CS-TR-3951 and UMIACS-TR-98-70

Performance Impact of Proxies in Data Intensive Client-Server Parallel 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

November 18, 1998

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 consider changing the typical client-server architecture of a class of data intensive applications. We show that given sufficient common interest among multiple clients, our enhancements reduce the response time per-client and reduce 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 as more clients are added to the system.

1 Introduction

Image processing and image browsing are very popular resource intensive applications, spanning many disciplines from Atmospheric Science to Clinical Pathology. When designed for a client-server environment, these 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 propose a modification to the typical client-server architecture to include a caching proxy server in between the client(s) and the data server. With the proxy in place and given sufficient common interest among multiple clients, several benefits can be realized. The response time seen by each client can be reduced, the amount of redundant data sent across the wide-area network can be reduced, and server scalability can be improved by reducing its utilization. On the other hand, the magnitude of any benefit provided by the proxy server is directly related to the amount of common interest among the clients and how well synchronized they are in time. While the current main target of the proxy server is applications that perform

*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.

interactive scanning through two- and three-dimensional datasets, we now discuss a more general class of data intensive applications in more detail.

1.1 Data intensive applications

We have been building infrastructure for constructing parallel database systems, called T2 [8], that enables integration of storage, retrieval and processing of multi-dimensional datasets. T2 provides support for common operations including index generation, data retrieval, memory management, scheduling of processing across a parallel machine and user interaction. It achieves its primary advantage from the ability to seamlessly integrate data retrieval and processing for a wide variety of applications and from the ability to maintain and jointly process multiple datasets with different underlying grids. Most other systems for multi-dimensional data have focused on uniformly distributed datasets, such as images, maps, and dense multi-dimensional arrays. Many real datasets, however, are non-uniform or unstructured. T2 can handle both uniform and non-uniform datasets.

T2 has been developed as a set of modular services. Since its structure mirrors that of a wide variety of applications, T2 is easy to customize for different types of processing. To build a version of T2 customized for a particular application, a user has to provide functions to pre-process the input data, map input data to elements in the output data, and aggregate multiple input data items that map to the same output element.

Data intensive applications that are currently being implemented using the T2 services include satellite data processing systems [9], analysis of digitized light microscopy data [2, 12] and water contamination studies that couple multiple physical process simulators through a shared database [10, 7]. A common thread through all these applications is that their datasets have an underlying multi-dimensional attribute space. The data dimensions can be spatial coordinates, time, or varying experimental conditions such as temperature, velocity or magnetic field. We will be using the Virtual Microscope, a system for browsing light microscope images, as an example T2 system throughout this paper. We therefore now describe the Virtual Microscope system in more detail.

1.1.1 The Virtual Microscope

The Virtual Microscope [2, 12] is a client-server software system that emulates a high power light microscope. The system is required to provide interactive response times for standard physical microscope behavior, including continuously moving the stage and changing magnification and focus. 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 a high performance parallel computer, potentially at a remote site.

The queries supported by the server are small in size, and allow a client to request a rectangular region at an available resolution from within the bounds of a given slide data set. The reply consists of image data for the requested region and can be 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.

The greatest difficulty in implementing the Virtual Microscope system is dealing with the extremely large quantities of data representing a collection of slides. For example, using the digitizing microscope currently available at Johns Hopkins Hospital, a single spot at a magnification of $400X$ 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 over $7GB$. 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.

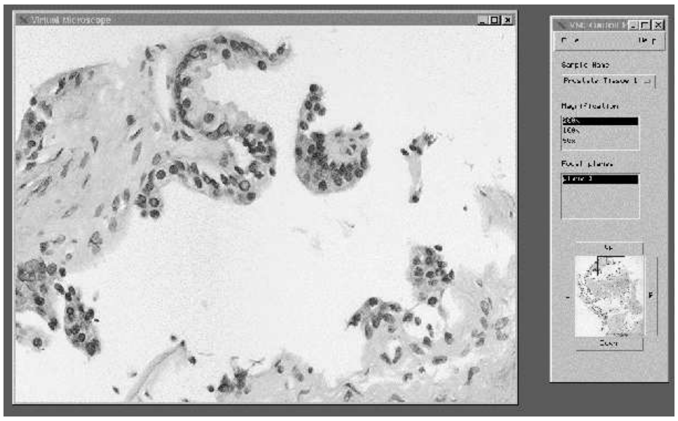


Figure 1: Virtual Microscope client browsing a slide

1.2 Challenges

Reduction of wide-area usage Traditional applications were typically confined to a single machine, whether it be a single processor workstation or a multi-node parallel machine. Such environments have many nice properties. The machines are all trusted, the interconnection network between machines is fast and fairly reliable, and all the machines are under the control of a centralized administrative entity. As internet usage increases and users are envisioning applications that collectively employ many machines connected via a wide-area network, this nice environment changes in many ways. Trust may not exist, the fast reliable connections are replaced by slower less reliable links, and there are many administrative domains. In addition, the ability to have strong consistency in the presence of updates may not be feasible.

A subtle (but important) difference is seen when considering who owns the resources used by an application. Traditional applications use resources in machines and networks owned by the organization running them. For wide-area applications, even if all the machines where computation is performed are owned, the wide-area links and intermediate hops are usually not. If more wide-area network resources are used by a particular application, this could degrade the performance of other wide-area applications. In the current wide-area network environment, the cost to send data is effectively free. As usage increases, this cost model may very well change to one where users pay in some manner for the amount of data sent. For example, Sprint has recently unveiled a new nationwide network that supports voice and data over a single line, with a cost model that eliminates the concept of long distance. Instead, users are charged by the amount of data actually sent. Both the network good citizen and cost-per-byte-transmitted arguments motivate the need for techniques to reduce wide-area bandwidth consumption.

Reduced response time For interactive applications, including those that use the T2 infrastructure, the system as a whole needs acceptably small response times for it to be usable. 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, it simply will not be used, so performance is critical for acceptance. Techniques are needed to reduce or eliminate the perceived latency seen between the client and the server. In addition, the server should be able to handle multiple simultaneous clients without prohibitively degrading response time.

System scalability A system that scales well, should *gracefully* degrade as more clients are added to the system. We expect performance to suffer, but it should be closer to following a linear curve rather than exponential. Most of the scalability seen in T2-based systems 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 the only 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.

1.3 Overview

The approach we are studying in this paper has many features similar to Semantic Data Caching [11]. We are taking advantage of the semantic knowledge available about T2 datasets, about the relationship between T2 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 perform data specific performance optimizations.

In the next section we describe the existing T2 system architecture, and present our changes. Next we show how interactive users are modeled for a set of experiments. Then simulation and implementation results are given for the Virtual Microscope system, followed by interpretation and conclusions.

2 Architecture

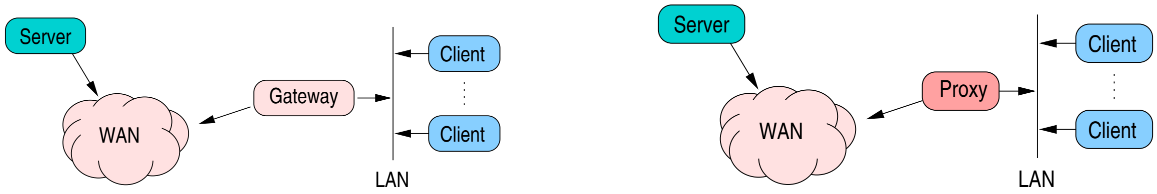
2.1 Original System

The original T2 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 of the parallel machine. The application data is declustered across the backend disks. The dissemination model used is request-response. 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. Most backend processes will have data for a given query on one of their local disks, given good spatial declustering. Each backend process with data for the batch of queries will read the data from the disks, process it as customized for the T2 application, and send it directly to the requesting clients.

The original system was designed primarily for fast data movement in the server. The proposed changes to the system are explicitly for supporting multiple co-located remote clients, where remote means the client and server are separated by a wide-area network connection. We want to find ways to reduce the impact on the wide-area network by eliminating redundant requests for the same data, reduce the perceived response time problem, and make the system scale more gracefully, as discussed in Section 1.2.

The T2 design already existed when we were considering alternate architectures, with multiple clients written and both sequential and parallel versions of the server implemented. There was also a well-defined communication protocol between the clients and the server. With an eye toward interoperability and minimization of total system changes, a major design goal was to change the existing system as little as possible. In certain places we were successful, and in other places, changes were necessary.

We want the techniques used to be generally available for (at least) all T2 applications. Toward this goal, we want to avoid needing to know what the structure of the data is. For example, for the Virtual Microscope system implemented with T2, all image data blocks are tagged with meta-data describing what slide the data is from, which focal plane it is from, what its spatial coordinates are, etc. All the meta-data is essential, but the actual image data structure should be ignored, and treated as black box data. A black box approach is important for various reasons, for example to allow for compression of data by the server and decompression by the clients, which could only be supported through the opaque treatment of data in the modified architecture.



(a) Original System (b) Modified System

Figure 2: Architecture for one remote server and one clustered set of clients

2.2 Modified System

Proxy

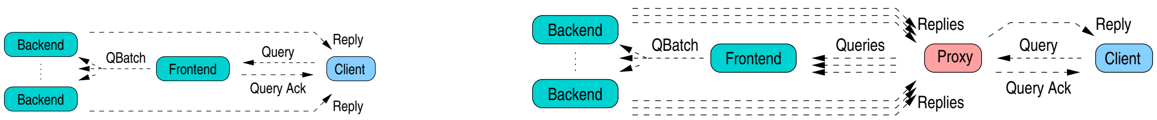
The major architectural change is the addition of a proxy as an intermediary in between a set of co-located clients and a remote server. Figures 2(a) and 2(b) show this scenario for both architectures. The proxy will appear to the server as a client, and to the clients as a server. It is possible for such a proxy to perform various functions such as caching, predictive prefetching, and computation on the data. For this initial design we only consider caching. This is primarily due to the fact that more advanced functionality would require changes in the server and client, thus breaking one of our original design goals.

Caching at the proxy provides a benefit when the clients are local to the proxy and there is some degree of commonality of interest between the set of clients. Ensuring the proxy is local to the clients reduces the long latency seen in contacting the remote server to request data if the needed data is in the proxy's cache. Having sufficient commonality among the client requests is important in reducing the working set size such that the cache does not overflow. With both these conditions satisfied, the best case is when the first request for a block of data is a cache miss in the proxy, causing it to be faulted in by the proxy with a request to the server. All subsequent client requests will hit in the cache, and be answered quickly over the local connection to the proxy.

Cache Model

The use of a proxy makes requests that miss the cache slower than when the proxy is not used. This can be minimized by making the cache lookup as fast as possible, thus speed is the main design goal in laying out the cache structure.

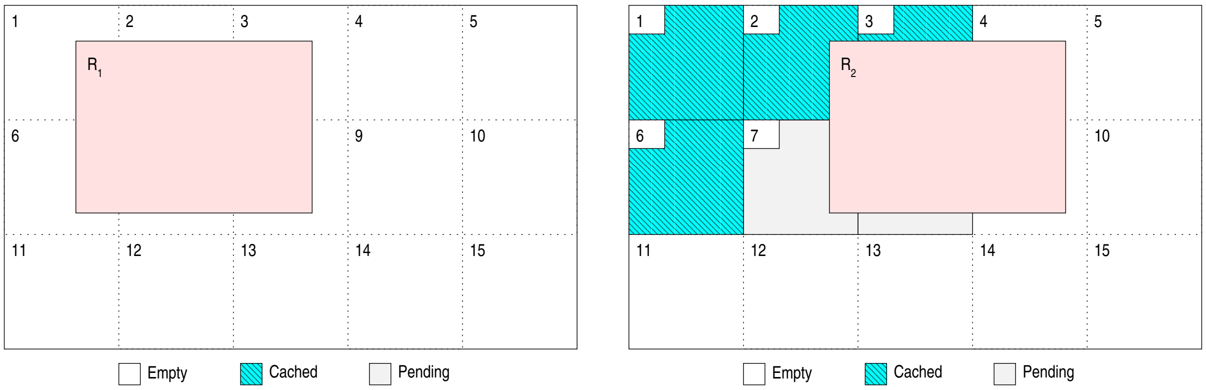
The first decision involves the unit of storage for the cache. The classic options here include variable or uniform sized blocks [6]. We chose to use uniform blocks for several reasons. Uniform blocks make the cache replacement decisions simpler. This occurs when the cache is full, and some new data arrives and needs to be cached. Picking a victim is easier when all cache entry sizes are the same since the space gain for evicting any block is the same. Another reason supporting uniform cache entry size is related to detecting when requests have some commonality. Given two requests R_1 and R_2, the question becomes, is there any overlap? Intersection of rectangles is simple to compute, since we know the orientation of the rectangles is guaranteed to be aligned with the x and y axes. What becomes harder to do quickly is determining the best way to handle the commonality. There could be a relatively small amount of overlap, in which case the benefit of eliminating the redundant data being requested from the remote server might not be great enough to outweigh the work to detect 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 reducing the volume



(a) Original System

(b) Modified System

Figure 3: Client Query Control Flow



(a) Request R_1 arrives at time t_1

(b) Request R_2 arrives at time t_2

Figure 4: Proxy Cache state as requests arrive

of redundant wide-area data sent, but it sacrifices response time.

Taking into consideration these and various other issues, we chose to break up the data space into uniform size blocks. Each incoming query is mapped to the set of blocks containing the query result, as seen in Figure 3(b). Blocks that are cache misses are immediately requested from the server. Blocks that are cache hits are sent to the set of clients waiting for them. Pending blocks (those that have been requested from the server but are not yet in the cache) are tagged with the client as being another recipient. Misses are handled first, since they will take the longest to ultimately arrive at the client. Uniform blocking of the slide image also helps in detecting 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 taken advantage of. Figure 4(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 4(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 proxy does not redundantly fault in blocks $\{7, 8\}$ from the server because they have already been requested.

3 Workload

While the Virtual Microscope T2 application we used for our experiments is becoming mature and is starting to be used routinely by pathologists, the current version is still a research prototype, which makes workload characterization difficult for even a single client. We instrumented one of the Virtual Microscope client programs and captured sets of 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 real microscope. We performed tracing for several sessions, and derived an abstract model of user behavior. We are actively encouraging use of the system and collecting more traces to refine our workload model.

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 were $50X$, $100X$, $200X$, and $400X$. 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 sizes of requests. 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 $300KB$. We expected magnification to be a predictor variable for data size, since at lower magnification the fine control buttons would not be used, and at high magnification fine control would sometimes be used. Instead, the pathologists who generated our traces never use the fine control. Still, we felt this may be atypical behavior, and included small non-zero probabilities for using fine control, as shown in Table 1(b).

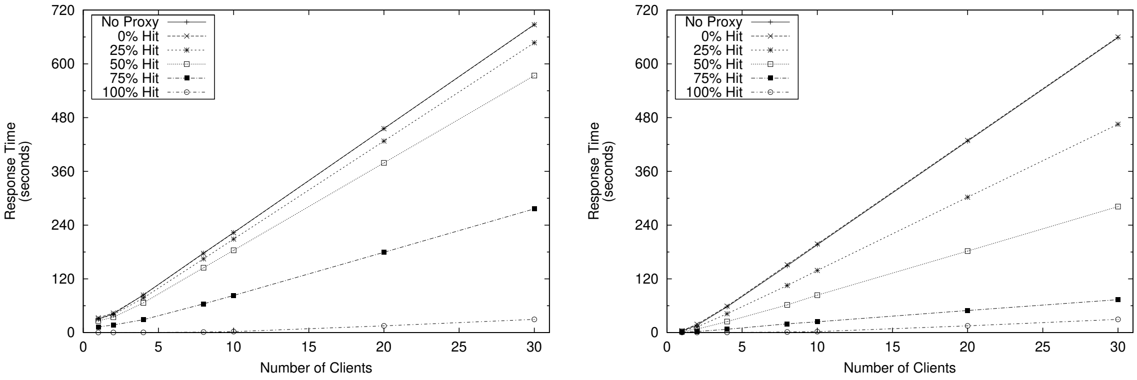
Next we consider inter-request time, also known as *think time*. Here again, magnification worked well to predict the amount of think time after a given request. Since the distribution of the times was markedly different depending on the magnification, we chose not to try to find compact representative distributions to generate the inter-request times. Instead, we randomly choose values from a histogram of the values in the traces.

4 Simulation

We simulated the system to validate our hypotheses and to quantify the potential benefits of using a proxy. Here we present highlights of a more complete simulation study [5].

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 prototype of the proxy. 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 between our local test machines and a test machine at the San Diego Supercomputing Center. These histograms of application level data transfer were abstracted into distributions for the local case, but the wide-area case was not as easily represented, so we use the raw histograms for wide-area simulation.

The factors and levels we used for simulation are listed in Table 2. 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,



(a) Large Replies (3MB)

(b) Small Replies (300KB)

Figure 5: Client Response Time with Increasing Number of Clients

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 cache hit rates while keeping the workload volume abstract and constant.

Location	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

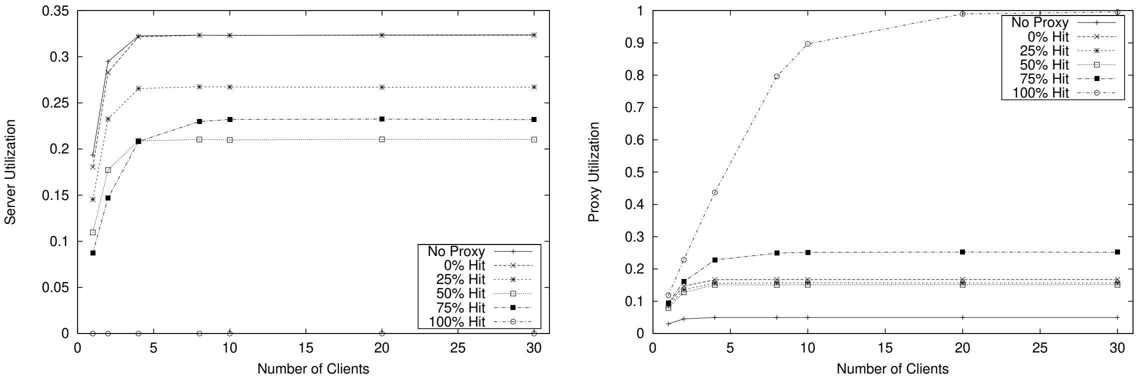
Table 2: Factors and Levels for Simulation

4.1 Response Time

The response time we care about is the time between the client request and when the client receives all the data for that request. We consider response time separately for queries resulting in large and small replies in Figures 5(a) and 5(b), respectively. For both graphs, the mean response time across all clients is shown as we scale the number of clients in the system. The system without a proxy is shown as a baseline, where the proxy machine would be a gateway merely passing packets between the client and server. 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. This is expected to not be completely true for the real system due to congestion effects we did not model. Our network model only insures that a single entity will be writing to the network at a given point in time. 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 that very good cache hit rates are critical to getting acceptable performance.

Next we consider the relative performance differences between the small and large cases. 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, making the cache policy more important for large requests. Interpreted



(a) Server Utilization

(b) Proxy Utilization

Figure 6: Component Utilization with Increasing Number of Clients

another way, if the request protocol was enhanced to perform some sort of pre-fetching of data so that long haul requests to the server are removed from the critical path for client response time, such an optimization would help most for large queries.

4.2 Utilization

Another benefit of cache hits in the proxy is to reduce the load seen at the server. This indirectly improves service time of requests handled by the server when server utilization is close to capacity. Server utilization is the percentage of time the server is busy, and is shown in Figure 6(a). For the case of 100% proxy cache hits, no blocks will ever be requested from the server, so the utilization is zero. The baseline curve is the **No Proxy** case. Utilization increases as clients are added to the system until saturation, and then 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. We expected the server to be more of a limiting factor. The version of the server used for benchmarking was a parallel data server with multiple disks per node, and we conclude that this prevents the server from being a bottleneck. The 0% hit rate curve should illustrate worse behavior than the baseline **No Proxy** case, because the proxy is adding overhead and is breaking the request into multiple requests for blocks. This is a case where our simulation model does not reflect the real system perfectly. The other curves lie between 0% and 100% as expected. The higher the cache hit rate, the fewer the number of requests that actually reach the server.

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 main reason we suspect the utilization is very low for all cases except the 100% cache hit rate, is due to the way time is charged in our simulation. Only the WAN is charged during contention, so the proxy appears idle while sending data. Had this time been charged to the proxy, the 0%, 25%, 50% and 75% curves would all be higher, but still far from full utilization, since much proxy time is spent waiting in the WAN/server loop. The 100% cache hit rate curve saturates the proxy at 20 clients, and would likely be fewer if the charging of time was changed. 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.

5 Prototype Proxy Implementation

As described in Section 2.2, our proxy implementation for T2 caches blocks based on a tiling of the data space. The simulation results indicated that fast operation in cache lookups are imperative for proxy scaling with increasing clients, and overall fast operation in translating queries to block queries is required to avoid increasing client response time. In response to these requirements, we designed the prototype proxy to internally use hashing for cache indexing. The hash key for Virtual Microscope data was based on the server number, slide number and data block number. These attributes provide enough information to give any data block a globally unique ID, while still allowing a fairly small key. Virtual Microscope data blocks are uniform in size, regardless of magnification level.

Every block in the cache is also indexed by other data structures to enable different cache replacement policies. At this time, we have added FIFO, LRU, and RANDOM policies. Briefly, FIFO keeps a list of cached blocks. New blocks are appended to the tail of the list, and the blocks chosen to be the victim for cache eviction when the cache is full is the block at head of the list. 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 use the LRU replacement policy. We intend to investigate these standard replacement policies and other domain specific policies in the future.

5.1 Experimental Setup

Experiments were run such that one of our driving scenarios for the Virtual Microscope system is emulated. 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 is run on an UltraSparc at the University of Maryland, and all raw image data was local to this machine. 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.[1] 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 2(b), because the proxy is not running on the network gateway machine, and 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 slower performance. Use of a gateway is ideal in this respect, although not practical for many situations.

(a) Left/Right (odd clients) (b) Up/Down (even clients)

Figure 7: Sweeping patterns over interesting points

[1]This fast network (155 Mbit/sec ATM) was used to approach usability in response times. Had we used traditional wide-area connections, the gain in using a proxy would be even better.

We use the same workload model for these experiments as we did for the simulations. This was accomplished by writing a driver to emulate client behavior based on the workload model. The client driver sends requests, accepts data and sits idle for some amount of think time in between requests. Interesting regions are modeled as points in the slide. When a user pans across an interesting region, there is a high probability a request will be generated. When the driver generates client events, some noise is added to avoid multiple clients asking for the same exact region. In addition, we need to avoid having all the clients scan the slide in the same manner. While this may be the best case for any caching mechanism, it is unrealistic. The driver either sweeps through the slide in an up-down fashion or a left-right fashion as shown in Figure 7, as was observed from real microscope users. The interesting regions were chosen once, and hard coded into the client driver.

5.2 Results

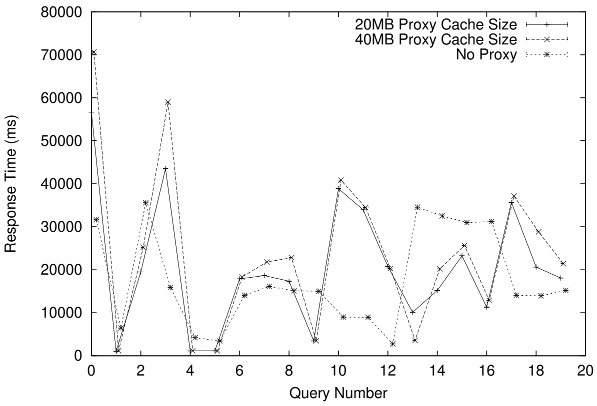
We now present a subset of the experiments we ran. In all cases, the client driver was run until a total of 20 requests were issued per client. This results in approximately $1.6GB$ of raw image data being read from the disk by the backend process and $47.2MB$ of image data being sent over the network 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. This choice biases the importance of temporal commonality.

Figure 8 shows separate response time curves for 1, 2, 4, and 8 clients. Each curve represents all the clients, with data points being the mean response time value and the error bars extending to the minimum and maximum response times. For the 1 client case shown in Figure 8(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 server 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 a type of 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. A 64KB block size was empirically found to be a good overall choice for the data set and workload used.

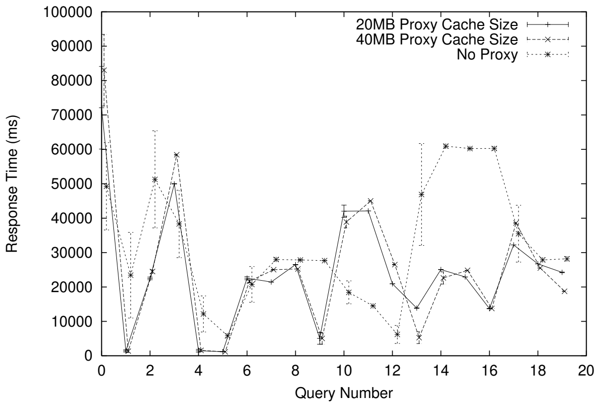
The 2 client case in Figure 8(b) is similar to the 1 client case in that commonality is limited. The two clients are performing different sweep patterns, and any commonality resulting from the common set of interesting points is scattered over time, thus cache misses are common. In considering 4 clients in Figure 8(c), we introduce explicit 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 8(d) as more commonality is added with the introduction of more clients while keeping the same two classes of sweep patterns. Another important observation applies to all the cases, where adding more clients and commonality reduces the response time variation as seen by the error bars in the plots. Not only can the use of a proxy reduce the response time, but it makes the response time more deterministic by avoiding the high variability of WAN communication.

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.

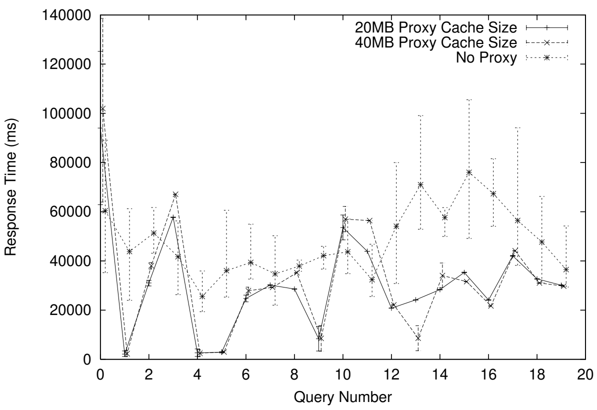
We now consider the effect a proxy has on server utilization. As in the simulated experiments, we expect the reduction of requests that are sent to the server to reduce overall server utilization. Table 3 shows server



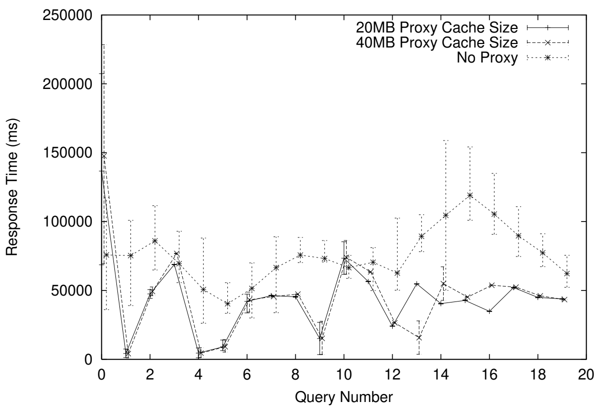
(a) 1 client



(b) 2 clients



(c) 4 clients



(d) 8 clients

Figure 8: Response Time as number of clients is increased, 64KB Proxy Block Size. The error bars delimit min and max times over all clients.

No. Clients	No Proxy	Proxy (20MB Cache)	Proxy (40MB Cache)
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 3: Server Utilization

utilization for the same set of experiments. For the single backend process configuration we used, only 2 clients are enough to saturate the server. Addition of the proxy reduces the utilization overall by 2% to 12%. Another way of thinking about the effects of a proxy is that the *request density* is reduced. Request density is the number of requests per unit of time. By satisfying certain blocks in the proxy cache, the stream of block requests is spaced out, thus reducing the request density. Given this reduction, the server can better keep pace with the request stream it is presented, as demonstrated by the utilization numbers, so is able to handle more clients concurrently.

Finally we look at the reduction in wide-area communication volume for these experiments. The general trend was that the WAN 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 the 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. We can still conclude that for cases when commonality is present, the proxy cache can result in a large reduction in wide area network traffic. The 1 client case again illustrates the benefit of blocking and the prefetching effect, in the absence of inter-client commonality.

6 Discussion

We can make several key observations from simulating and building a proxy for the T2 system. Better understanding of these issues is important for improving the performance of T2-like applications.

Server Phasing

The proxy transforms client queries into a series of block queries. One major problem that can arise is phasing at the server. The T2 frontend process reads a set of requests up to some fixed maximum number, and then broadcasts the query batch to the backend processes. Each backend processes the query batch as a single unit, scheduling disk I/O, performing computation as needed, and sending the result data to the clients. When completed, the backend processes are then ready to handle the next query batch.

This model of operation is well suited for individual clients sending large queries directly to the server, but can cause phasing problems when used by the proxy sending many small queries. Consider the case where the set of block queries sent by the proxy are not all included in the same query batch sent to the backends. In that case, the data for the original single client query will not arrive at the proxy until at least two backend query batches are completed. This defeats the whole point of using a parallel server and declustering to achieve fast disk I/O for each query.

This problem could be solved in various ways. The T2 protocol packet format could be augmented to include a flag that asks the frontend to continue reading client queries until the one marked *last*. Then all the proxy block queries would be executed in a single backend processing phase. Alternatively, we could add a new T2 query packet that can hold more than one multi-dimensional range request, and the proxy would only send one multi-query to the server.

Uniform Cache Block Size

In the design of the proxy prototype for the Virtual Microscope, we wanted all magnification levels for a particular block of data to land in the same hash bucket. This would have allowed for automatic down-sampling of block data when a request arrives for a lower resolution image for which higher resolution blocks exist in the cache. Slide data opaqueness is violated, but we wanted to keep that possibility open if there were sufficient need. Considering a request for a given reply size, the size of the blocks and therefore the number of blocks would vary based on the magnification level in the query.

The problem is this design decision causes a significant loss in performance. 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. Thus we have a block size of only $1KB$ 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, it almost did not matter if anything was cached, because response time was dominated by the nearly universal need to access the server for some of the blocks.

This design problem 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 become quite significant. Suffice to say, the entire system ran very slowly with variable block sizes.

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 T2 uses a request-response style dissemination mechanism [13], forcing queries and responses to pass through an intermediary causes a guaranteed latency penalty. Simply moving up the protocol stack in the proxy and back down again will add overhead. In early tests of the system with the proxy, a simple pass-thru intermediary caused a doubling of the client latency. Any proxy for a request-response based application will see a similar effect. Perhaps better would be a publish-subscribe model of dissemination where clients instead ask for data ahead of time, and the server simply pushes data to the client. Of course, this is only useful if there is a way to know what future data interests will be. For many T2 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 client steers the data flow, rather than asking for each piece. Wide-area applications may have to mandate such protocols if interactivity is needed.

Beyond publish-subscribe dissemination, we could also try 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 both the original client that requested it and the proxy. In addition to reducing the response time for the client query, this model would also reduce server utilization. The server only has to spend time sending the data onto the network once, and the network is responsible for forking the data somewhere en route. It would be difficult to have the reply data sent to all recipients of a block and the proxy, because the main expense in using IP Multicast is in building and maintaining the multicast tree as users join and leave a multicast group. The multicast model may work well for long running jobs that infrequently have clients joining and leaving the multicast group, but would be prohibitively expensive for the fine granularity changes that would be needed to handle common block requests in the proxy. The proxy architecture would potentially require multicast group membership changes for every query and block, for which current IP Multicast is ill suited.

6.1 Related Work

Much work is being done to find ways to make wide-area data access more efficient. To reduce latency in processing queries on relational databases, Query Scrambling [1, 3, 4] is a technique that changes the query plan at runtime to do useful work when delays are seen from some data sources. Our proxy technique does not require more than one data source. 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 [14] 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 closely resembles where our work is heading, with the exception that we utilize intermediate nodes in the path from the client to the server. The use of intermediate nodes is a primary distinction between our work and most related database research. Tradeoffs in using client and server resources has been considered for some time now [15].

Due to the explosion of World Wide Web use, there is a large body of work in trying to effectively deal with reducing impact on the wide-area network. Caching web proxies have been studied to great length [17]. A difficult problem here is trying to find a model that does a good job of representing typical web traffic [19]. Our work is different in that we are primarily dealing with multi-dimensional data, and can leverage the semantics of the data and usage to have a better idea of what data will be requested in the future. In addition web data is inherently variable sized [6], whereas we are able to use a uniform sized cache unit.

7 Conclusion

We have shown how the addition of an intermediary proxy in between the clients and server of T2 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 increased 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. All these benefits were achieved without changing the existing client or server T2 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 taken advantage of. 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.

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.

References

[1] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Scrambling query plans to cope with unexpected delays. In *Proceedings of the 1997 ACM-SIGMOD Conference*, Tucson, AZ, May 1997.

[2] Asmara Afework, Michael D. Beynon, Fabian Bustamante, Angelo Demarzo, Renato Ferreira, Robert Miller, Mark Silberman, Joel Saltz, Alan Sussman, and Hubert 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, March 1998. To appear in Proceedings of the 1998 AMIA Fall Symposium.

[3] Laurent Amsaleg, Philippe Bonnet, Michael Franklin, Anthony Tomasic, and Tolga Urhan. Improving responsiveness for wide-area data access. *IEEE Data Engineering Bulletin*, 20(3), September 1997. http://www.cs.umd.edu/users/franklin/debull/amsaleg.ps.

[4] Laurent Amsaleg, Michael J. Franklin, Anthony Tomasic, and Tolga Urhan. Scrambling query plans to cope with unexpected delays. In *Proccedings of the 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, December 1996.

[5] Michael D. Beynon. Performance evaluation of client-server architectures for large-scale image-processing applications. Virtual Microscope (to be assigned a University of Maryland TR#), University of Maryland, College Park, May 1998.

[6] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 193–206, Berkeley, CA, 1997.

[7] Carl F. Cerco and Thomas Cole. User's guide to the CE-QUAL-ICM three-dimensional eutrophication model, release version 1.0. Technical Report EL-95-15, US Army Corps of Engineers Water Experiment Station, Vicksburg, MS, 1995.

[8] C. Chang, A. Acharya, A. Sussman, and J. Saltz. T2: A customizable parallel database for multi-dimensional data. *ACM SIGMOD Record*, 27(1):58–66, March 1998.

[9] Chialin Chang, Bongki Moon, Anurag Acharya, Carter Shock, Alan Sussman, and Joel 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, April 1997.

[10] Srinivas Chippada, Clint N. Dawson, Monica L. Martínez, and Mary F. Wheeler. A Godunov-type finite volume method for the system of shallow water equations. *Computer Methods in Applied Mechanics and Engineering*, 1997. Also a TICAM Report 96-57, University of Texas, Austin.

[11] Shaul Dar, Michael J. Franklin, Bjorn T. Jonsson, Divesh Srivastava, and Michael 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.

[12] 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., October 1997.

[13] Michael Franklin and Stan Zdonik. Dissemination-based information systems. *IEEE Data Engineering Bulletin*, 19(3):20–30, September 1996.

[14] Michael Franklin and Stan 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.

[15] Michael J. Franklin, Bjorn Thor Jonsson, and Donald Kossman. Performance tradeoffs for client-server query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD96)*. ACM Press, June 1996.

[16] John Jamison, Randy Nicklas, Greg Miller, Kevin Thompson, Rick Wilder, Laura Cunningham, and Chuck Song. vBNS: not your father's Internet. *IEEE Spectrum*, 35(7):38–46, July 1998.

[17] Thomas M. Kroeger and Darrell D. E. Long. Exploring the bounds of Web latency reduction from caching and prefetching. In *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 13–22, Berkeley, CA, 1997.

[18] Microsoft Corp. Microsoft TerraServer. http://www.terraserver.microsoft.com, 1998.

[19] Jeffrey C. Mogul. Network behavior of a busy web server and its clients. Technical Report 95/5, Digital Western Research Laboratory, Palo Alto, CA, October 1995.