Systems Support for Remote Visualization of Genomics Applications over Wide Area Networks

Lars Ailo Bongo[1], Grant Wallace[2], Tore Larsen[1],
Kai Li[2], Olga Troyanskaya[2,3]

[1] Department of Computer Science, University of Tromsø, N-9037 Tromsø, Norway.

[2] Department of Computer Science, Princeton University, Princeton NJ 08544, USA.

3 Lewis-Sigler Institute for Integrative Genomics, Princeton University,

Princeton NJ 08544, USA.

{larsab, tore}@cs.uit.no

{gwallace, li, ogt}@cs.princeton.edu

Abstract. Microarray experiments can provide molecular-level insight into a variety of biological processes, from yeast cell cycle to tumorogenesis. However, analysis of both genomic and protein microarray data requires interactive collaborative investigation by biology and bioinformatics researchers. To assist collaborative analysis, remote collaboration tools for integrative analysis and visualization of microarray data are necessary. Such tools should: (i) provide fast response times when used with visualization-intensive genomics applications over a low-bandwidth wide area network, (ii) eliminate transfer of large and often sensitive datasets, (iii) work with any analysis software, and (iv) be platform-independent. Existing visualization systems do not satisfy all requirements. We have developed a remote visualization system called Varg that extends the platform-independent remote desktop system VNC with a novel global compression method. Our evaluations show that the Varg system can support interactive visualization-intensive genomic applications in a remote environment by reducing bandwidth requirements from 30:1 to 289:1.

Keywords: Remote visualization, genomics collaboration, Rabin fingerprints, compression.

1. Introduction

Interactive analysis by biology and bioinformatics researchers is critical in extracting biological information from both genomic [1], [2] and proteomic [3], [4], [5], [6], [7] microarrays. Many supervised and unsupervised microarray analysis techniques have been developed [8], [9], [10], [11], and the majority of these techniques share a common need for visual, interactive evaluation of results to examine important

patterns, explore interesting genes, or consider key predictions and their biological context.

Such data analysis in genomics is a collaborative process. Most genomics studies include multiple researchers, often from different institutions, regions, and countries. Of the 20 most relevant papers returned by BioMed Central with the query "microarray," 14 had authors located at more than one institution, and 7 had authors located on either different continents or cross continents. Such collaboration requires interactive discussion of the data and its analysis, which is difficult to do without sharing a visualization of the results. To make such discussions truly effective, one in fact needs not just static images of expression patterns, but an opportunity to explore the data interactively with collaborators in a seamless manner, independent of the choice of data analysis software, platforms, and of researchers' geographical locations.

We believe that an ideal collaborative, remote visualization system for genomic research should satisfy three requirements. First, synchronized remote visualization should have a fast response time to allow collaborating parties to interact smoothly, even when using visualization-intensive software across a relatively low-bandwidth wide area network (WAN). Second, collaborating parties should not be required to replicate data since microarray datasets can be large, sensitive, proprietary, and potentially protected by patient privacy laws. Third, the system should allow collaborators to use any visualization and data analysis software running on any platform.

Existing visualization systems do not satisfy all three requirements above. Applications with remote visualization capabilities may satisfy the first and the second requirements, but typically not the third as require universal adoption among participating collaborators. Thin-client remote visualization systems, such as VNC [12], Sun Ray [13], THINC [14], Microsoft Remote Desktop [15] and Apple Remote Desktop [16] satisfy only the second requirement because they do not perform intelligent data compression and all except VNC are platform-dependent. Web browser-based remote visualization software can satisfy the third requirement, but not the first two because these systems are not interactive and do not optimize the network bandwidth requirement.

This paper describes the design and implementation of a remote visualization system called Varg that satisfies all three requirements proposed above. To satisfy the first requirement, the Varg system implements a novel method to compress redundant two-dimensional pixel segments over a long visualization session. To satisfy the second and the third requirements, the Varg system is based on a platform-independent remote desktop system VNC, whose implementation allows remote visualization of multiple applications in a network environment.

The main contribution of the Varg system is the novel method for compressing 2-D pixel segments for remote genomic data visualization. Genomic data visualization has two important properties that create opportunities for compression. The first is that datasets tends to be very large. A microarray dataset typically consists of a matrix of expression values for thousands or tens of thousands of genes (rows). The

second is that due to the limitation of display scale and resolution, researchers typically view only tens of genes at a time by frequently scrolling visualization frames up and down. As a result, the same set of pixels will be moved across the display screen many times during a data analysis and visualization session. We propose a novel method to identify, compress and cache 2-D pixels segments. Not sending redundant segments across the WAN greatly improves the effective compression ratio reducing network bandwidth requirements for remote visualization.

Our initial evaluation shows that the prototype Varg system can compress display information of multiple genomic visualization applications effectively, typically reducing the network bandwidth requirement by two orders of magnitude. We also demonstrate that this novel method is highly efficient and introduces a minimal overhead to the networking protocol; and that the Varg system can indeed support multiple visualization-intensive genomic applications in a remote environment interactively with minimal network bandwidth requirement.

2. System Overview

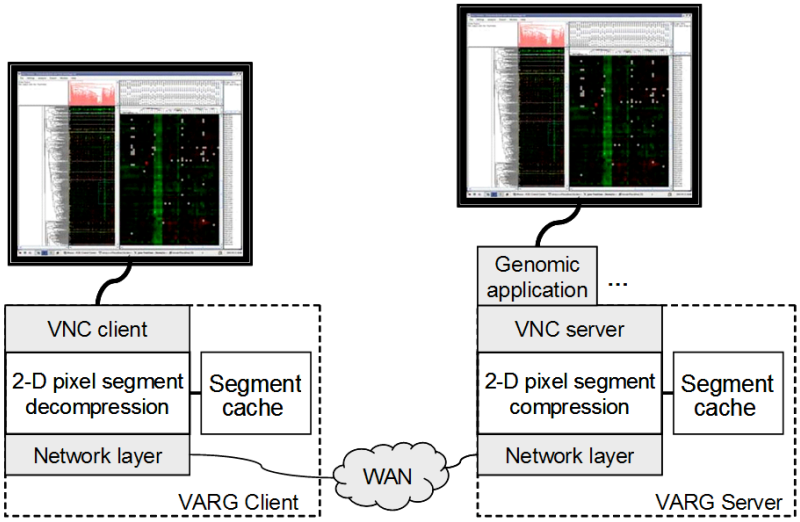

Figure 1: Remote visualization overview. The VNC server sends screen updates to the VNC client. The Varg system caches updates and provides compression by replacing updates already in the client cache with the cache index.

Varg is a network bandwidth optimized, platform-independent system that allows users to interactively visualize multiple remote genomic applications across a WAN. The architecture of Varg is based on a client-server model as shown in Figure 1. Varg leverages the basic VNC protocol (called RFB) to implement platform-independent remote visualization and extends it with a high-speed 2-D pixel segment compression module with a cache in the server and a decompression module with a cache in the

client. The Varg server runs multiple visualization applications, compresses their two-dimensional pixel segments, and communicates with the remote Varg client. The client decompresses the data utilizing a large cache and performs remote visualization.

The caches of the Varg server and client cooperate to minimize the required network bandwidth by avoiding redundant data transfers over the network. Unlike other global compression methods for byte data streams [17, 18], Varg is designed to optimize network bandwidth for remote data transfers of 2-D pixels segments generated by genomic visualization applications on the VNC server.

Since Varg is built on the VNC protocol, it allows multiple users to conveniently visualize and control a number of applications in a desktop across a network. When an owner of some sensitive or very large data set wants to collaborate with a remote collaborator, she can run one or more analysis programs that access her sensitive data on her Varg server, which connects with a Varg client on her collaborator's site. The researchers can then use these programs in a synchronized fashion across the network. Although the collaborator can visualize and control the application programs in the same way as the owner, the Varg client receives only visualization pixels from the Varg server; no sensitive data is ever transferred across the network. We expect that this feature may be especially useful to researchers working with clinical data due to privacy and confidentiality concerns.

3. Compressing 2-D Pixel Segments

The main idea in the Varg system is to compress visualization pixel data at a fine-grained 2-D pixel segment level. The system compresses 2-D pixel segments by using a global compression algorithm to avoid sending previously transferred segments and by applying a slow, but efficient, local compression [19] on the unique segments. This section describes Varg's basic compression algorithm, explains our novel content-based anchoring algorithm for 2-D pixel segments, and outlines an optimization of the compression algorithm using a two-level fingerprinting scheme that we developed.

3.1. Basic Compression Algorithm

The basic compression algorithm uses fingerprints together with cooperative caches on the Varg server and client to identify previously transferred pixel segments, as shown in Figure 2.

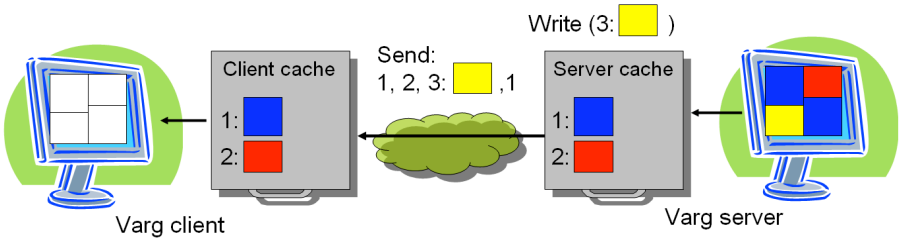

Figure 2: Compression scheme. The screen is divided into regions which are cached at both ends of the low-bandwidth network. Fingerprints are sent in place of previously sent regions.

The algorithm on the Varg server is:

- Process an updated region of pixels from the VNC server
- Segment the region into 2-D pixel segments
- For each segment, compute its fingerprint and use the fingerprint as the segment's identifier to lookup in the server cache. If the segment has not been sent to the Varg client previously, compress the segment with a local compression method and send the segment to the client. Otherwise, send the fingerprint instead.

The algorithm on the Varg client is:

- If the received data is a 2-D pixel segment, decompress it with a corresponding algorithm, write the fingerprint and segment to the cache, and then pass the segment to the VNC client
- If the received data is a fingerprint, retrieve the segment of the fingerprint from its cache and then pass the segment to the VNC client.

The basic algorithm is straightforward and its high-level idea is similar to previous studies on using fingerprints (or secure hashes) as identifiers to avoid transfer of redundant data segments [20, 21], [22], [17], [18]. The key difference is that previous studies are limited to deal with one-dimensional byte streams and have not addressed the issue of how to anchor 2-D pixel segments. In a later section, we will also present an algorithm to use short fingerprints to compress repeated 2-D pixel segments.

3.2. Content-Based Anchoring of 2-D Pixel Segments

One basis of our approach is content-based anchoring where the 2-D region of display-pixels is divided into segments based on segment content. A simpler approach would be to anchor segments statically (such as an 8×8 pixel grid, used in MPEG compression algorithms). The problem with a static approach is that the anchoring is sensitive to screen scrolls. When a user scrolls her visualization by one pixel, the segmentation of 2-D pixels will be shifted by one pixel relative to the displayed image. Even if the entire scrolled screen has been transferred previously,

the content of segments will typically have changed, giving a new fingerprint and requiring a new transfer across the WAN.

Our approach is to perform content-based anchoring instead of static anchoring. The anchoring algorithm takes its input from the frame-buffer, and returns a set of rectangular segments which subdivide the screen. The goal of the algorithm is to consistently anchor the same groups of pixels no matter where they are located on the screen. The main difficulty in designing a content-based anchoring algorithm for a screen of pixels is that the data is two dimensional.

Manber introduced a content-based technique to anchor one-dimensional data segments for finding similar files [22]. His method applies a Rabin fingerprint filter [23] over a byte data stream and identifies anchor points wherever the k least significant bits of the filter output are zeros. With a uniform distribution, an anchor point should be selected every 2^k bytes.

Our algorithm combines the statically divided screen approach with Manber's technique. The algorithm is based on the observation that content motion in microarray analysis is often due to vertical or horizontal scrolling. However, it is not practical to do redundancy detection both horizontally and vertically due to the computational cost and reduced compression ratio caused by overlapping regions. Therefore, we estimate whether the screen has moved mostly horizontally or mostly vertically using Manber's technique. We generate representative fingerprints for every 32nd row, and every 32nd column for the screenshot, and compare how many fingerprints are similar to the row and column fingerprints of the previous screenshot. Assuming that horizontal scrolling or moving will change most row fingerprints, but only a few column fingerprints, we can compare the percentage of similar row and column fingerprints to estimate which movement is dominant.

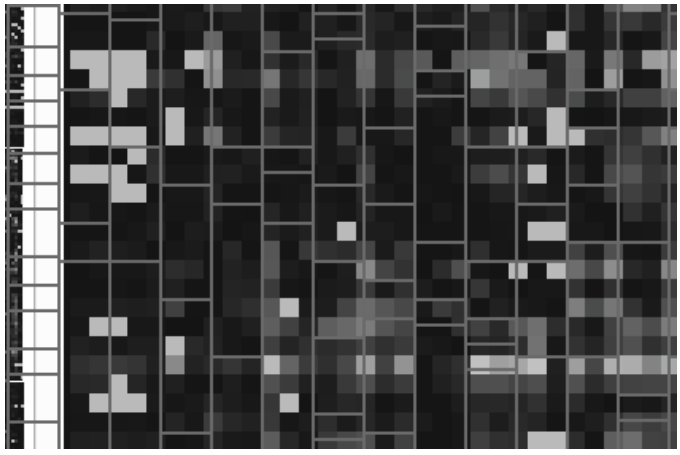

Figure 3: A portion of the screen that is divided into segments that move with the content.

For predominately vertical motion we statically divide the screen into *m* columns (*m* times screen height) and divide each column into regions by selecting anchoring rows. The anchoring rows are selected based on their fingerprint calculated using a

four byte at a time Rabin fingerprint implementation. The column segmentation is ideal for scrolling because the regions move vertically with the content. If we detect predominately horizontal motion instead, we run the same algorithm but divide the screen into rows first and then divide each row into regions by selecting anchoring columns.

Screen data can include pathological cases when large regions of the screen have the same color. For such regions, the fingerprints will be identical. Thus, either all or no fingerprints will be selected. To avoid such cases, our algorithm does fingerprint selection in three steps. First all fingerprints are calculated. Second, we scan the fingerprints and mark fingerprints as *similar* if at least *s* subsequent fingerprints are identical. Third, we select fingerprints using the *k* most significant bits, while imposing a minimum distance *m* between selected fingerprints. Also, the first and last rows are always selected.

Empirically we have found that the best results are achieved for s = 8, m = 16 or 32, and k such that each 64th row on the average is selected. Also, such similar regions compress well using a local compression algorithm such as zlib [24] due to their repeated content. We have found empirically that imposing a maximum distance does not improve the compression ratio or compression time.

3.3. An Optimization with Two-Level Fingerprinting

An important design issue in using fingerprints as identifiers to detect previously transferred data segments is the size of a fingerprint. Previous systems typically chose a secure hash, such as 160-bit SHA-1 [25], as a fingerprint so that the probability of a fingerprint collision can be lower than a hardware bit error rate. However, since the global compression ratio is limited to the ratio of the average pixel segment size to the fingerprint size, increasing the fingerprint size reduces this limit on the compression ratio.

To maximize the global compression ratio and maintain a low probability of fingerprint collision, we use a two-level fingerprinting strategy. The low-level fingerprinting uses 32-bit Rabin fingerprint of fingerprints, one for each 2-D pixel segments. Although using such short fingerprints will have a higher probability of a fingerprint collision, they can be computed quickly using the fingerprints already computed for the anchoring, thereby maintaining a high global compression ratio.

The high-level fingerprinting uses SHA-1 hashes as fingerprints. It computes a 160-bit fingerprint for each of the transferred pixel segments. The server computes such a long fingerprint as a strong checksum to detect low-level fingerprint collisions. When a low-level fingerprint collision is detected, the server resends the pixel segment covered by the long fingerprint.

Another way to look at this method is that the server may send two sets of updates, the first based on short fingerprints that can have collisions, and the second set of updates consisting of corrections in case of short fingerprint collisions. This method reduces the user perceived end-to-end latency.

4. Implementation

We have implemented a prototype system (called Varg) consisting of a sequential server and a client, as described in Section 2. The Varg server implements the 2-D pixel segment compression algorithm and Varg client implements the corresponding decompression algorithm described in the previous section.

The integration of Varg compression, decompression, and cache modules with the VNC client and server are simple. VNC has only one graphics primitive: "Put rectangle of pixels at position (x, y)" [12]. This allows separating the processing of the application display commands from the generation of display updates to be sent to the client. Consequently the server only needs to detect updates in the frame-buffer, and can keep the client completely stateless.

Varg employs a synchronized client and server cache architecture that implements an eventual consistency model using the two-level fingerprinting mechanism. The client and server caches are initialized at Varg system start time. The client cache is then synchronized by the updates sent from the server. The compression algorithm requires the client cache to maintain the invariant that whenever the client receives a fingerprint, its cache must have the fingerprint's segment. Since short fingerprints may have collisions, our prototype allows the client cache to contain any segment of the same short fingerprint at a given time. The long fingerprint will eventually trigger an update to replace it with a recently visualized segment.

5. Evaluation

We have conducted an initial evaluation of the Varg prototype system. The goal of the evaluation is to answer the following two questions:

- What are the network communication requirements for remote visualization of genomic applications?
- How much compression of network communication data can the Varg prototype system achieve for remote visualization of genomic applications?

To answer the first question, we have measured the difference between available bandwidth on current WANs and the required bandwidth for remote visualization of Genomic applications. To answer the second question, we used a trace-driven VNC emulator to find how much the Varg system can reduce the communication time for three genomic applications. In the rest of this section, we will present our experimental testbed and then our evaluation results to answer each question.

5.1. Experimental Testbed

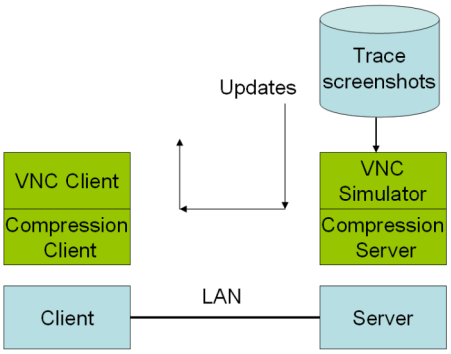

Figure 4: Experimental testbed for the bandwidth requirements and compression ratio evaluation.

In order to compare compression ratios of various compression algorithms, our experimental testbed (Figure 4) employs two identical Dell Dimension 9150, each with one dual-core 2.8 GHz Pentium D processor and 2 GB of main memory. Both computers run Fedora Core 4, with Linux kernel 2.6.17SMP.

The server runs with a screen resolution of 1280x1024 pixels and with a color depth of 32 bits per pixel. We also run an experiment on a display wall with a resolution of 3328x1536 pixels.

To compare different systems, an important requirement is to drive each system with the same remote visualization workloads. To accomplish this goal, we have used a trace-driven approach. To collect realistic traces, we used the Java AWTEventListener interface to instrument three genomic microarray analysis applications. We used these to record a 10-minute trace containing all user input events for each case. Later the traces were used to create a set of screenshots, each taken after playing back a recorded mouse or keyboard event that changes the screen content. The screenshots are used by a VNC simulator that copies a screenshot to a shadow framebuffer, and invokes the Varg server, which does change detection and compression before sending the updates to the client.

5.2. Network Communication Requirements

In order to answer the question about the network communication requirements for remote visualization of genomic applications, we need to answer several related questions including the composition of communication overhead, the characteristics of available networks, the behavior of remote visualization of genomic applications, and the required compression ratio to meet certain interactive requirements. Our finding is that genomic applications require high compression ratio to compress the pixel data to use existing WAN connections.

The network communication overhead can be expressed with a simple formula:

$$2L + \frac{S}{B \cdot R} + C \quad (1)$$

where L is the network latency, S is the data to be transferred, B is the network bandwidth, R is compression ratio, and C is compression time. The formula considers compression a part of the network communication mechanism, thus the total communication overhead includes the round-trip network latency plus the time to compress and transfer the data. This formula ignores the overheads of several software components such as the VNC client and server. Also, we usually ignore decompression time since it is low compared to the compression time (less than 1msec).

Based on this formula, it is easy to see that different network environments have different implications for remote visualization. Conventional wisdom assumes that WANs have low bandwidth. To validate this assumption we used Iperf [26] to measure the TCP/IP throughput between a server and a client connected using various local and wide area networks. The following table shows that the WAN throughput ranges from 0.2 to 2.13 Mbytes/sec (Table 1). This is up to 400 times lower than for Gigabit Ethernet. Also, the two-way latency is high, ranging from 11—120 ms.

Table 1: TCP/IP bandwidth and latency for client-server applications run on local area and wide area networks.

Network	Bandwidth (Mbytes/sec)	Latency (msec)
Gigabit Ethernet	80.00	0.2
100 Mbps Ethernet	8.00	0.2
Princeton – Boston	2.13	11
Princeton – San Diego	0.38	72
Princeton (USA(– Tromsø (Norway)	0.20	120

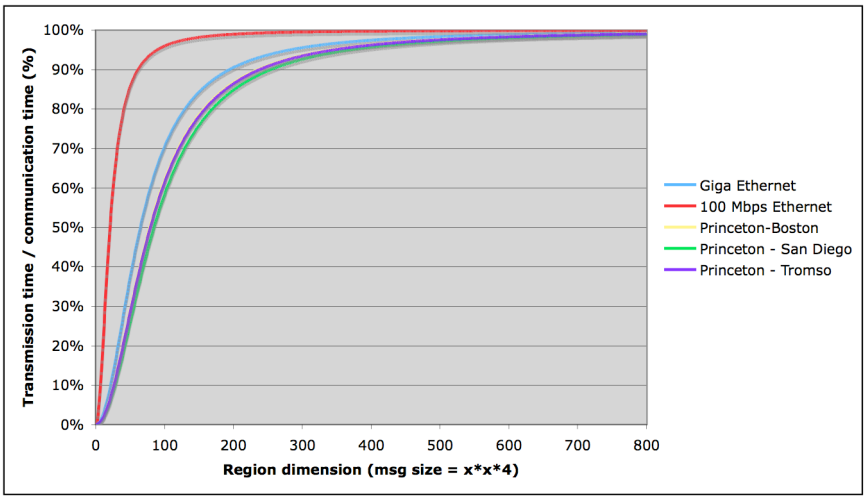

Figure 5: For regions larger than 80×80 pixels, the transmission time dominates the total communication overhead.

Based on the characteristics of the available networks, an interesting question is what size of network transfers contribute significantly to the total communication overhead. Figure 5 shows how much transmission contributes to the communication time depending on the amount of pixel data sent over the network connection. For all WAN networks, the ratio of transmission time to communication time is more than 50% for regions more than about 80×80 pixels or 25 Kbytes.

Two natural questions are, how frequent are screen updates larger than 80×80 pixels for genomics applications, and are the update sizes different compared to Office applications usually used in remote collaboration. To answer these questions, we measured the average VNC update size for three sessions using three applications on Windows XP:

1. Writing this paper in Microsoft Word.
2. Preparing the figures for this paper in Microsoft PowerPoint.
3. Microarray analysis using the popular Java Treeview software [27].

For each application, we recorded a session lasting about 10 minutes. We instrumented the VNC client to record the time and size of all screen updates received. We correlated these to when the update requests were sent, to get an estimate for the size of each screen update.

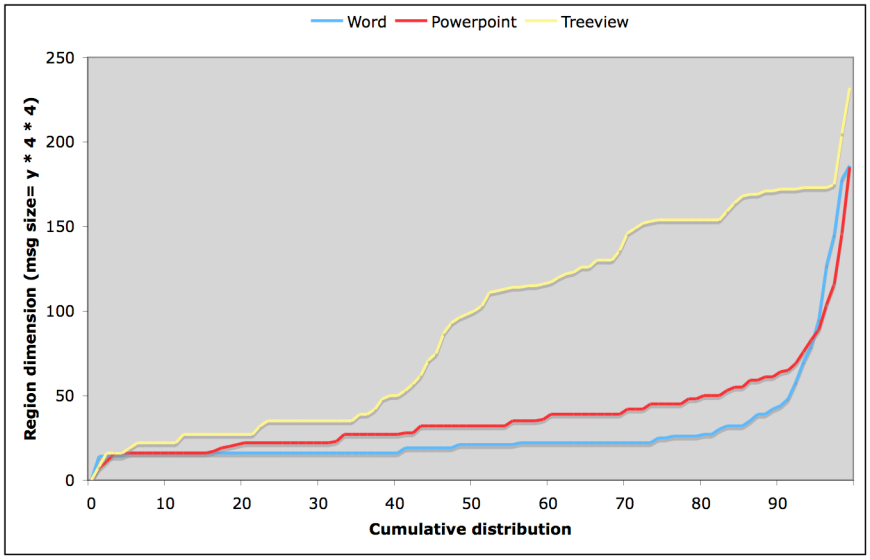

Figure 6: The screen regions update sizes for the Java Treeview application are much larger than for the Office applications. About 50% of the messages are more than 80x80 pixels.

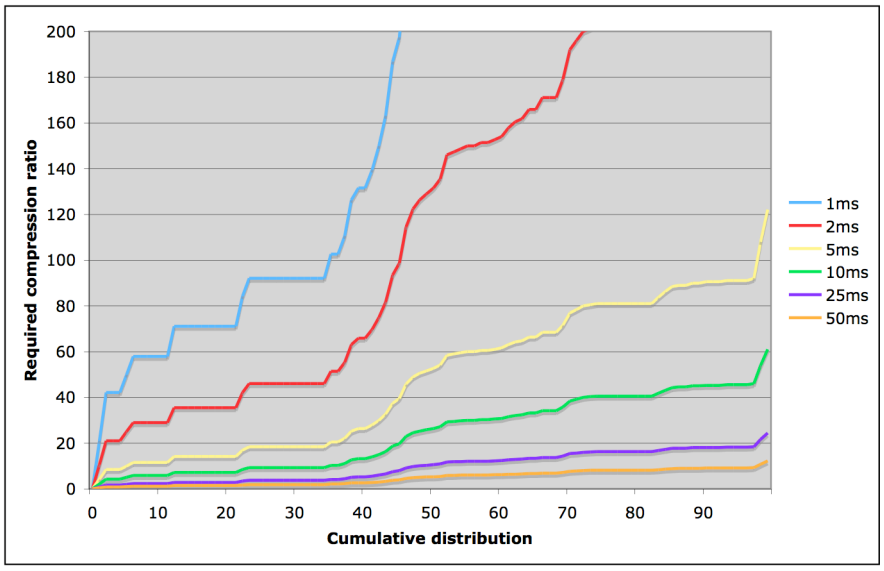

Figure 7: Compression ratio required to keep transmission overhead below a given threshold for the Princeton-San Diego network connection. The x-axis shows the percentiles for the Treeview message sizes in Figure 6**.** Compression time is not taken into account.

The results show that updated regions are much larger for the genomic application than for the two office applications (Figure 6). About 50% of the messages are larger

than 80×80 pixels, and hence for these the transmission time will be longer than the network latency for the WANs. Another observation is that the genomic application has a higher update frequency than office applications. Combined these increase the required bandwidth.

To see the impact of compressing pixel data for remote visualization, we have calculated the compression ratio necessary to maintain the transmission time below a given threshold for a cross-continent WAN (Figure 7). We have several observations from the results. First, it requires a compression ratio of about 25:1 to keep the transmission time below 10 msec for most of the network traffic. Second, the compression ratio required to maintain the same transmission time increases rapidly for the top two percentiles. Third, as the message size increases, the compression ratio required for the different transmission times increases.

The following section examines which compression ratio and compression time gives the best transmission time.

5.3. Compression results

To answer the question about what compression ratios the Varg system can achieve for remote visualization of genomic applications, we have measured compression ratios, compression cost and the reduction of transmission time.

Table 2: Compression ratio for four genomic data analysis applications.

	Differencing	2D pixel segment compression	Ziv-Lempel (zlib)	Total compression
TreeView	1.89	5.74	19.98	216.76
TreeView-Cube	2.87	4.05	24.88	289.19
TMeV	1.52	2.46	7.90	29.54
GeneVaND	3.15	2.72	10.85	92.96

To measure the compression ratios the Varg system can achieve, we have used four 15-minute traces recorded using: Java Treeview [27], Java Treeview on the display wall [28], TMeV [29], and GeneVaND [30]. For Treeview, the visualizations mostly are scrolling and selecting regions from a single bitmap. GeneVaND has relatively small visualization windows and the trace includes 3D visualizations as well as some 2D visualizations. TMeV trace includes different short visualizations.

The total compression ratios by our method are 217, 289, 30 and 93 for the four traces respectively (Table 2). These high compression ratios are due to a combination of three compression methods: Region differencing, 2D pixel segment compression, and zlib local compression. We have several observations based on these data. First, the combined compression results are excellent. Second, zlib contributes the most in all cases, but zlib alone is not enough to achieve high compression ratios. Third, the 2D pixel segment compression using fingerprinting contributes fairly significantly to

the compression ratio ranging from 2.5 to 5.7. This is due to the fact that the differencing phase has already removed a large amount of redundant segments.

Table 3: Average compression time per screen update. The total compression time depends on the application window size, and how well the differencing and 2D pixel segment compression modules compress the data before zlib is run.

	Differencing	2D pixel segment compression	Ziv-Lempel (zlib)	SHA-1
TreeView	0.9 ms	3.8 ms	11.1 ms	3.5 ms
TreeView-Cube	2 ms	7.9 ms	30.2 ms	7 ms
TMeV	1.3 ms	6.6 ms	83.4 ms	7.7 ms
GeneVaND	1 ms	2.7 ms	10.1 ms	1.5 ms

To understand the contribution of different compression phases to the compression time, we measured the time spent in each module (Table 2). The most significant contributor is zlib, which consumes more than 10 ms in all cases. In TMeV it consumes more than 83ms, since more data is sent through this module due to the low compression ratios for the differencing and 2D pixel segment compression modules. The second most significant contributor is anchoring, but it is below 8 ms even for the display wall case. Although SHA-1 calculation contributes up to 8ms in the worst case, its computation overlaps with network communication.

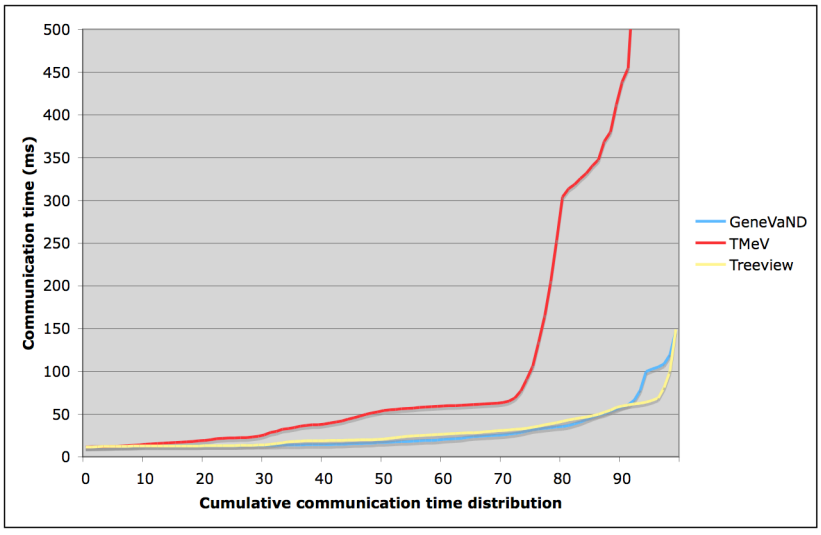

Figure 8: Communication time distribution for update messages over the Princeton—Boston network. For Treeview and GeneVaND more than 90% of the communication overheads are less than 100ms. The update size distribution differs from Figure 5, since a more accurate tracing tool was used to capture the trace.

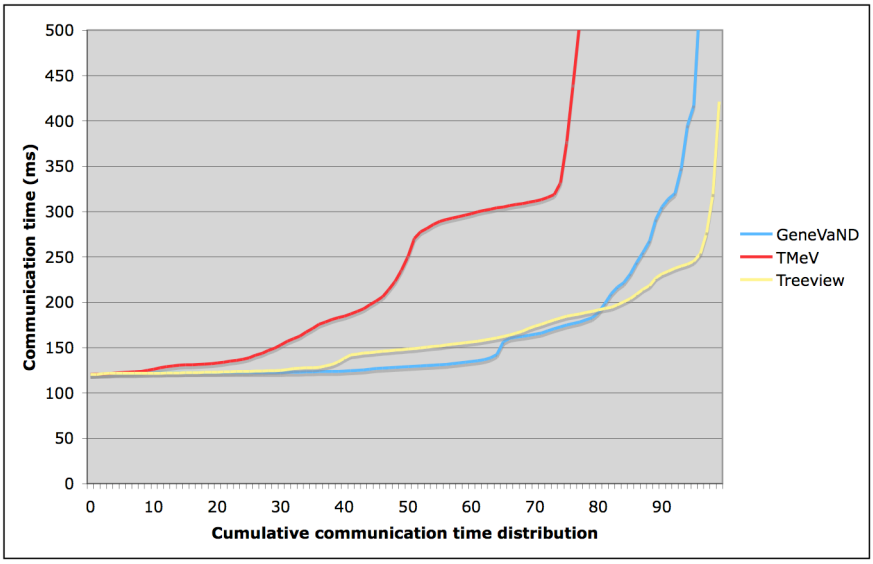

Figure 9: Communication time distribution for the Princeton—Tromsø network. For Treeview and GeneVaND more than 80% of the communication overheads are less than 200ms.

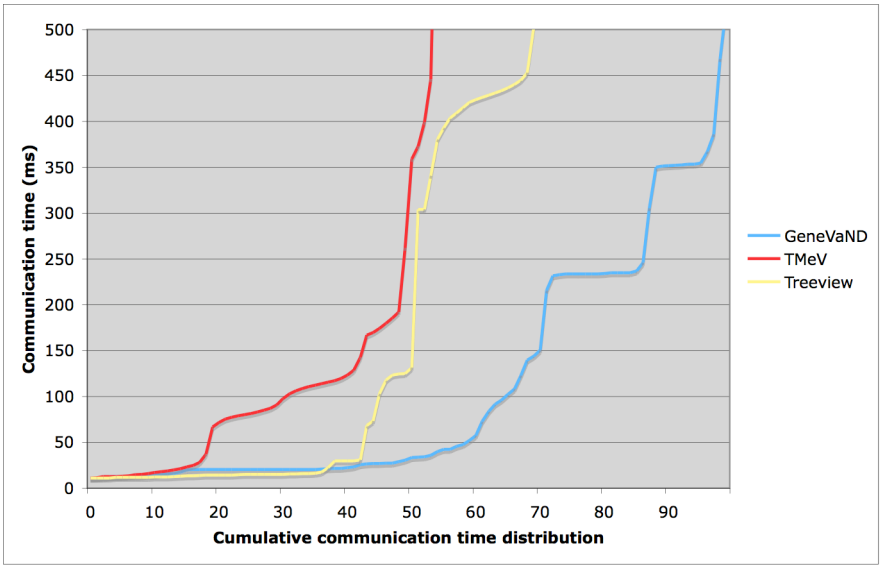

Figure 10: Communication time distribution with VNC compression for the Princeton—Boston network. Compared to Varg the communication time shown in Figure 9 significantly increases for large messages.

To understand the reduction in communication time, we recorded for each update the number of compressed bytes returned by each module, and the compression time for each module. This allows us to use Formula 1 to estimate the communication time for each of the WANs in Table 1. The cumulative distribution of communication times for the highest and lowest bandwidth networks are shown in Figure 8 and Figure 9. Without compression the communication overhead for the largest updates is several seconds. For the Princeton—Boston network the communication overhead with Varg is less than 100ms for over 90% of the messages (except for TMeV). On the Princeton—Tromsø network, for 80% of the update operations the communication overhead is less than 200ms, of which the latency contributes to 112 ms.

To compare our compression against the zlib compression used in many VNC implementations for low-bandwidth networks, we disabled the 2D pixel segment compression module in Varg, and did a similar calculation as above (Figure 10). The results show a significant increase in communication time, especially for Treeview where the communication overhead is more than 300ms for about 50% of the messages.

6. Related Work

Compression algorithms used by VNC [12] implementations either take advantage of neighboring region color similarities, use general purpose image compression [31] such as JPEG [32], or general purpose compression such as zlib [19]. Neighboring region redundancy compression is fast but has low compression ratio. Therefore zlib is usually used for WANs. Our results show that the compression time for zlib is high. JPEG is lossy, and is not suited for Microarray analysis, since it may introduce visual artifacts that may influence the biologist's interpretation of the data.

Remote visualization systems that use high level graphics primitives for communication, such as Microsoft Remote Desktop [15], are able to cache bitmaps used for buttons and other GUI components. However, the high-level graphics primitives do not compress well leading to performance problems in WANs [13, 33].

Encoders used for streaming video, such as MPEG [34], compress data by combining redundancy detection and JPEG type compression. Usually a static pixel grid is used, which we have shown gives worse performance than our approach. In addition the MPEG compression is lossy and there are no real time encoders available. TCC-M [35] is a block movement algorithm designed for thin-client visualization that use unique pixels in an image (feature sets) to detect 2D region movement. However, redundancy is only detected between the two latest screen updates thus reducing the compression ratio.

Earlier one-dimensional fingerprinting approaches [17, 18] require the two-dimensional screen to be converted to some one-dimensional representation. This will split up two-dimensional regions on the screen causing the size of the redundant regions to decrease, hence reducing the compression ratio.

Access Grid [36] provides multiple collaborators with multimedia services by multicasting audio, video and remote desktop displays such as VNC. However, Access Grid does not provide compression to reduce the network bandwidth requirement for specific data visualization such as genomic data exploration. Since Varg extends the VNC protocols to compress 2D segments for genomic data visualization, it can effectively work together with Access Grid systems to support multi-party collaborations.

7. Conclusion

This paper presents the design and implementation of the Varg system: a network bandwidth optimized, platform-independent system that allows users to interactively visualize multiple remote genomic applications across a WAN. The paper has proposed a novel method to compress 2-D pixel segments by using fingerprinting and proposed a two-level fingerprinting method to improve global compression, and to reduce compression time.

We also found that genomic applications have much higher network bandwidth requirements than office applications. They require substantial compression of network data to achieve interactive remote data visualization on some examples of existing WAN.

An initial evaluation of our prototype system shows that the proposed 2-D pixel segment compression method works well and imposes only modest overheads. By combining with zlib and differencing compression methods, the prototype system achieved compression ratios ranging from 30:1 to 289:1 for four genomic visualization applications that we have experimented with. Such compression ratios allow the Varg system to run remote visualization of genomic data analysis applications interactively across WANs with relatively low available network bandwidths.

8. Acknowledgments

This work was done while LAB and TL were visiting Princeton University and was supported in parts by Princeton University, The University of Tromsø, The Research Council of Norway (incl. project No. 164825), NSF grants CNS-0406415, EIA-0101247, CNS-0509447, CCR-0205594, and CCR-0237113, and NIH grant R01 GM071966. OGT is an Alfred P. Sloan Research Fellow

9. References

1. Lipshutz RJ, Fodor SPA, Gingeras TR, Lockhart DJ: **High density synthetic oligonucleotide arrays**. *Nature Genetics* 1999, **21**:20-24.

2. Schena M, Shalon D, Davis RW, Brown PO: **Quantitative Monitoring of Gene-Expression Patterns with a Complementary-DNA Microarray**. *Science* 1995, **270**(5235):467-470.
3. Cahill DJ, Nordhoff E: **Protein arrays and their role in proteomics**. *Adv Biochem Eng Biotechnol* 2003, **83**:177-187.
4. Sydor JR, Nock S: **Protein expression profiling arrays: tools for the multiplexed high-throughput analysis of proteins**. *Proteome Sci* 2003, **1**(1):3.
5. Oleinikov AV, Gray MD, Zhao J, Montgomery DD, Ghindilis AL, Dill K: **Self-assembling protein arrays using electronic semiconductor microchips and in vitro translation**. *J Proteome Res* 2003, **2**(3):313-319.
6. Huang RP: **Protein arrays, an excellent tool in biomedical research**. *Front Biosci* 2003, **8**:d559-576.
7. Cutler P: **Protein arrays: the current state-of-the-art**. *Proteomics* 2003, **3**(1):3-18.
8. Kerr MK, Churchill GA: **Bootstrapping cluster analysis: assessing the reliability of conclusions from microarray experiments**. *Proc Natl Acad Sci U S A* 2001, **98**(16):8961-8965.
9. Yeung KY, Haynor DR, Ruzzo WL: **Validating clustering for gene expression data**. *Bioinformatics* 2001, **17**(4):309-318.
10. Mendez MA, Hodar C, Vulpe C, Gonzalez M, Cambiazo V: **Discriminant analysis to evaluate clustering of gene expression data**. *FEBS Lett* 2002, **522**(1-3):24-28.
11. Datta S, Datta S: **Comparisons and validation of statistical clustering techniques for microarray gene expression data**. *Bioinformatics* 2003, **19**(4):459-466.
12. Richardson T, Stafford-Fraser Q, Wood KR, Hopper A: **Virtual network computing**. *Ieee Internet Computing* 1998, **2**(1):33-38.
13. Schmidt BK, Lam MS, Northcutt JD: **The interactive performance of SLIM: a stateless, thin-client architecture**. In: *Proceedings of the seventeenth ACM symposium on Operating systems principles*. Charleston, South Carolina, United States: ACM Press; 1999.
14. Baratto RA, Kim LN, Nieh J: **THINC: a virtual display architecture for thin-client computing**. In: *Proceedings of the twentieth ACM symposium on Operating systems principles*. Brighton, United Kingdom: ACM Press; 2005.
15. Cumberland BC, Carius G, Muir A: **Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference**. In. Edited by Press M. Redmond, WA; 1999.
16. **Apple Remote Desktop** [http://www.apple.com/remotedesktop/]
17. Spring NT, Wetherall D: **A protocol-independent technique for eliminating redundant network traffic**. In: *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Stockholm, Sweden: ACM Press; 2000.
18. Muthitacharoen A, Chen B, Mazières D: **A low-bandwidth network file system**. In: *Proceedings of the eighteenth ACM symposium on Operating systems principles*. Banff, Alberta, Canada: ACM Press; 2001.

19. Ziv J, Lempel A: **A universal algorithm for sequential data compression**. *IEEE Transactions on Information Theory* 1977 **23**(3):337 - 343.
20. Broder A: **Some applications of Rabin's fingerprinting method**. In: *Sequences II: Methods in Communications, Security, and Computer Science: 1993*; 1993.
21. Broder A: **On the Resemblance and Containment of Documents**. In: *Proceedings of the Compression and Complexity of Sequences 1997*. IEEE Computer Society; 1997.
22. Manber U: **Finding similar files in a large file system**. In: *Proceedings of the Winter 1994 USENIX Technical Conference*. San Francisco, CA; 1994.
23. Rabin MO: **Fingerprinting by random polynomials**. In: *Technical Report TR-15-81*. Center for Research in Computing Technology, Harvard University; 1981.
24. **DEFLATE Compressed Data Format Specification version 1.3**. In: *RFC 1951*. The Internet Engineering Task Force; 1996.
25. **Secure Hash Standard**. In: *FIPS PUB 180-1*. National Institute of Standards and Technology; 1995.
26. **Iperf webpage** [http://dast.nlanr.net/Projects/Iperf/]
27. Saldanha AJ: **Java Treeview--extensible visualization of microarray data**. *Bioinformatics* 2004, **20**(17):3246-3248.
28. Wallace G, Anshus OJ, Bi P, Chen HQ, Clark D, Cook P, Finkelstein A, Funkhouser T, Gupta A, Hibbs M *et al*: **Tools and applications for large-scale display walls**. *Ieee Computer Graphics and Applications* 2005, **25**(4):24-33.
29. Saeed AI, Sharov V, White J, Li J, Liang W, Bhagabati N, Braisted J, Klapa M, Currier T, Thiagarajan M *et al*: **TM4: a free, open-source system for microarray data management and analysis**. *Biotechniques* 2003, **34**(2):374-378.
30. Hibbs MA, Dirksen NC, Li K, Troyanskaya OG: **Visualization methods for statistical analysis of microarray clusters**. *BMC Bioinformatics* 2005, **6**:115.
31. Richardson T: **The RFB Protocol version 3.8**. In.: RealVNC Ltd; 2005.
32. Gregory KW: **The JPEG still picture compression standard**. *Commun ACM* 1991, **34**(4):30-44.
33. Lai AM, Nieh J: **On the performance of wide-area thin-client computing**. *ACM Trans Comput Syst* 2006, **24**(2):175-209.
34. Gall DL: **MPEG: a video compression standard for multimedia applications**. *Commun ACM* 1991, **34**(4):46-58.
35. Christiansen BO, Schauser KE: **Fast Motion Detection for Thin Client Compression**. In: *Proceedings of the Data Compression Conference (DCC '02)*. IEEE Computer Society; 2002.
36. **Access Grid** [http://www.accessgrid.org/]