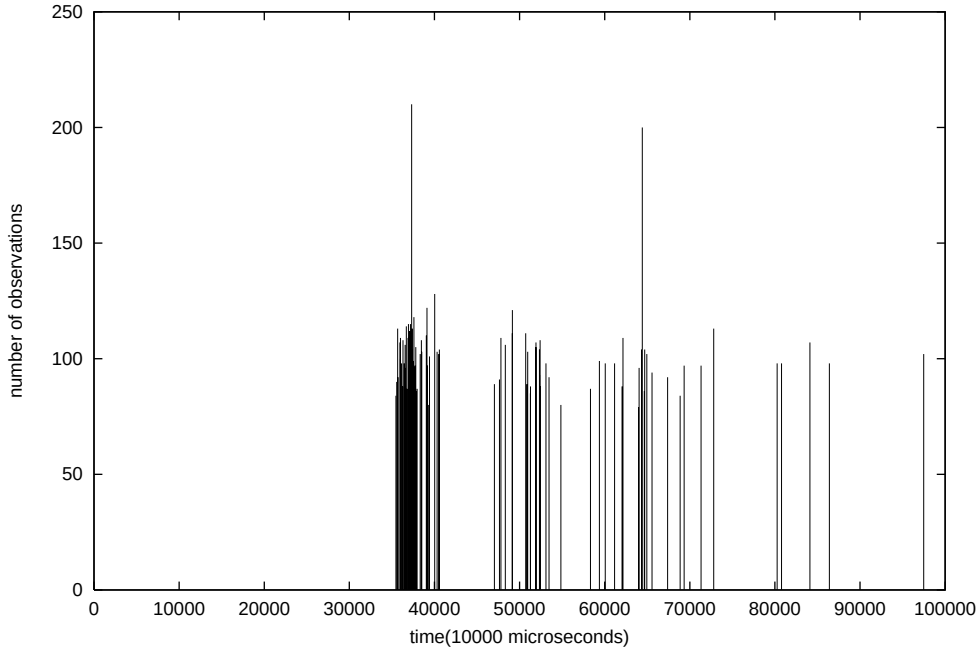

Figure 20 - n= 10000

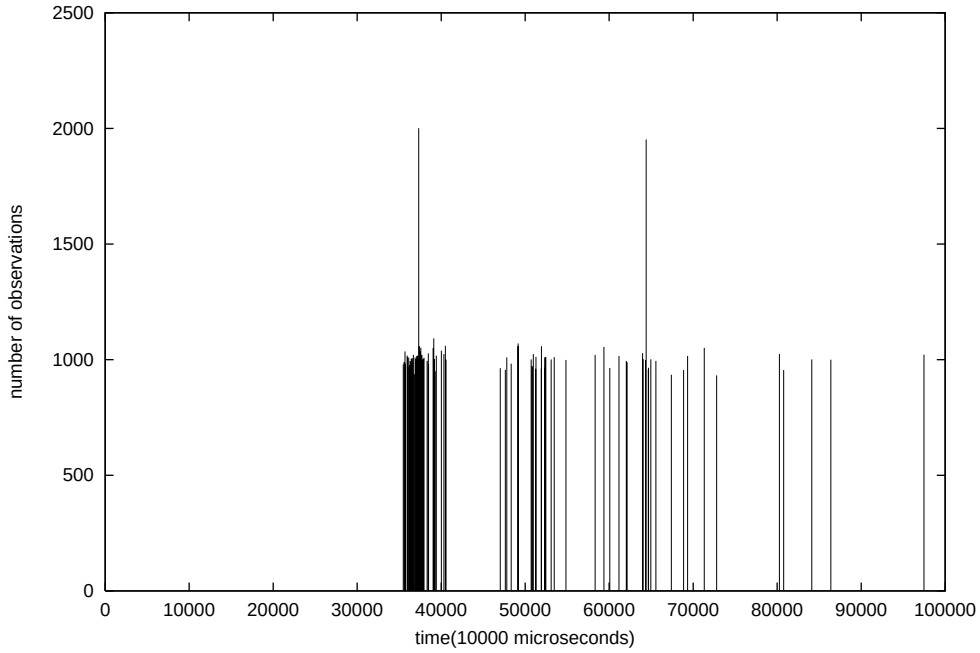

Figure 21 - n = 10000