

Performance Impact of Proxies in Data Intensive Client-Server Applications *

Michael D. Beynon, Alan Sussman, Joel Saltz

UMIACS and Department of Computer Science
University of Maryland
College Park, MD 20742
{beynon,als,saltz}@cs.umd.edu

ABSTRACT

Large client-server data intensive applications can place high demands on system and network resources. This is especially true when the connection between the client and server spans a wide-area internet link. In this paper, we describe our experience changing the typical client-server architecture for a class of data intensive applications. We show that given sufficient common interest among multiple clients, our enhancements reduce the response time per-client, the response-time variation and the amount of data sent across the wide-area link. In addition, we also see a reduction in server utilization which helps to improve server scalability.

Keywords

Wide-area, Proxy, Caching, Client-Server, Scalability.

1 INTRODUCTION

When designed for a client-server environment, image processing and image browsing applications can place high demands on the underlying system and network resources. For example, the Microsoft TerraServer archive of high resolution satellite imagery [18] currently contains about 3.5 TeraBytes of data (uncompressed) that are available for interactive browsing. When interactivity is needed and the client to server connection spans a wide-area internet connection, the demands on the network can be extremely high.

In this paper we consider the effect of changing the typical client-server architecture to include a caching proxy server in between colocated clients and the data server. With the proxy in place and given sufficient common interest among multiple clients, typical proxy benefits can be realized. The response time seen by each client can be reduced and made more deterministic, the amount of redundant data sent across the wide-area network can be reduced, and server scalability can be improved by reducing its utilization. The magnitude of any possible benefit (or degradation) is directly related to the amount of common interest among the clients and how well synchronized it is in time.

*This research was supported by the National Science Foundation under Grants #ASC-9619020 (UC Subcontract #10152408), #ASC 9318183 and #CDA9401151, by DARPA under Grant #DABT63-94-C-0049 (Caltech Subcontract #9503), and by the Office of Naval Research under Grant #N66001-97-C-8534.

1.1 Application

The Virtual Microscope [2, 11] is a client-server software system that emulates a high power light microscope. The system is required to provide interactive response times similar to a physical microscope, including continuously moving the stage and changing magnification. The client software runs on an end user's PC or workstation, while the server database software for storing, retrieving and processing the microscope image data runs on potentially remote high performance parallel computer.

The queries supported by the server are small in size, and allow a client to request a 2-D rectangular region at a particular magnification from within the bounds of a given slide data set. The reply consists of image data for the requested region. The data set size for a single slide varies, but typical images at 400X magnification approach 7GB to 10GB per focal-plane.

The Virtual Microscope is representative of a larger class of data-intensive applications [6] that involve browsing and processing large multi-dimensional datasets. Other examples include satellite data processing systems [7], and water contamination study simulators that utilize a shared database.

1.2 Challenges

Reduction of wide-area usage Wide-area applications make use of resources the user does not directly own, namely the long-haul links and intermediate nodes in the network. Currently, use of such resources is effectively free. As use increases, this cost model may very well change to one where users are charged for the amount of data sent.

Data compression works remarkably well for many domains. For most of the applications we are considering, data loss is not possible. Non-lossy compression is limited to not much more than 2-4x reduction. We believe compression cannot be the only mechanism used to reduce wide-area volume for this class of applications.

Reduced response time For interactive applications the system as a whole needs acceptably small response times. Response time is the amount of time between the initiation of a request and when the last piece of data is delivered. If a response takes too long, the application becomes unusable. Techniques are needed to reduce or eliminate the perceived latency seen between the client(s) and the server.

System scalability Most of the scalability seen in this class of applications is achieved through the use of a parallelized data server, wherein a single query is handled by many processors and disks. Reduction in workload is an important way to improve system scalability from outside the server. With less applied workload, the system should see reduced utilization, and hence better performance as more clients are added.

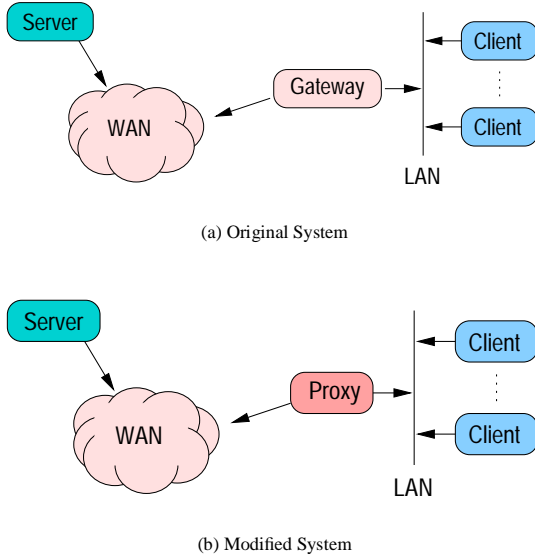


Figure 1: Architecture example for one remote server and one clustered set of clients

2 ARCHITECTURE

2.1 Original System

The original system is comprised of a server and one or more clients. The server is further decomposed into a single frontend process and one or more backend processes with attached disks. Since the server is designed to run on a parallel machine, the backend processes are intended to each run on a separate node. The application data is declustered across the backend disks. The dissemination model used is request-response (Figure 2(a)), where clients send a single multi-dimensional rectangular range query to the frontend process. The frontend reads a batch of queries, and broadcasts them to the backend processes. Given good spatial declustering, *most* backend processes will have data for a single query on one of their local disks. Each backend process with data for the batch of queries will read the data from the disks, perform computation on the data, and send it directly to the requesting clients.

We want the techniques used to apply for all instances of our class of applications. Toward this goal, we want to avoid data dependent operation where possible. We rely completely on the meta-data associated with data blocks and not the data blocks themselves for the Virtual Microscope system modifications. This meta-data describes what slide the data is from, focal plane, spatial coordinates, etc. If we wanted to allow changes in the server and client to support data block compression, opaque treatment of the data blocks reduces changes needed in the modified system code.

2.2 Modified System

The original system was designed primarily for fast data movement in the server. The proposed changes are explicitly for supporting multiple co-located remote clients.

Proxy

The major architectural change is the addition of a proxy as an intermediary in between a set of co-located clients and a remote server (Figures 1(a) and 1(b)). As for any proxy, it will appear to the server as a client, and to the clients as a server. It is possible for such a proxy to perform various functions such as caching, predictive

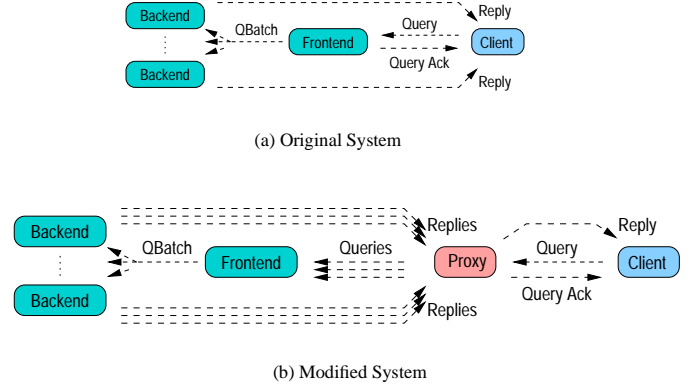


Figure 2: Client Query Control Flow

prefetching, and computation on the data. For this initial design we only consider caching.

Caching at the proxy provides a benefit when two properties are present. First, the clients need to be local to the proxy. This reduces the long latency seen in contacting the remote server to resolve proxy cache misses. Second, some commonality of interest between the set of clients must exist. This reduces the working set size in the proxy which helps avoid cache overflow. With both conditions satisfied, the maximum benefit occurs when only the first request for a block of data is not in the proxy cache.

Note that we could also consider caching at the client, which can provide many of the benefits we describe for the proxy caching. While this is a valid approach, this work attempts to isolate the case where caching happens in a proxy.

Cache Model Issues

The first consideration involves the unit of storage for the cache. The classic options here include variable or uniform sized blocks [5]. We chose to use uniform blocks for several reasons. Uniform blocks make the cache replacement decisions simpler, because the space gain for any eviction is the same. Detecting when requests have some commonality is faster with a uniform cache entry size. Given two requests R_1 and R_2 , the question becomes, is there any overlap? Intersection of rectangles is simple to compute, but deciding quickly how to handle the overlap is the problem. There could be a relatively small amount of overlap, in which case the benefit of eliminating the redundant data might not be great enough to outweigh the work to detect and decide how to handle it.

Another question is how to handle commonality in requests that are offset temporally. Should R_k be delayed for future requests, with the intention to coalesce them together into a single request? If so, how long should R_k be delayed? This approach may be good for achieving the minimum data redundancy, but it can increase response time.

Cache Model Design

Each incoming query is translated to a set of queries for individual blocks containing the query result, as seen in Figure 2(b). Blocks that are cache misses are immediately requested from the server. Blocks that are cache hits are sent to the client. Pending blocks¹ are tagged with this client's id as another recipient. When blocks

¹Pending blocks are those that have been requested from the server for a previous query but have not yet arrived at the proxy cache

eventually arrive from the server, they are sent to the set of waiting clients.

Uniform blocking helps detect commonality in a passive yet efficient manner. No request is ever delayed in the hope of finding regions in common with those of another request, yet block-level commonality is exploited. Figure 3(a) shows request R_1 arriving with an empty cache in the proxy. All blocks for the request are faulted from the server. Consider the case where blocks $\{1, 2, 3, 6\}$ have arrived at the proxy, and blocks $\{7, 8\}$ are pending when request R_2 arrives, as shown in Figure 3(b). In this case, the proxy faults blocks $\{4, 9\}$ from the server and sends blocks $\{2, 3\}$ immediately from the cache. Recognizing the commonality between these requests, the client's id is added to the pending list for blocks $\{7, 8\}$.

3 WORKLOAD

We instrumented one of the Virtual Microscope clients and captured sets of traces of trained pathologists using the system. The user examined several slides, thoroughly searching each slide for any abnormality, as they would with a real microscope. We performed tracing for several sessions, and derived an abstract model of user behavior. We present highlights of the detailed model [4] here.

| Zoom | P(400X) | P(200X) | P(100X) | P(50X) |
|------|---------|---------|---------|--------|
| 400X | 0.65 | 0.35 | 0.00 | 0.00 |
| 200X | 0.02 | 0.68 | 0.18 | 0.12 |
| 100X | 0.00 | 0.05 | 0.50 | 0.45 |
| 50X | 0.15 | 0.04 | 0.08 | 0.73 |

(a) Choose next magnification

| Zoom | P(3MB) | P(300KB) |
|------|--------|----------|
| 400X | 0.99 | 0.01 |
| 200X | 0.99 | 0.01 |
| 100X | 0.92 | 0.08 |
| 50X | 0.70 | 0.30 |

(b) Choose next request size

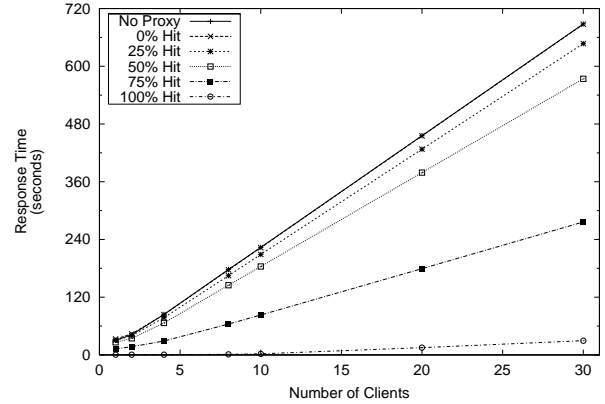
Table 1: Workload model probability transition matrices

The available magnifications for our test slide data set ranged from $50X$ to $400X$. Through analysis of the traces, we discovered that the previous magnification selected heavily influences the magnification to be chosen next. Table 1(a) shows the probability matrix used for choosing a magnification, based on the current magnification.

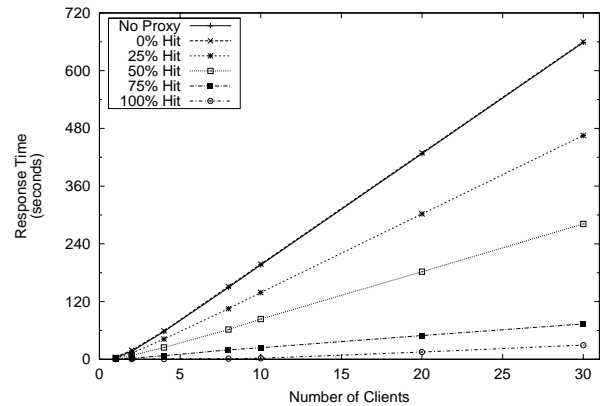
The client generates two request sizes. The larger request size is for the full view screen, which for a default window size is approximately $3MB$. The smaller request size results from the fine control buttons in the client that allow for incremental movement in any direction, which generate a request of approximately $300KB$. Table 1(b) describes the probabilities of the next request size, based on the current. Similarly, magnification also worked well as a predictor for inter-request time, also known as *think time*.

4 SIMULATION

We simulated the system to quantify the potential benefits of using a proxy. Here we present highlights of a more complete simulation study [4].



(a) Large Replies (3MB)



(b) Small Replies (300KB)

Figure 4: Client Response Time as No. Clients Increased

We are interested in determining the number of clients that can be supported at varying quality of service levels. To this end, we instrumented the Virtual Microscope client and server as well as the proxy prototype. From the resulting execution traces on a server with a *single* backend process, we generated component level performance histograms, which are used to find representative distributions. These, combined with the workload model, are used to drive a discrete event-driven simulation. For network wide-area (WAN) and local area (LAN) characterizations, we ran micro-benchmarks on our local test machines and a test machine at the San Diego Supercomputing Center.

One abstraction that was necessary for simulation was to decouple cache hits from actual request regions. Requests are modeled in an opaque way, never specifying the spatial coordinates within the slide. Each request is broken into $256KB$ sized blocks, and a Bernoulli trial is used for each block based on the cache hit rate. This allows us to easily consider the effects of different workloads (cache hit rates) while keeping the volume abstract and constant.

4.1 Response Time

We consider response time separately for queries resulting in large and small replies in Figures 4(a) and 4(b), respectively. For both

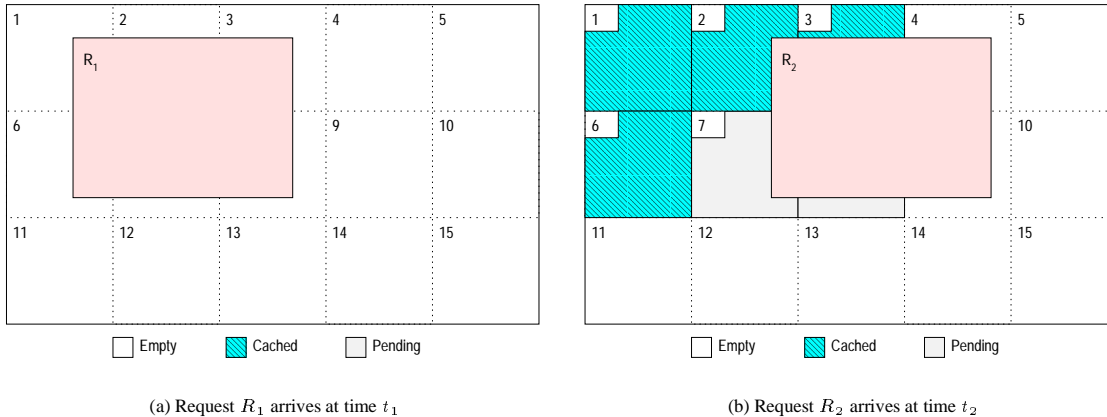


Figure 3: Proxy Cache state as requests arrive

graphs, the mean response time across all clients is shown as we scale the number of concurrent clients. The system without a proxy is shown as a baseline. The remaining curves are for configurations with the proxy turned on, with different cache hit probabilities.

For the **No Proxy** case, the response time increases almost linearly as the number of clients is increased. The 0% cache hit rate follows the **No Proxy** baseline very closely, as expected. The only overhead is the constant time for the data to pass through the proxy. The 25%, 50% and 75% curves all show improvement in the response time due to the reduction in WAN communication and reduced server load. It is interesting to note the large reduction in response time between the 75% and 100% cache hit rates, which indicates (as expected) that very good cache hit rates are critical to getting the best performance.

For large requests, the 0%, 25% and 50% configurations all somewhat closely follow 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. Larger requests cover more cache blocks, so are more likely to include the full latency cost of contacting the server over the WAN.

4.2 Utilization

Server utilization is the percentage of time the server is busy, and is shown in Figure 5(a). For the case of 100% proxy cache hits, no blocks will ever be requested from the server, so the utilization is zero. Utilization increases as clients are added to the system until saturation, and then levels off. This maximum utilization is about 32.5%. Note this is artificially low due to the method of time allocation used in our simulation². The absolute values of the utilization should be ignored, and the relative trends used to draw conclusions.

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, the overhead of cache maintenance increases, and proxy utilization rises. The 100% cache hit rate curve saturates the proxy at 20 clients. Initially we had expected the number of clients that a proxy could handle would be higher, so this low number was surprising. An efficient cache lookup scheme is absolutely imperative in supporting as many clients as possible with a single proxy.

²The time for sending and receiving data was charged to the network instead of the server.

5 PROXY IMPLEMENTATION

As described in Section 2.2, our proxy implementation caches blocks based on a uniform tiling of the data space. The proxy uses hashing internally for all cache indexing. The hash key is based on server number, slide number and data block number. This metadata provide enough information to give any data block a globally unique ID, while still allowing a fairly small key.

Every block in the cache is also indexed by other data structures to enable different cache replacement policies. At this time, we have implemented FIFO, LRU, and RANDOM policies. FIFO evicts the oldest block first. LRU maintains reference information, such that the victim is the block that has not been accessed for the longest time. RANDOM chooses a victim randomly from the set of cached blocks. For the rest of this discussion, we are using LRU replacement. We intend to investigate these replacement policies and other domain specific policies in the future. We expect policies that take advantage of semantic knowledge of the data and application to out-perform the simple policies.

5.1 Experimental Setup

Experiments were run to emulate one of our driving scenarios. This involves a classroom setting where students are remote with respect to the data. Several users start at approximately the same time, and search a slide looking for interesting regions, presumably to make a diagnosis. This is an ideal situation to experiment with a proxy, due to the geographical proximity of several users with common interest in remote data. The data server was run on an UltraSparc at the University of Maryland, and all raw image data was on local disks. This single backend configuration matches the simulation setup, and allows a manageable workload to saturate the server. Clients were run on different nodes of the 128 node IBM SP at the San Diego Supercomputing Center, over the fast wide-area vBNS [16] connection between these sites.³ When a proxy was used, it ran on another of the San Diego IBM SP nodes, with various cache and block sizes. This setup does not exactly match the architecture shown in Figure 1(b), because the proxy is not running on the network gateway machine. This results in a slight performance penalty by causing request and reply data to make two trips over the client LAN, resulting in more contention and lower performance.

We wrote a driver to emulate client behavior based on the same workload model used for simulation. The main difference is that the requests are for particular regions in the data set, and any cache

³This fast network (155 Mbit/sec ATM) was used to approach usability in response times. Had we used slower wide-area connections, the proxy benefits would increase.

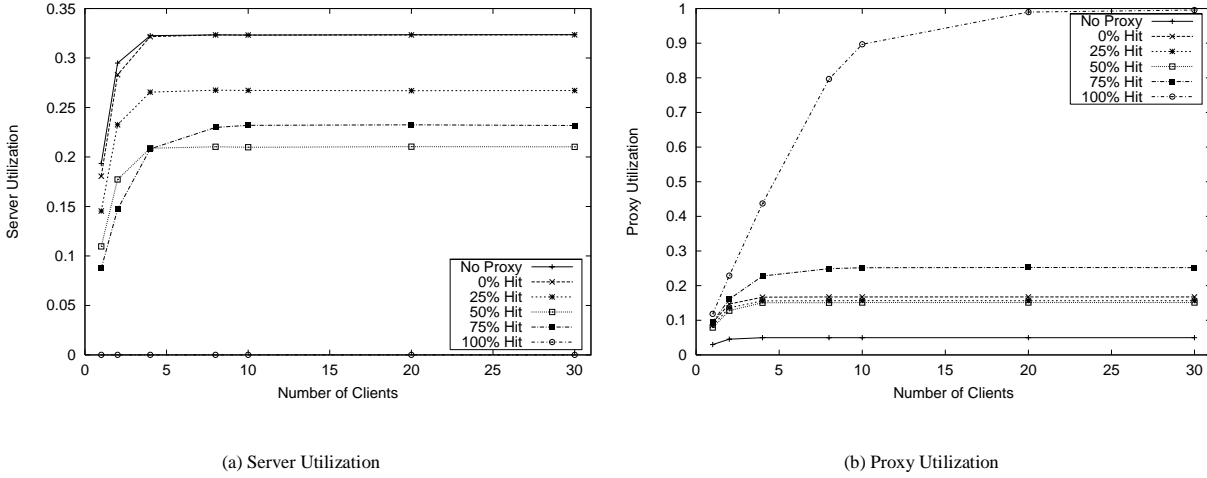
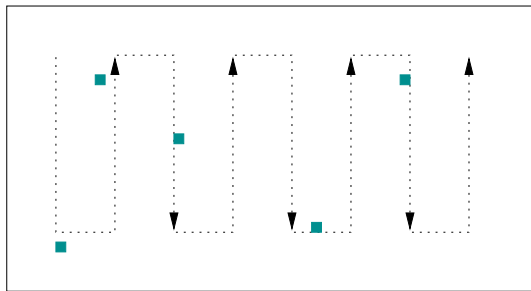


Figure 5: Utilization as No. Clients Increased



(a) Left/Right (odd clients)



(b) Up/Down (even clients)

Figure 6: Sweeping patterns over interesting points

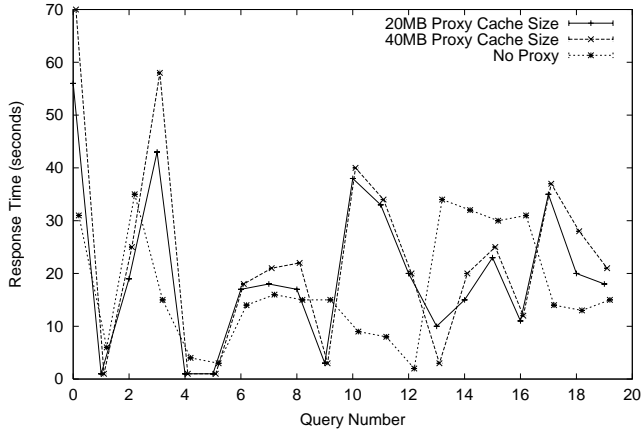
hits will be based on commonality in the actual driver requests. Interesting regions are modeled as points in the slide, and hard-coded into the driver. When a user pans *near* an interesting region, there is a high probability a request will be generated. The driver adds noise to requests to avoid multiple clients asking for the same region. In addition, we need to avoid having all the clients scan the slide in the same manner. The driver either sweeps through the slide in an up-down fashion or a left-right fashion as shown in Figure 6, as observed from real users.

5.2 Results

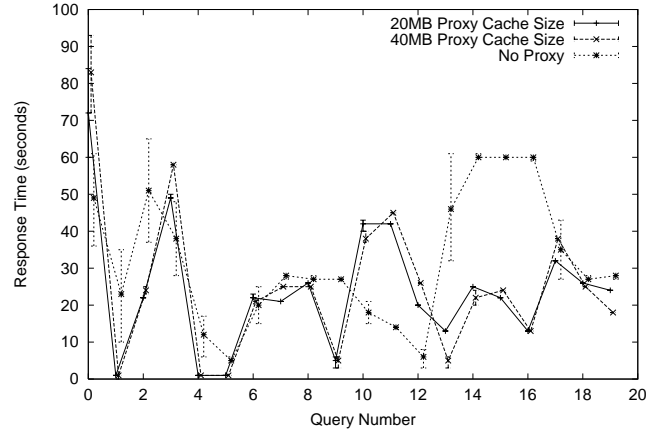
We now present a subset of the experiments we ran. In all cases, the client driver was run until a total of 20 requests were issued and satisfied per client. This results in approximately 1.6GB of raw image data being read from the disk by the backend process and 47.2MB of image data being requested for each client. The client image data size is important, because the cache sizes were chosen to avoid having everything fit in cache, thus exercising the replacement policy and biasing temporal commonality.

Figure 7 shows separate response time curves for 1, 2, 4, and 8 clients. For the 1 client case shown in Figure 7(a), the results are as expected. There is no commonality across clients when there is only a single client, so the proxy is only adding overhead by breaking a single query into several block queries, and forcing all reply data to flow through the proxy on the way back to the client. This is seen most dramatically for the first query with a proxy, where all data must be faulted from the server. Despite these problems, there are times when improvement is seen over the **No Proxy** curve. This occurs when the blocking of the slide causes extra image data to be retrieved by the proxy for a given request, and the next request is for a nearby region already in the proxy's cache. In essence, blocking the data causes some prefetching. Of course, there is a tradeoff in the choice of block size, in that larger blocks will improve the prefetching effect, but will also add to the cost of reading and sending the data. We tried block sizes of 16KB, 32KB, 64KB, 128KB, and 256KB. The 64KB block size was empirically found to be a good overall choice for the data set, workload, and machine and network configuration used.

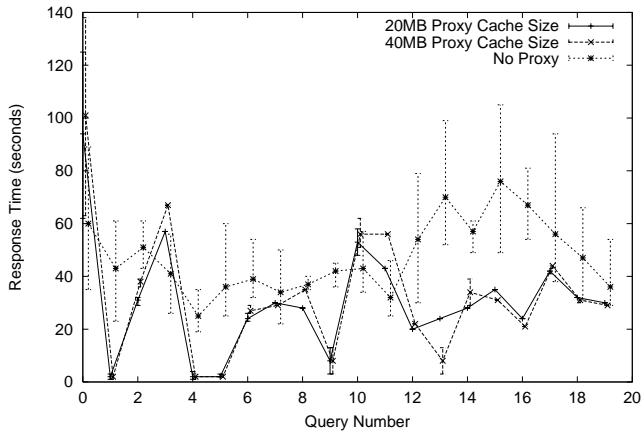
The two clients in Figure 7(b) are performing different sweep patterns, and any commonality resulting from the common set of interesting points is scattered over time, thus cache misses are common. Considering 4 clients in Figure 7(c), we introduce explicit



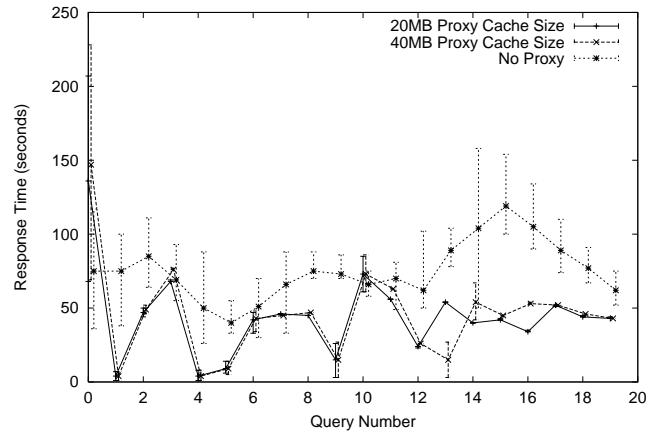
(a) 1 client



(b) 2 clients



(c) 4 clients



(d) 8 clients

Figure 7: Mean Response Time as number of clients is increased, 64KB Proxy Block Size. Error bars delimit min and max times over all clients.

temporal commonality in the client sets $\{1,3\}$ and $\{2,4\}$ from the sweep patterns. In this case, we see the response time curves for both proxy cases are generally at or below the **No Proxy** case. This trend continues in Figure 7(d) with the introduction of more clients while keeping the same two classes of sweep patterns. Another important observation applies to all cases, where more cache hits reduces the response time variation as seen by the error bars in the plots.

For all experiments, doubling the size of the cache from $20MB$ to $40MB$ does not seem to make a difference. For most cases, the curves are similar. This is probably due to the working set size being less than or approximately equal to $20MB$. Further experiments with smaller cache sizes are needed to explore cache size sensitivity.

We now consider the effect a proxy has on server utilization. As in the simulation experiments, we expect the reduction of requests that are sent to the server to reduce overall server utilization. Table 2 shows server utilization for the same set of experiments. For the single backend process configuration we used, 2 clients are enough to saturate the server. Addition of the proxy reduces the utilization

| Clients | No Proxy | Proxy (Cache Size) | |
|---------|----------|--------------------|--------|
| | | 20MB | 40MB |
| 1 | 90.89% | 85.79% | 88.57% |
| 2 | 98.11% | 84.39% | 83.31% |
| 4 | 98.35% | 85.48% | 83.92% |
| 8 | 98.94% | 86.17% | 82.01% |

Table 2: Server Utilization

overall by 2% to 12%. Another way of thinking about the effects of a proxy is the *request density*⁴ is reduced. By satisfying certain blocks in the proxy cache, the stream of block requests is spaced out, reducing request density. The server can better keep pace with the request stream it is presented, as demonstrated by the utilization numbers.

Finally we look at the reduction in wide-area communication volume for these experiments. The general trend was that the WAN

⁴Request density is defined to be the number of requests per unit time.

volume increases by about $49MB$ for each client in the **No Proxy** configuration. When a proxy is used, this reduces to about $30MB$ for both cache sizes. Seeing the same volume decrease for both cache sizes, and the small increase in communication volume as clients are added, indicates that for most cases the first request for data results in a cache miss and most subsequent requests are satisfied in the cache. Even though all data required by the clients does not fit in the cache, the current working set probably does. More cache statistics are needed to determine if this is the case. If this is true, we should either add more interesting regions to increase the working set size, add more noise to the workload or reduce the cache size to make the experiments more fair and exercise the replacement policy.

6 DISCUSSION

We can make several key observations from simulating and building a proxy for the Virtual Microscope system. Better understanding of these issues is important for improving the performance of other applications of this type.

Server Phasing

One major problem that can arise is *phasing*⁵ at the server. The frontend process reads a batch of requests up to some fixed maximum number, and broadcasts the query batch to the backend processes to be satisfied as a single unit. When completed, the backend processes are then ready to handle the next query batch.

Since the proxy rewrites a single client query into potentially many queries for individual blocks, poor phasing can occur when the set of block queries spans two or more server batches. The server is well suited for large client queries sent directly to the server, but breaks down when used for many small proxy queries.

This problem can be handled in various ways. The protocol could be augmented to include a flag that asks the frontend to continue reading client queries until the one marked *last*. Alternatively, we could add a new query packet that can hold more than one multi-dimensional range query, and the proxy would only send one multi-query to the server.

Uniform Cache Block Size

In the first proxy design, we wanted all magnification levels for a particular block of data to land in the same hash bucket. This would allow the proxy to down-sample cached block data when a request arrives for a lower resolution. Slide data opaqueness is violated, but we wanted to investigate the trade-offs. In this case, the size and number of blocks would vary based on the query magnification.

Since Virtual Microscope slides are two-dimensional, the reduction in block size is quite dramatic as magnification is reduced. For example, a $64KB$ block size at $400X$ magnification corresponds to a $64KB/4 = 16KB$ block size at $200X$, and $1KB$ at $50X$. This results in a huge number of block queries to the server for a large query at low resolution (approx. 3072), all to answer a single client query that completely misses the cache. In these cases, caching was ineffective because response time was dominated by the nearly universal need to access the server for a large number of blocks.

This also added a large amount of overhead because of smaller work unit sizes and payload sizes in the server, proxy, and even the network. Without amortizing cost over large work units, the system overhead became significant. Suffice to say, performance was a major problem when using variable block sizes in this manner. The solution we employed was to break the slide space into uniform size blocks regardless of resolution.

⁵Phasing occurs when a single unit of work spans multiple server processing loops.

Communication Protocol Issues

The communication protocol between the client and server makes a large impact on how amenable the application is to proxy-based performance improvements. Since the system uses a request-response style dissemination mechanism [12], forcing queries and responses to pass through an intermediary causes a guaranteed latency penalty. In early tests, a simple pass-thru intermediary nearly doubled latency. Any proxy for a request-response based application will see a similar effect. It would perhaps be better to use a publish-subscribe model of dissemination where clients indicate data interests ahead of time, and the server simply pushes data to the client without the extra round trip time latency penalty of request-response. Of course, this is only useful if there is a way to know what future data interests will be. For many of these applications, the proxy could monitor direction vectors from concise information provided by the client to estimate future requests, and have the server send data without the client asking for it. In this model, the clients steer the data flow, rather than asking for each piece. Wide-area applications may have to use such protocols if interactivity is needed.

We can also consider other methods to alleviate the penalty involved in sending the reply image data through the proxy before it gets to the client. IP Multicast could be used to send data simultaneously to the requesting client(s) and the proxy. In addition to reducing the response time for the client query, this would also reduce server utilization. The server only has to spend time putting the data onto the network once, and the network is responsible for replicating the data somewhere en route. The main problem with using multicast for this application is granularity of group changes, and also in the sparseness of group members in a wide-area network [9]. A large cost of multicast is building and maintaining the multicast tree as users join and leave the multicast group. This works well for long running jobs that send many messages to amortize the cost. Multicast could be prohibitively expensive for the fine granularity group changes needed to handle common block requests in the proxy. The proxy architecture would potentially require group membership changes for every block in a query, for which current multicast is ill suited.

6.1 Related Work

Proxy servers are not new, and have been studied and deployed extensively in the Web context [17, 15, 10]. Our work is different in several ways. First, the application domain is different. Web proxies deal with caching html documents, while we are caching multi-dimensional spatial data query results. Second, the cache unit is different. Web proxies cache all or none of a variable sized document [5], because the whole document is the smallest possible granularity. The equivalent for our domain would be to cache full query results. Instead, we introduce a degree of freedom by decomposing the data space, and translate the query into several uniform sized sub-queries in the decomposed domain. Third, the reason for the cache being effective is subtly different. Web proxies work because a subset of all documents are requested more frequently than others. In our case, the proxy is effective because there is *some* overlap between queries. In fact, we would argue our proxy is more similar to cpu cache lines than Web proxies.

This work has some features similar to Semantic Data Caching [8]. We take advantage of the semantic knowledge available about the datasets, about the relationship between the datasets and their corresponding meta-data, and also about the coordinate systems and spatial relationships present in such multi-dimensional datasets. This knowledge is leveraged to exploit the temporal and spatial locality in the query streams.

To reduce latency in processing queries into relational databases, Query Scrambling [1, 3] is a technique that changes the query plan at runtime to do useful work with other data sources

when initial delays are seen from some data sources. We are taking advantage of spatial locality in a specific type of non-relational data. Scrambling is dealing with a different problem on a different type of data.

Broadcast Disks [13] tries to alleviate request-response protocol problems by pushing commonly used data onto a constantly broadcast circular stream. This requires techniques to decide when a client should wait for the next occurrence of needed data on the “disk”, or request it directly from the server. This work is one extreme in handling client-server data needs. The other extreme is unicast request-response, which is what the original Virtual Microscope system represents. The proxy version is somewhere in between, utilizing intermediate nodes in the path from the client to the server to reduce redundancy. The use of intermediate nodes is a primary distinction between our work and most related database research. Tradeoffs in using client and server resources have been considered for some time [14].

7 CONCLUSION

We have shown how the addition of an intermediary (proxy) in between the clients and server of a particular class of data intensive applications can improve performance. Given sufficient commonality among a set of clients local to the proxy, we have demonstrated utility by reducing overall system response time and making it more deterministic, and increasing server scalability by decreasing server utilization. In addition, we have reduced the amount of data sent over the wide-area connection between the server and the clients. Even in cases with no inter-client commonality, we see some prefetching improvements for a single client. These benefits were achieved without changing the existing client or server application code.

To be fair, there are cases when commonality between client requests does not exist, or the commonality is so temporally distributed that it cannot be exploited. We plan to follow up this work by investigating techniques for deciding when the use of a proxy will be advantageous. If this can be determined, the proxy could be automatically used for cases where it improves performance, and avoided when it does not.

This entire work assumes read-only browsing of the dataset. We have not dealt with the important problems of cache consistency when allowing updates to the dataset. Future work involves allowing such updates to occur.

8 ACKNOWLEDGMENTS

We would like to thank Anurag Acharya for many early discussions. We would also like to thank Renato Ferreira and Asmara Afework for their work on the simulation study and prototype servers, clients and drivers. John Davis was very helpful in reading and helping revise presentation of this paper.

REFERENCES

- [1] S. Acharya, M. Franklin, and S. Zdonik. Scrambling query plans to cope with unexpected delays. In *Proceedings of the 1997 ACM-SIGMOD Conference*, Tucson, AZ, May 1997.
- [2] A. Afework, M. D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital dynamic telepathology - the Virtual Microscope. Technical Report CS-TR-3892 and UMIACS-TR-98-23, University of Maryland, Department of Computer Science and UMIACS, Mar. 1998. Appears in Proceedings of the 1998 AMIA Fall Symposium.
- [3] L. Amsaleg, P. Bonnet, M. Franklin, A. Tomasic, and T. Urhan. Improving responsiveness for wide-area data access. *IEEE Data Engineering Bulletin*, 20(3), Sept. 1997. <http://www.cs.umd.edu/users/franklin/debull/amsaleg.ps>.
- [4] M. D. Beynon, R. Ferreira, A. Afework, and G. K. Mohan. Performance evaluation of client-server architectures for large-scale image-processing applications. Technical Report CS-TR-3970 and UMIACS-TR-98-17, University of Maryland, College Park, MD, USA, December 1997.
- [5] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8-11, 1997*, pages 193-206, Berkeley, CA, USA, 1997. USENIX.
- [6] C. Chang, R. Ferreira, A. Sussman, and J. Saltz. Infrastructure for building parallel database systems for multi-dimensional data. In *Proceedings of the Second Merged IPPS/SPDP (13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing)*. IEEE Computer Society Press, Apr. 1999. To appear.
- [7] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high performance remote-sensing database. In *Proceedings of the 1997 International Conference on Data Engineering*, pages 375-384. IEEE Computer Society Press, Apr. 1997.
- [8] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proceedings of the 22nd VLDB Conference*, pages 330-341. Morgan Kaufmann Publishers, Inc., 1996. <http://SunSite.Informatik.RWTH-Aachen.DE/dblp/db/conf/vldb/DarFJST96.html>.
- [9] S. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. The PIM architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2):153-162, April 1996.
- [10] F. Douglass, A. Haro, and M. Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS-97)*, pages 83-94, Berkeley, December 8-11 1997. USENIX Association.
- [11] R. Ferreira, B. Moon, J. Humphries, A. Sussman, J. Saltz, R. Miller, and A. Demarzo. The Virtual Microscope. In *Proceedings of the 1997 AMIA Annual Fall Symposium*, pages 449-453. American Medical Informatics Association, Hanley and Belfus, Inc., Oct. 1997.
- [12] M. Franklin and S. Zdonik. Dissemination-based information systems. *IEEE Data Engineering Bulletin*, 19(3):20-30, September 1996.
- [13] M. Franklin and S. Zdonik. Data in your face: Push technology in perspective (invited paper). In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD98)*. ACM Press, June 1998. <http://www.cs.umd.edu/projects/bdisk/inyourface.ps>.
- [14] M. J. Franklin, B. T. Jonsson, and D. Kossman. Performance trade-offs for client-server query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD96)*. ACM Press, June 1996.
- [15] A. Iyengar and J. Challenger. Improving Web server performance by caching dynamic data. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS-97)*, pages 49-60, Berkeley, December 8-11 1997. USENIX Association.
- [16] J. Jamison, R. Nicklas, G. Miller, K. Thompson, R. Wilder, L. Cunningham, and C. Song. vBNS: not your father's Internet. *IEEE Spectrum*, 35(7):38-46, July 1998.
- [17] T. M. Kroeger and D. D. E. Long. Exploring the bounds of Web latency reduction from caching and prefetching. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8-11, 1997*, pages 13-22, Berkeley, CA, USA, 1997. USENIX.
- [18] Microsoft Corp. Microsoft TerraServer. <http://www.terra-server.microsoft.com>, 1998.