Consuming Network Bandwidth with Visapult

Wes Bethel and John Shalf, Lawrence Berkeley National Laboratory

Summary

In one view, high performance and large data visualization is largely an exercise in the efficient use of bandwidth. Terascale visualization problems involve movement of large amounts of data from spinning disk to software components that transform the data into visible pixels. End-to-end, problems of this magnitude require a staggering amount of bandwidth and processing power. When the data, software components used for visualization, and users are separated by a network, then efficient bandwidth management becomes an even more critical concern. This chapter describes a highly specialized visualization application, Visapult, which performs high performance remote and distributed visualization. Visapult is perhaps best known for its ability to effectively utilize the network, having been used to win the High Performance Network Bandwidth Challenge at the annual IEEE/ACM Conference on Supercomputing three years in a row (2000-2002).

Introduction

During the period of the Next Generation Internet Combustion Corridor project (1999-2000), there was a conundrum: the network research community declared there were no applications that could take advantage of high speed networks, yet applications developers were confounded by poorly performing networks that impeded high performance use. Our goal during that period of time was to create an extremely high performance visualization application that was not only capable of pushing the performance envelope on wide area networks, but which was also generally useful from a scientific research perspective. We wanted an application that would perform remote interactive visualization of large scientific data sets, but which would not suffer from the delays inherent in network-based applications. We wanted our application to use multiple, distributed resources so that data, user and required computing machinery need not be collocated. Such objectives are consistent with the needs of contemporary scientists: large datasets are often centrally located, with many remote users needing to view and analyze the data.

Our answer was to create the Visapult application. Visapult is a visualization application composed of multiple software components that execute in a pipelined-parallel fashion over wide-area networks. By design, Visapult was tailored for use in a remote and distributed visualization context. The first use of Visapult was for visualizing turbulent flow simulation data computed on supercomputers at the National Energy Research Scientific Computing (NERSC) by a researcher located at Sandia National Laboratories in Livermore, California during the early part of 2000. Over time, we broadened our scope of efforts to include a more careful study of and more deliberate use of networking infrastructure, as well as refining the Visapult design to maximize performance over the network. Visapult is arguably the world's fastest performing distributed application, consuming approximately 16.8 gigabits per second in sustained network bandwidth during the SC02 Bandwidth Challenge over transcontinental network links. Visapult's performance is a direct result of architecture, careful use of custom network protocols and application performance tuning.

In this chapter, we "reveal the secrets" used to create the world's highest performing network application. In the first section, we present an overview of Visapult's fundamental architecture. Next, we present three short case studies that reflect our experiences using Visapult to with the SC Bandwidth Challenge in 2000, 2001 and 2002. Finally, we conclude with a discussion about future research and development directions in the field of remote and distributed visualization.

Visapult Architecture

Visapult is a highly specialized and extremely high performance implementation of and extension to Müller et. al's Image Based Rendering assisted volume rendering [Müller99] (IBRAVR) approach to lightweight volume rendering. Visapult consists of two software components, a *viewer* and a *backend.* The viewer component implements the IBRAVR framework using OpenRM Scene Graph [Bethel01], an Open Source

scene graph API. The viewer receives the source images it needs for the IBRAVR algorithm from the backend component. Each of the viewer and backend components are implemented as parallel applications, and they communicate using a custom TCP-based protocol to exchange information (control data) and images (payload data). Figure 1 illustrates the architecture, which is discussed in substantially more detail in an earlier publication [Bethel00].

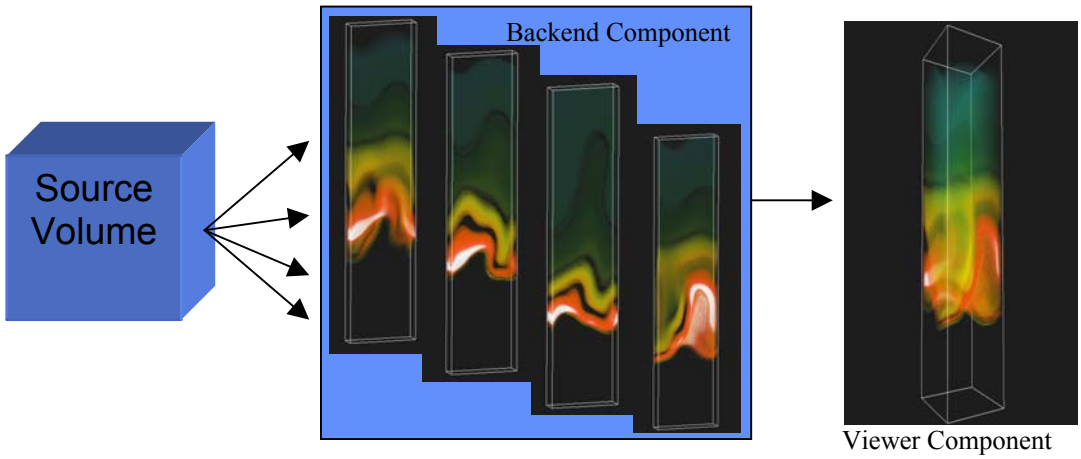

Figure 1. Visapult consists of a viewer component and a backend component.

Looking more deeply into each of these components, we see that each of the *backend* and *viewer* components is in turn a parallel application. The backend is a distributed-memory parallel software component written using MPI. Each backend PE is responsible for reading and rendering a subset of a 3D volume of data, and sending the resulting image to a peer listener in the viewer. The viewer is also a parallel application, but uses threads and a shared memory model. The viewer creates one thread per backend PE to receive image payload data. Each of these listener threads is responsible for updating a portion of a thread-safe scene graph. A detached rendering thread performs interactions with the user and invokes the scene graph system's frame-based rendering function to generate new images.

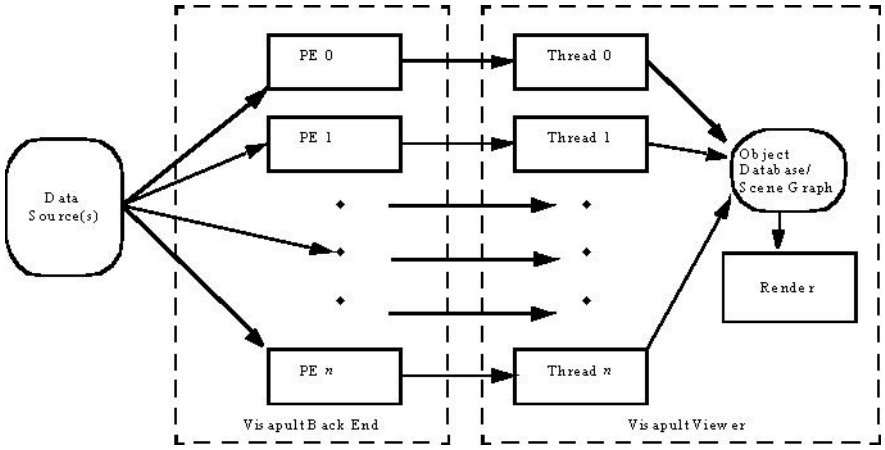

Figure 2. Visapult's End-to-End Data-Parallelism: Major blocks represent components that can be co-located or separated via long-haul network connections. Vector graphics image?

Our earliest Visapult deployment tests in late 1999 used data sources that were resident on the same platform as the backend PEs, and our attention was focused upon the wide-area networking behavior between the backend and the viewer. We wanted to use computational resources located "close to" the data, and provide interactive visualization capabilities to a remotely located user. With such requirements, our primary concern was on the network link between the backend and viewer.

However, in preparing to use Visapult for the SC2000 bandwidth challenge, we wanted to stretch the limits of our design and use the emerging high-speed network backbones in order to test the hypothesis that pipelined-parallelism, over a WAN, was useful for remote and distributed visualization. During early field tests, we had established that Visapult's pipelined architecture indeed supported interactivity on the desktop regardless of the performance of the underlying network. Early in Visapult's evolution, the principal developer routinely used a DSL connection between the backend and viewer. Even with a DSL line, the viewer was completely interactive on the desktop; the listener threads would accumulate the next frame's worth of partial pre-rendering before updating the scene graph. Meanwhile, the freely running rendering thread would continue to operate at desktop rates. This behavior is possible because of the substantial amount of data reduction that occurs as scientific data traverses the Visapult pipeline: incoming data is of size $O(n^3)$; the backend renders data down to images, resulting in data of size $O(n^2)$.

We were motivated to find a way to successfully capitalize upon an emerging trend within the computational science community: a few, high-powered resources are interconnected via high-speed fabric, and provide service to users regardless of location. The idea is that data produced by simulations is stored on data caches located near, in network space, to the resource used to perform the simulation. Similarly, subsequent visualizations are performed by remotely located users, leveraging resources that are interconnected by high performance networks. Ideally, the user need not be aware of the vagaries of accessing remote resources; all such connections are automatically "brokered" by an intermediate agent. Visapult's design – using IBRAVR to reduce the data load between backend and viewer by an order of magnitude – fits well within this model. With these objectives in mind, we began to explore the use of high speed, network-attached data caches.

Reading Data Over the Network Using TCP

To begin the process of using such data caches, we added support in Visapult to use the Distributed Parallel Storage System (DPSS) [Tierney99], created by the Data Intensive and Distributed Computing group at Berkeley Lab. The basic idea behind DPSS is "RAID-0 over the network." To the DPSS client, such as each backend PE, the DPSS looks like a file that is accessible through *open*, *seek*, and *read* calls. DPSS is a block-oriented system composed of commercial, off-the-shelf components and the DPSS library (see Figure 3). A client application, using the DPSS library, issues a "file open" call, which in turn results in many different files being opened on the DPSS servers, all mediated by the DPSS master. Next, the client's "read" calls are routed to the DPSS master, but answered by the DPSS servers. The client application does not know that it is in fact talking to multiple machines on the network, since the DPSS libraries encapsulates those details. We'll discuss the performance characteristics of DPSS in the section on the SC00 Bandwidth Challenge.

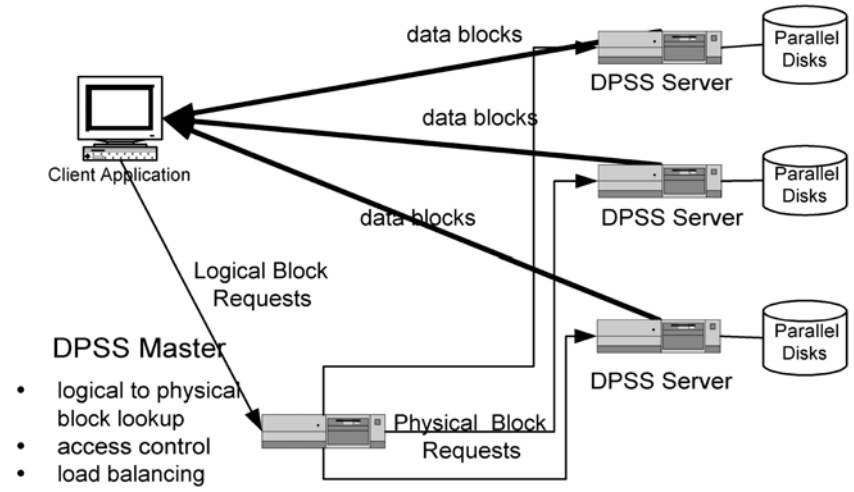

Figure 3. DPSS Architecture

During field testing, we used a DPSS system located in Berkeley, and two separate computer systems located at Sandia National Laboratory in Livermore, CA. One Sandia system was an SGI Origin system, the other the "CPLANT" facility, which is a large, Alpha-based Linux cluster. Between the two sites was an experimental OC-48 NTON[1] network connection. Despite the substantial amount of computing and NIC horsepower, we were not satisfied with performance results, which indicated we were consuming only a small fraction of a dedicated OC-48 link between two sites (about 10%-15%).

Based upon initial performance data, we began to perform careful execution profiling of the Visapult application using a tool called Netlogger [Tierney98]. In order to use Netlogger's profiling capabilities, you must instrument your application by inserting subroutine calls to the Netlogger library. When the instrumented application is executed, the Netlogger code invoked from the application sends data to a Netlogger host, which accumulates data (similar in fashion to the familiar syslog facility in Unix systems). After a run has been completed, Netlogger's analysis and display tool presents the profile data obtained during the run for visual inspection. Netlogger's strength lies in the ability to perform profiling and execution analysis of distributed software components.

In Figure 4, we see the results from profiling the Visapult backend and viewer for a sample run. The graph on the top shows the execution profile of the Visapult viewer. The graph on the bottom shows the execution profile of the Visapult back end. The horizontal axis of the graph is elapsed time, and the vertical axis represents instrumentation points in the code. For the purposes of this discussion, we are concerned only with the performance of the Visapult back end, since the "heavy payload" network link was between DPSS and Visapult's back end. The profile graph of the back end is colored so that even-numbered frames are show in red, and odd-numbered frames are displayed in blue. The observation we can make from the performance profile shown in Figure 4 is that the network performance will never reach its theoretical maximum value since I/O is blocked while rendering occurs. Data is loaded into the back end between the vertical axis labels "BE_LOAD_START" and "BE_LOAD_END." Then, back end rendering occurs between "BE_RENDER_START" and "BE_RENDER_END." The result is "gaps" of time when there is no network I/O.

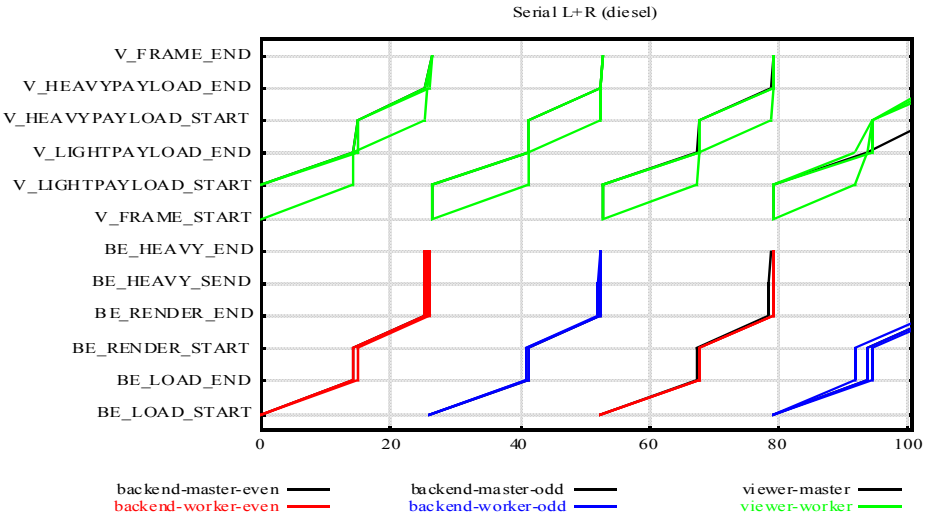

Figure 4. Visapult duty cycle with serial I/O and rendering.

[1] NTON – the National Transparent Optical Network testbed – was an experimental WAN that consisted of dedicated OC-48 (and higher) segments connecting sites on the West Coast of the United States. The NTON website, www.ntonc.org, is now defunct.

As a result of the performance analysis shown in Figure 4, we modified the Visapult back end so that rendering and network I/O were placed into separate threads of execution[2]. By doing so, we were able to maintain a constant load on the network, thereby eliminating one source of network inefficiency in Visapult. Figure 5 shows the profile analysis resulting from having I/O and rendering occur simultaneously in the back end. So long as software rendering speed exceeds the time required to load data over the network, the network link will be kept as full as possible. As long as the time required for rendering is less than the time required to send data over the network, the network will remain completely filled, or so we thought. The reality turned out to be quite a bit different.

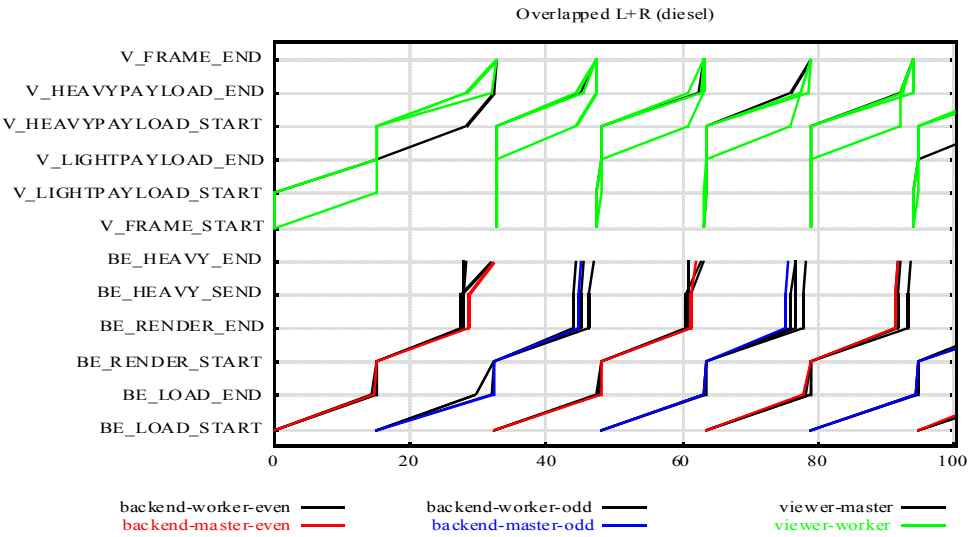

Figure 5. Visapult duty cycle with overlapped I/O and rendering.

The Bandwidth Challenges and Results

Beginning in 2000, a new High Performance Computing contest was announced: the High Performance Network Bandwidth Challenge. The primary objective of the competition was to use as much network bandwidth as possible within a window of time. Secondary objectives included most creative use of network bandwidth, and so forth. The "real prize," however, was unabashed and gluttonous consumption of resources. We felt, based upon the results of our field tests and performance analysis, that we had a reasonably good chance to make a competitive showing at the Bandwidth Challenge using Visapult. The sections that follow describe out experiences in the SC Bandwidth Challenge during the years 2000-2002. As we shall explain, we learned many lessons during these competitions.

SC 2000 Bandwidth Challenge

For the SC 2000 Bandwidth Challenge (BWC), we teamed with Helen Chen and Jim Brandt from the Networking Security Research Group at Sandia National Laboratory, Livermore CA, and with Brian Tierney, Jason Lee and Dan Gunter of the Data Intensive and Distributed Computing group at LBNL. The collection of resources we used consisted of an eight-node DPSS system located at Berkeley Lab, an OC-48 connection to SC00 in Dallas, Texas, and a pair of hosts on the show floor. One host was an eight-node SGI Onyx2, located in the ASCI booth. The other was a small Linux cluster, located in the SC00 booth of Argonne National Laboratory (ANL). We ran two separate Visapult backends, one on each of these ASCI and ANL platforms, and ran Visapult viewers in the LBL booth (see Figure 6).

[2] In MPI-parallel applications, it is possible for the MPI application to launch detached threads, but those threads typically cannot directly participate in MPI communication (only the parent/master threads can). The converse is not true.

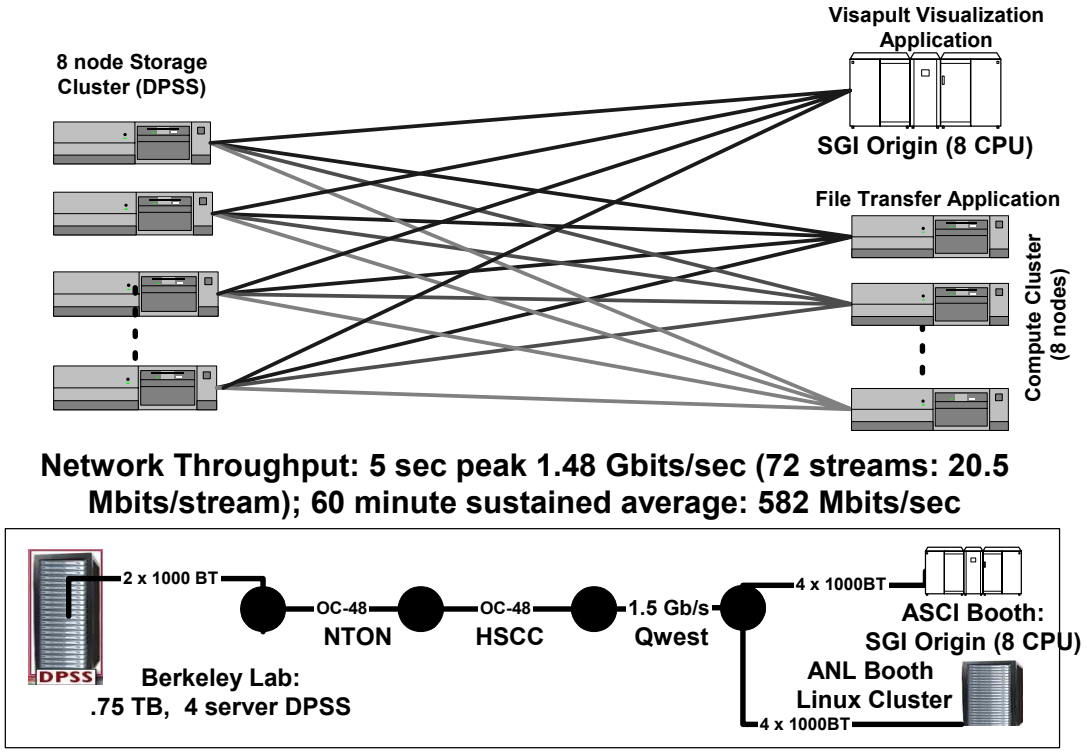

Figure 6. SC 2000 Bandwidth Challenge Resource Map

Figure 7 shows the performance of our application, as measured in the SCinet NOC during our run. Despite having adequate computing power, and a dedicated OC-48 link, we were able to consume only a fraction of the available network bandwidth. The "glitch" in the middle of our run occurred due to an application crash, which required restarting the DPSS. During our 60-minute run, we achieved a peak bandwidth rate of about 1.54 Gbps, with a sustained average of about 582 Mbps.

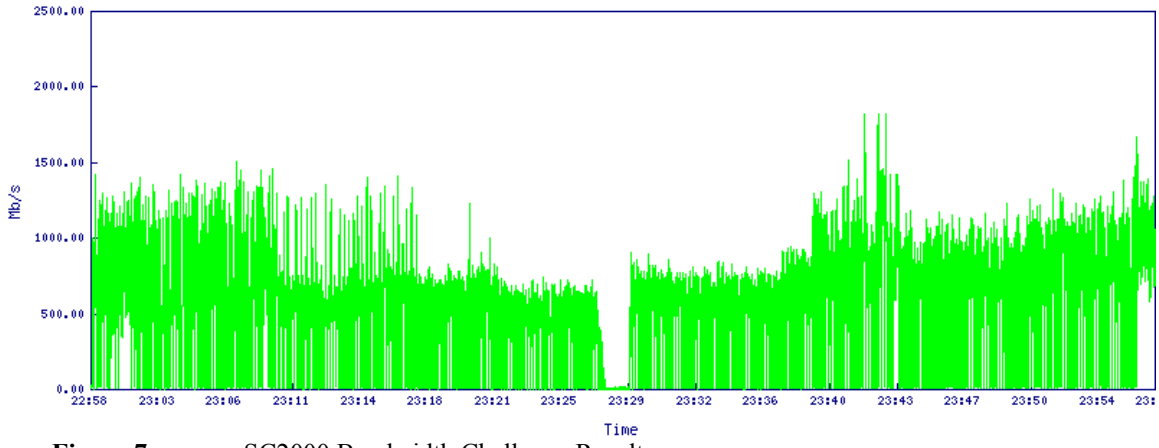

Figure 7. SC2000 Bandwidth Challenge Results

The fundamental reason Visapult was not able to sustain greater than 25% of the theoretical network capacity on the OC-48 link between Berkeley, CA and SC 2000 in Dallas, Texas was due to TCP, not application design. Visapult and DPSS were able to load the network with data, but the flow control algorithm used by TCP resulted in poor throughput, even on an unloaded network. It is well known that single stream TCP performance is poor on high bandwidth networks. The response is to use a multistreamed approach, which is exactly what we did for our SC 2000 BWC entry. The TCP congestion

avoidance algorithm regulates the rate at which packets are dispatched onto the network in response to perceived congestion events. It performs such control on a per-stream basis, which means that any flow regulation activities performed on one stream are independent of all other streams. In other words, a congestion event occurring on one stream, along with the resultant "TCP backoff" (reduction in bandwidth usage), will not affect peer TCP streams. This method creates a network stream that behaves as a "bully," for the bully doesn't respond as rapidly to congestion as its peers. As you can see from the performance graphs in Figure 7, the result can be best characterized as "unstable." Such instability has a huge impact on the quality of interactive visualization applications as they stutter and halt, while the TCP stream's data rate "thrashes" in response to perceived congestion.

SC 2001 Bandwidth Challenge

Based upon the lessons we learned during SC 2000 – namely, that TCP is inadequate as a high performance network protocol – we tried a different approach for SC 2001 BWC entry. For our SC 2002 BWC entry, we modified Visapult to use a custom, application-level protocol based upon the User Datagram Protocol (UDP) method of packet movement. The fundamental difference between TCP and UDP is that TCP guarantees delivery of packets in the order sent, whereas UDP makes no such guarantees. In UDP, packets could arrive in any order, or might not arrive at all. In terms of performance, TCP uses mechanisms that control the rate at which packets are sent over the network. In contrast, UDP has no such flow regulation. The flow congestion avoidance algorithm used by TCP is widely recognized as the reason TCP-based flows are unable to realize more than 25-30% of the theoretical line rate. We felt compelled to use UDP in order to achieve the maximum possible level of network performance.

UDP-based flows are "unregulated" in the sense that there is no flow control provided by the IP stack. Whereas TCP automatically adjusts its flow rate in response to environmental conditions, UDP flow-rate management must be performed by the application. One can mimic TCP's behavior using Madhavi and Floyd's TCP-friendly method [Madhavi97], but in doing so, will suffer the same corresponding performance loss that we've come to expect of TCP. Rather than attempting to infer congestion from packet loss rates at runtime, we decided to carefully select a flow rate based on the measured end-to-end capacity of these dedicated links, allowing us to realize nearly 100% of the theoretical line rate. While abandoning TCP is considered "cheating" in some circles, we feel that we were merely addressing the problem at hand in the most direct manner possible. Under these conditions, we felt that we would be able to realize much higher rates of bandwidth utilization by employing UDP than would be possible with TCP, and our SC 2001 BWC entry was the field test for our hypothesis.

Unfortunately, replacing TCP with UDP was not as straightforward of replacing one set of subroutine calls with another. In our SC 2000 TCP-based implementation, Visapult would request a large block of data from the DPSS with a single read call. The large block of data represented one timestep's worth of data. Then, after the data for a given timestep had arrived, the backend would then begin rendering (and at the same time, the next timestep's worth of data was requested). One of the nice properties of TCP is that the application can ask that *N* bytes of data be moved from one machine to another. TCP breaks the *N* bytes worth of data into packet-sized chunks on the outbound side, and reassembles the *N* bytes of data from individual packets on the receiver side. In contrast, UDP-based applications are completely packet-oriented: a there is no mechanism built into the IP stack that breaks large data blocks into packets on the sender side, nor that reassembles packets into blocks on the receiver side[3]. The maximum amout of data that can be sent by a UDP-based application with one transfer is one packet's worth of data[4]. It is up to the application to partition large data blocks into packets for transmission by UDP, then to reassemble the packet payload data into a large data block on the receiving end.

[3] In IPv4, large packets are automatically fragmented and reassembled. This activity is not supported in IPv6.

[4] The size of a network packet is defined by the Maximum Transmission Unit (MTU) which is the minimum of the configuration parameter of a Network Interface adapter and the minimum packet size supported by all intervening switches along a given path on the network. Typically, the Ethernet MTU is 1500 bytes. So-called "jumbo frames" use an MTU of 9000 bytes.

Another difference from our previous entry was the use of a live-running simulation as a data source, rather than using precomputed data stored on the DPSS disk cache. The simulation was built using the Albert Einstein Institute's Cactus framework (http://www.cactuscode.org), which models the collision of binary black holes by directly evolving Einstein's equations for General Relativity. This particular simulation has substantial computational demands, including the need for exorbitant amounts of physical RAM and processing power in order to solve a very large set of equations. More information about General Relativity is provided in (http://jean-luc.aei.mpg.de).

In order to enable use of UDP in Visapult, we needed to do away with "frame boundaries," while at the same time using the existing backend architecture that provided for simultaneous network I/O and rendering. In addition, we needed to provide enough information in each UDP packet so that each packet could be treated independently. To that end, we encoded contextual information into the header of each UDP packet. The information encoded into the header was sufficient to identify the location in the computational grid where the data payload should be placed. In this way, placement of the payload data, or the actual simulation data we wanted to visualize, was independent of the order in which packets arrived. The encoding scheme we used is described more fully elsewhere [Shalf03], and resolved the problem of packets arriving out of order.

To a large extent, we can avoid creating conditions in which packets are dropped by regulating the rate at which UDP packets are placed onto the network. Generally speaking, packets are lost because of unrecoverable bit errors that are detected by the Cyclic Reduncancy Check (CRC) or because of buffer overflows at either the endpoints or congested switch interfaces in the network core. Our testing showed a very low rate of packet loss when we manually regulated the flow to match the known end-to-end network capacity. This approach is exactly what we used in our custom UDP protocol between Cactus and Visapult. While we can minimize packet loss through careful flow regulation, packets are still sometimes dropped. In our experiments and testing, we find the impact of a few lost packets on the resulting visualization to be negligible. In fact, we observe that many well-accepted forms of media are predicated upon lossy compression methods. JPEG and MPEG compression, for example, typically produce 10%-80% of information loss during the compression process, but the resulting images are visually quite acceptable. On the other hand, packet loss is entirely unacceptable in certain critical systems, such as medical applications. For the purposes of visualization for remote monitoring, however, consumers of this technology are well-accustomed to lossy representations of information. An example of a Cactus/Visapult visualization, complete with missing data, is shown in the "Lessons Learned" section.

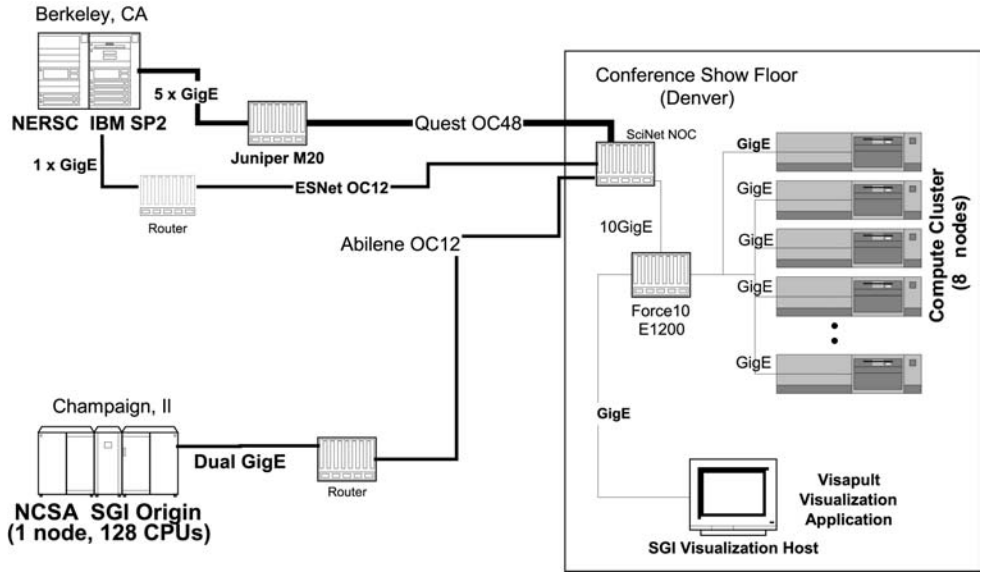

Figure 8. Resource Map for SC2001 Bandwidth Challenge

The demonstration involved a simple task-spawning scenario with the Cactus code. In this case a large simulation typically runs for a few days on the 5 Teraflop NERSC SP-2 simulating the merger of two black holes. During the course of the simulation, events occur such as the joining of the event horizons of the two black holes that require additional analysis. Rather than interrupt the current simulation, Cactus can spawn off the "horizon finder" to perform that specialized analysis on the SGI Origin supercomputers at NCSA as a slave to the master simulation running at NERSC. Visapult's distributed component architecture made it very simple to use our "visualization cluster" to process large quantities live data arriving from multiple, widely distributed simulation resources in real-time.

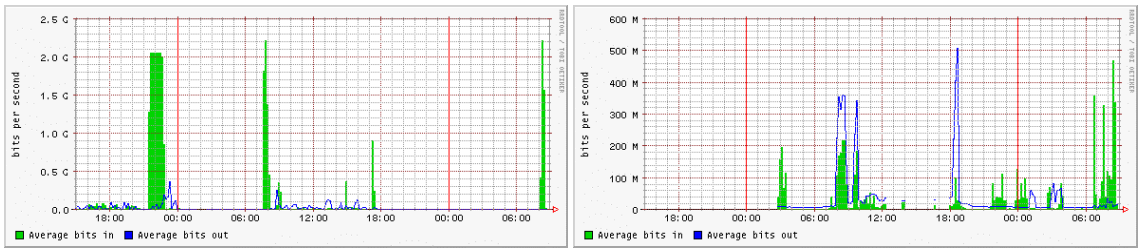

Figure 9. These are images of the MRTG performance graphs from the incoming datastreams arriving from Esnet OC-48 and NCSA OC-12 respectively for the SC2001 Bandwidth Challenge Application. The challenge took place for approximately 7 minutes at 8:30AM. The performance MRTG graphs actually under-report the bandwidth utilization because they use a 5 minute average. The instantaneous bandwidth (which also included an additional ESNet OC-12) reached 3.3 Gigabits/sec. The 2+ Gigabit peaks on the ESNet graph were practice runs using Visapult.

The SCInet network engineers who were monitoring the performance of our challenge entry were shocked to see the ramp-up of the data-stream was instantaneous, whereas a typical TCP-slow-start would take several minutes to reach full line rate assuming absolutely no packet loss. Within 7 minutes, we were able ramp up beyond 3 Gigabits; double the performance of the nearest competing application (which was also UDP-based). For the first time in many years, we were able to fully utilize a dedicated high-bandwidth pipe—a feat had been increasingly difficult to achieve given TCP's inadequacies.

SC 2002 Bandwidth Challenge

Because of the success of our SC 2001 entry, along with results we obtained in the laboratory during the Summer of 2002[5], our SC 2002 entry was intended to literally "flatten" any network we could get our hands on. As it turned out, garnering resources for the SC 2002 entry proved to be the real challenge. The result is reflected in the diversity of team members and resources we assembled for the SC 2002 run. The SC2002 entry was a transglobal collaboration involving supercomputers, networking resources and people from around the world drawn from a different project known as the "Global Grid Testbed Collaboration" (GGTC)[6].

[5] In July of 2002, we were able to completely fill a 10G network link using the Cactus/Visapult combination. See http://www.supercomputingonline.com/article.php?sid=2252.

[6] The Global Grid Testbed Collaboration was organized during the Global Grid Forum Applications Working Group meeting in Chicago in the Fall of 2002. The GGTC includes people and resources from five continents and over fourteen countries that aggregated about 70 machines totaling approximately 7500 processors. Architectures range from a five Teraflop SP2 system to a Sony Playstation 2 running Linux. See http://scb.ics.muni.cz/static/SC2002/

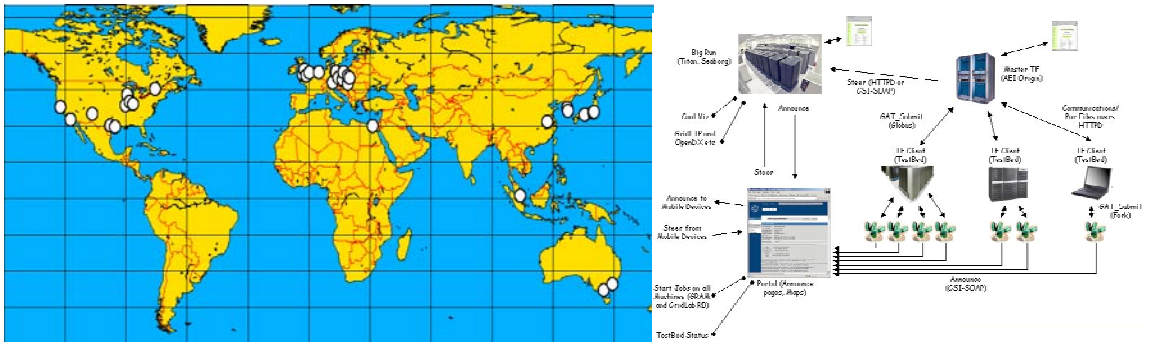

Figure 10. To the left is a map of sites participating in theSC2002 Bandwidth Challenge Global Grid Testbed Collaboration. On the right is the logical workflow of the Grid task-farming scenario. The task farmer uses an application level abstraction layer for Grid Services called the Grid Application Toolkit (GAT). For more information on the GAT, please refer to http://www.gridlab.org.

Our bandwidth challenge application expanded on the task-spawning scenario of the SC2001 entry. In the task-spawning scenario, the primary simulation spawns dozens of smaller subtasks during execution in order to perform a run-time parameter study with results that feed-back to the master simulation. The master simulation responds by making adjustment to the spawned simulations -- in effect steering them. For instance, the master simulation can spawn a parameter study to determine the effect of adjusting its courant factor. The result might show improvement in evolution rate, but also show an unexpected impact on the accuracy and stability of the numerics. The primary simulation code could actually scan for available resources worldwide to launch slave simulations to explore all of those possibilities. Visapult fit into this framework as a means to visually inspect the results of these slave simulations in real-time using volume rendering.

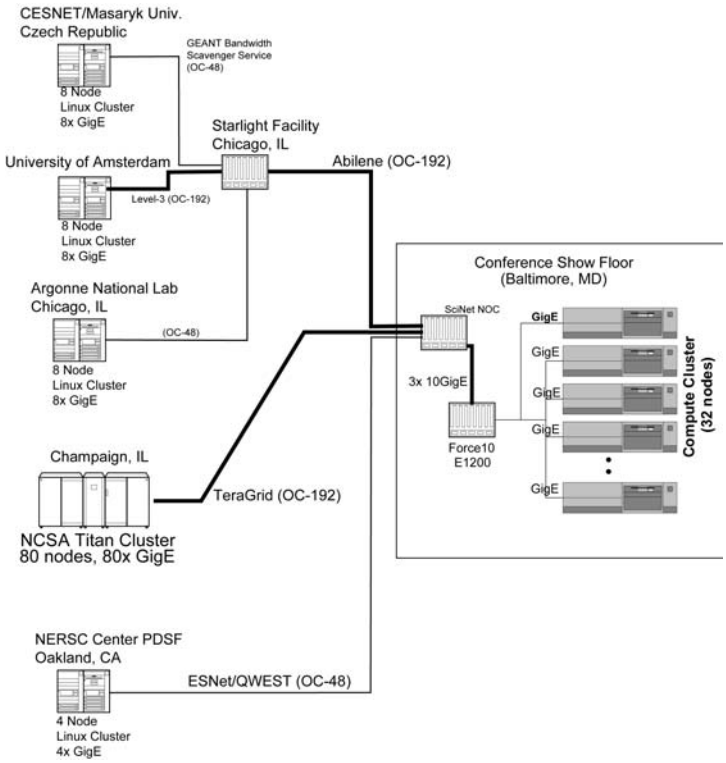

Figure 11. SC02 Bandwidth Challenge Resource Map

For the purpose of the Bandwidth Challenge, we focused our effort on 5 well-connected sites; NCSA in Champaign Illinois, NERSC in Oakland California, Argonne National Laboratory in Chicago, The University of Amsterdam, and Masaryk University in the Czech Republic (Figure 11). During the actual challenge run, we opted to force manual launching at these strategic sites in order to reach full bandwidth within the 15 minutes we were allotted for the benchmarked run. The resulting run managed to push an unprecedented 16.8 Gigabits/sec of data to the LBNL cluster on the SC showfloor in Baltimore from these worldwide-distributed simulation sources. Figure 12 shows the SCinet "weathermap," which depicts the bandwidth consumed over each of the three inbound WANs used as part of our run. Note that the sum of bandwidth across the three WANs totals 17.2 Gbps. Our metered average rate of 16.8 Gbps covers a 15-minute window, and we were ahead of the average rate at the time when Figure 12 was made.

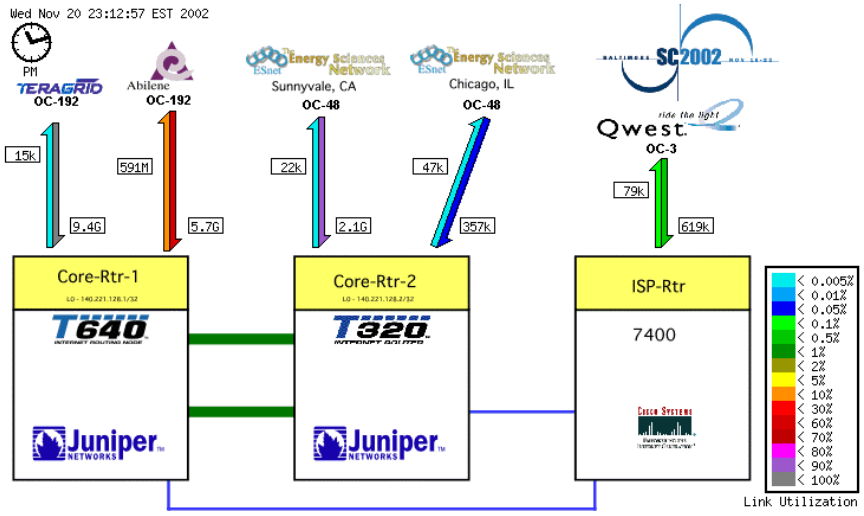

Figure 12. SCinet Weathermap

In contrast, Figure 13 shows the bandwidth measured between the SCinet NOC outbound to the three 10-Gigabit connections to the LBL booth. Figures 12 and 13 are two different views of the same network traffic. While Figure 12 shows flow rates on each of the inbound WANs, Figure 13 shows the flow rate on the LAN side of the SCinet NOC. During the SC02 run, we achieved a peak transfer rate of about 17.21 Gbps, and had a minimum transfer rate of about 15.04 Gbps. During our 15-minute window, we transferred a total of approximately 138.26 Tbits of data from the remote resources into the Visapult backend.

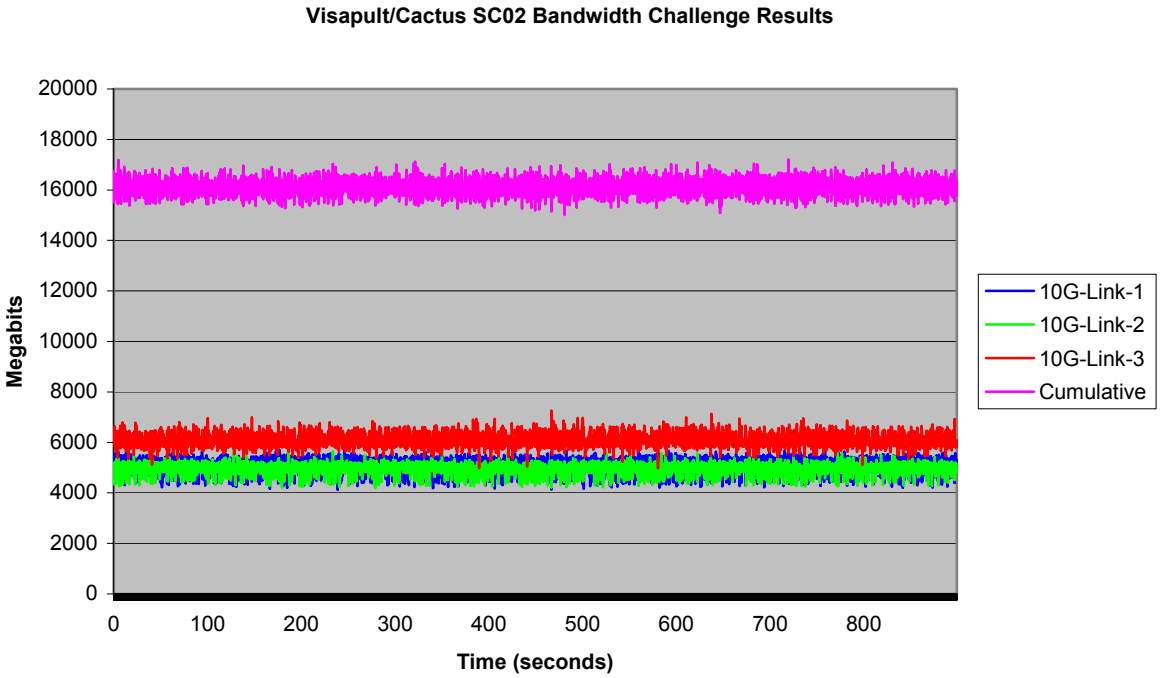

Figure 13. SC02 Bandwidth Challenge Results

Visapult itself was extensively rewritten during the time between the SC01 and SC02 BWC events. Whereas the original Visapult IBR used slices exclusively in the Z-direction to create the illusion of volume rendering, the SC2002 version supported omnidirectional viewing by automatically changing the choice of compositing slices as a function of view angle (Figure 14). The Visapult backend was rewritten to accommodate block decomposition to support the new rendering capabilities, and also to improve overall load balancing and flexibility in the visualization process.

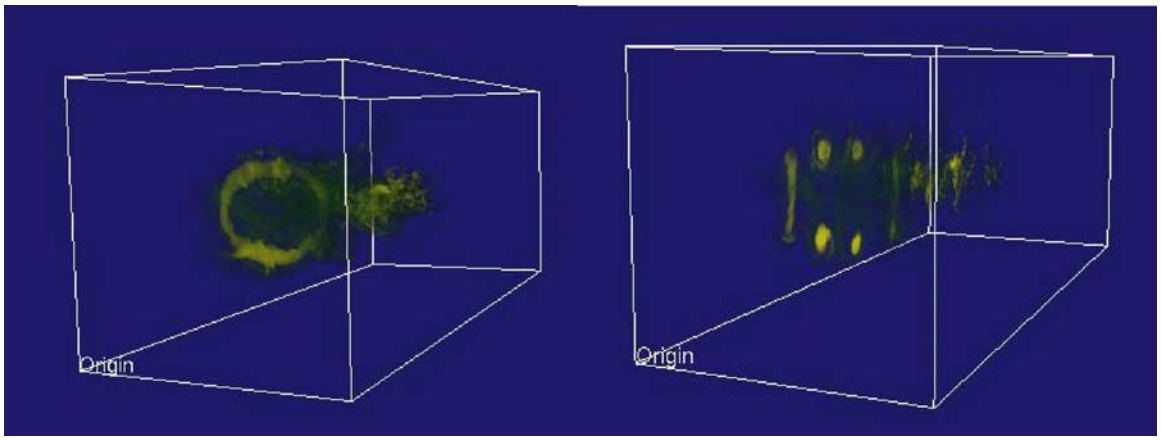

Figure 14. Visapult's new "Omniview" capabilities (left) produce much better visual fidelity than the original IBRAVR algorithm.

The Visapult/Cactus protocol was enhanced significantly to support auto-negotiation of load-balanced data-parallel UDP streams between the Cactus and the backEnd in contrast to the static mapping required by the SC2001 version. Finally, the accuracy of the packet-rate regulation method used by Cactus was greatly improved by moving from a static inter-packet delay mechanism to a method based on error diffusion. Errors from previous packet sends were carried forward for computing the inter-packet delay of the next

packet send, thereby achieving a high degree of accuracy in regulating the flows despite the coarse granularity of available system timers.

Of critical importance was the fact that this bandwidth challenge was part of a larger fabric of interconnected Grid services and applications. For instance, all of the spawned simulation tasks were tracked automatically using a Grid Portal [VonLaszewski02]. We could even follow links the portal using a web browser to directly connect to the running simulation codes to monitor its progress, steer it, adjust network bandwidth, and restart or kill jobs running anywhere in the world. This infrastructure even allowed us to dynamically attach to running jobs to perform live/interactive visualization to the emerging results we did with Visapult. Such access was even available from handheld devices like iPAQ handheld computers and cell phones. Some testbed applications would even leave SMS messages on the scientists cell phones to keep them apprised of their progress and location. The advances in this emerging Grid infrastructure and application-level services were nearly as exciting and dramatic as the increased network bandwidth and global connectivity that such applications engender. This event gave us a glimpse into a future of the Grid – a framework composed of independently developed components that can be tightly woven together into a pervasively accessible global infrastructure. Future development of Visapult and other RDV frameworks at LBL will focus on this larger view of component architectures for distributed high performance computing on the Grid rather than any individual stand-alone application.

Lessons Learned

There are two primary lessons we learned while evolving Visapult to make effective use of the network in a remote and distributed visualization context. First is that TCP, the most commonly used network transport protocol, is very ill-suited for use in high performance network applications. In order to effectively use high bandwidth networks, you must avoid use of TCP. The ramifications of such a strategy are profound and far-reaching in terms of application design. Second, we learned that using UDP, which is an alternative to TCP for point-to-point network communication and is much better in terms of using available bandwidth, comes with a set of costs: applications must be resilient to dropped network packets, and applications designed for TCP will probably need to be retooled for use with UDP.

Don’t Bet on TCP for High Performance Network Applications

TCP’s algorithm for regulating packet flow rates is having great difficulty meeting the needs of high performance networking. If one takes a dramatically simplified view of the problems facing the aging TCP protocol, you could focus on two causal factors out of a field of many. First is the TCP congestion avoidance algorithm’s assumption that any packet loss indicates network congestion that requires rapid reduction in the flow rate (a fundamental assumptions about the meaning of packet loss). The second is the slow rate at which TCP returns to full speed in response to packet loss. In order to understand why TCP is inadequate for today’s high performing networks, we must look at how the TCP congestion avoidance algorithm works, and how that algorithm is inappropriate for use on modern, high performance networks.

Two key parameters used to characterize network performance are latency and bandwidth. The network latency is affected by buffering in the network routers and switches as well as the responsiveness of the hosts at the endpoint. However, it is primarily bounded by the fundamental limit of the speed of light. Bandwidth, on the other hand, has been increasing dramatically over the past decade—outpacing Moore’s Law by four times. The TCP congestion avoidance algorithm was created to deal with rate management on networks that peaked at one to five megabits in the late 1980s. When a packet is lost in a transfer, TCP immediately assumes the loss indicates congestion, and “backs off” to half its current flow rate. The rate at which it can return to its original flow rate is bounded by the round-trip-time (RTT) of the network. When the product of the bandwidth of the network and this round-trip-time (the “bandwidth-delay product”) is a small number, as it was in the late 1980s, the rate of recovery was not particularly noticeable.

With networks now pushing four orders of magnitude more bits per second with very similar latencies, the TCP congestion avoidance algorithm is proving to be a hindrance. As bandwidth increases, so too does the exposure to the packet loss. Generally speaking, packets are lost because of unrecoverable bit errors that

are detected by the Cyclic Redundancy Check (CRC) or because of buffer overflows at either the endpoints or congested switch interfaces in the network core. Given the comparatively large bandwidth-delay products involved, the stream cannot recover as quickly from lost packets despite its increased sensitivity to the loss. In fact, using TCP on a 10 gigabit/second network connection, it would take 1 2/3 hours of continuous transmission without a single lost packet to achieve full line rate[7]. Increasing the rate that TCP can recover from packet loss events makes the TCP algorithm most unstable, as it oscillates around its quiescent non-congestive data rate. Consequently, the Additive Increase Multiplicative Decrease (AIMD) algorithm employed for TCP congestion avoidance (common knowledge…you can quote Stevens if you want) is necessary for stability from the control-theoretical standpoint. It also guarantees increasingly inefficient use of the network resources as the bandwidth-delay product increases. Based upon empirical evidence, the asymptotic behavior results in a tendency to send fewer and fewer packets, with steady state reaching of about 20%-25% of the maximum line rate. As network bandwidth continues to increase, these efficiencies will continue to plummet.

Therefore, the time has come for a fundamental paradigm shift in the way that congestion is managed on the WAN. Some have proposed having the network infrastructure provide direct feedback to the TCP/IP stacks of the endpoint hosts as to current congestive conditions on the network ([Floyd94],[Stoica98],[Katabi02]) but these methods require dramatic changes to the routers and switches that comprise our current network infrastructure. Other methods attempt to monitor the end-to-end performance of the network externally using probes ore instrumented TCP kernels to provide this information ([Agarwal02],[DMF],[Dunigan02]) but can suffer from incomplete coverage or weak ability to understand anomalous behavior in the network core. Since we started using this fixed data rate method for our transport, we've seen a number of other efforts adopt a similar manual-rate-control methodology, including Tsunami and SABUL ([Gu02], [Meiss02]). While we believe the latter approach is the most effective given current circumstances, it is not clear how it will scale as its use becomes more pervasive. In the special case of HPC applications, it is not uncommon to see dedicated links, such manual control is entirely reasonable. However, over time there will be an increasing need for brokers to mediate these fixed-bandwidth data streams.

How Well Does a Lossy Transport Mechanism Perform for Visualization?

When using TCP for data transport, data loss is not an issue. TCP guarantees delivery of packets, and does so by ordering the packets, and asking for retransmission if they are lost along the way. The result is that TCP is slow. When using UDP for transport, which is very fast, we will occasionally lose packets. Just how much of an issue is the absence of a few packets?

The approach we used for encoding data into UDP packets results in portions of the data from the computational grid being copied into the packet. To maximize efficiency on the sender side, we copy data from adjacent memory locations into the data payload portion of the packet. Typically, these regions consist of partial "scanlines" from the source volume. When a packet is dropped, we don't have data for that particular scanline when doing the visualization. Instead, what we have is data leftover from the previous timestep that was loaded into memory on the receiver side. Keep in mind that we use a single-buffering approach for the receive buffer; there is no notion of "frame boundaries," which correspond to one time step's worth of simulation data, so there is no advantage to using a double-buffered approach on the receive side. We simply receive data as fast as we can, and asynchronously render it. Due to the asynchronous nature of the receive/render cycle, there is no design guarantee that at any given point in time, that the receive buffer will have data from just one simulation time step.

Another advantage of this approach is the nearly immediate delivery of data to the application. TCP, in its attempt to enforce the in-order presentation of data, must actually hide received data from the application until all gaps in the packet stream are filled. If there is a gap in the stream because a packet is lost, then the receiver is blocked while the source is notified of the loss, and the packet retransmitted. The result is that

[7] An observation by Sally Floyd's IETF draft High Speed TCP document http://www.icir.org/floyd/papers/draft-floyd-tcp-highspeed-02.txt

the receiver spends a good deal of time being idle waiting for data that has already arrived and is simply held by the OS in an opaque system buffer. For interactive graphics applications, the application performance is irregular, and is very noticeable and annoying to application. In contrast, UDP always delivers the information to the application almost immediately after it arrives. When using UDP, the application must have a graceful way to deal with loss and ordering issues. Visapult demonstrates that such management is possible and provides very reasonable results.

The following figure shows six different snapshots of an evolving simulation. In the upper left corner, the simulation has just started. You can see portions of the receive buffer that have not yet been filled in with simulation data – you can see the background through the volume rendering. Moving to the right, these regions not begin to "fill in" with data while the simulation continues to evolve. This particular example was created using a rate limited flow of approximately 10Mbps over a 100Mbps link. Since we were using only a fraction of the total available bandwidth, the likelihood of packet loss is quite small.

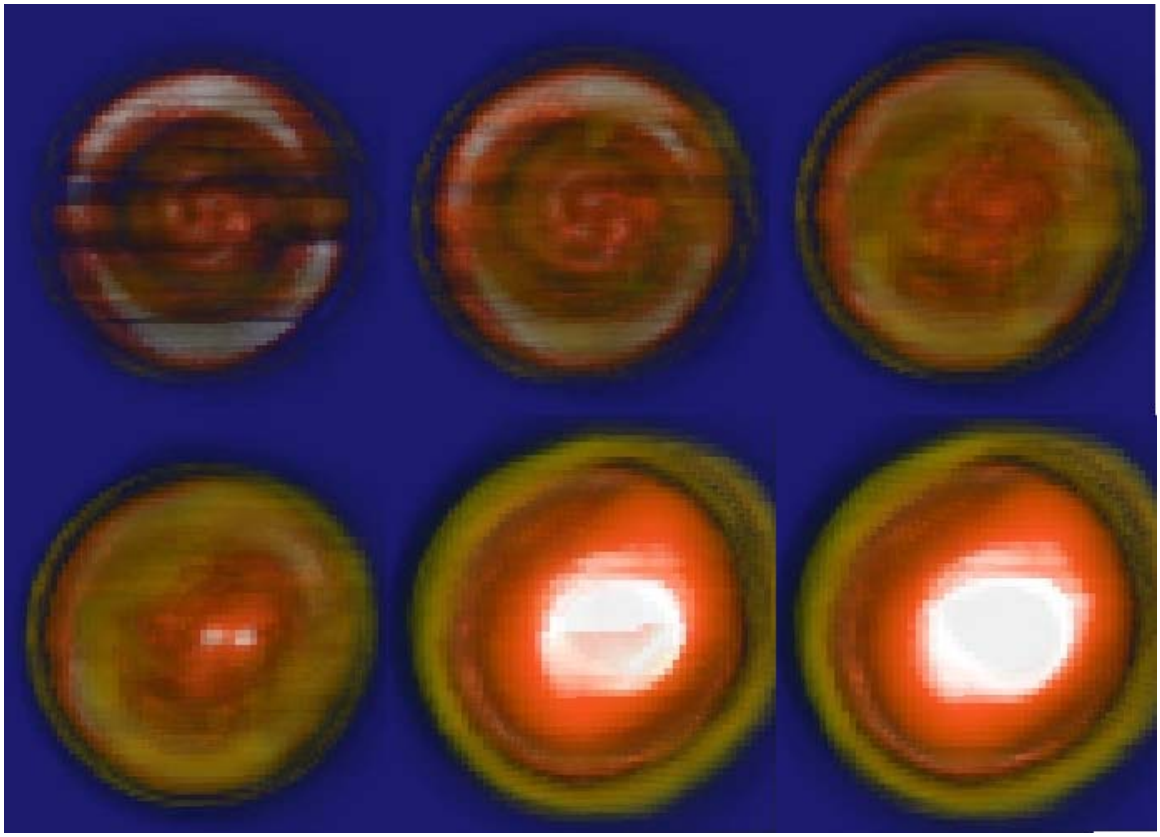

Figure 15. Visapult Renderings of Cactus's 3D Wave Equation Evolution

In the second row, the simulation has evolved even more. The middle and rightmost images on the bottom row show the effects of how an asynchronous receive/render strategy can produce artifacts. In the middle image of the bottom row, the center of the field has a region of white that is partially covered with a light orange, which terminates abruptly. This is a good example of having data from (at least) two time steps loaded into memory at once. After a few more updates, that artifact goes away, as seen in the bottom right image.

Keep in mind the visualization we were performing is of a dynamic, evolving simulation. The artifacts seen in the images we present here are short-lived in time. If a packet is lost, the resulting visualization has an artifact that can be characterized, in this case, as a "streak." Due to the nature of the fundamental algorithmic design, the resulting visualizations have artifacts caused by the presence of data from more than one timestep; it is difficult to ascertain whether or not any given visual artifact is cause by packet loss or by regular and normal software execution.

Mitigation of the type of artifacts we have described can occur on two fronts. First is careful rate-regulation of UDP flows. We perform such regulation as a general practice; the Cactus interface allows a user to specify the bit rate used to send packets from Cactus to Visapult. The second is to impose some notion of "frame boundaries" upon the Cactus/Visapult rendering algorithm. Doing so would ensure that data from only one timestep is used to perform visualization. The disadvantage of imposing such a restriction is that overall end-to-end performance will be reduced. For the types of problems being studied, physicists are able to quickly ascertain whether or not a simulation run is proceeding in "the right direction" by examination of the large structures seen in the resulting visualization. A few visual artifacts does not cause them concern for general questions like "are my simulation parameters grossly incorrect?" For situations requiring more careful visualization and analysis work, lossless data transmission is entirely appropriate.

Future Directions for High Performance Remote and Distributed Visualization

The work we have described in this chapter addresses only a small number of topics in high performance, remote and distributed visualization. Generally speaking, our work is similar in approach to related activities in distributed visualization: a collection of custom software components using a custom protocol on dedicated resources. The future of research in remote and distributed visualization must include sweeping changes in basic infrastructure. For instance, toolkits based upon "standard" techniques, should be employed to implement flow regulation underneath the application. These tools need access to standard mechanisms to obtain information about network-based resources, such as link bandwidth, computational and graphics capacity of a resource, and so forth. Without such standards, each new research project must "reinvent the wheel" but creating these components that form the basis for remote and distributed visualization applications. With such an infrastructure, a new world of opportunity opens for transition of research prototypes into the hands the visualization technology consumers.

The UDP method we have described so far relies on pacing of packets to meet, but not exceed, network capacity. Indeed, there is no reason that these fixed-data-rate methods cannot be applied to reliable transport protocols like TCP. Recent examples of fixed-rate reliable protocols include University of Illinois SABUL [Gu02] that is used to support data mining and more recent demonstrations of Indiana University's Tsunami [Meiss02] file transport program. In both cases, the protocols emphasize throughput for a stream-oriented protocol requiring considerable buffering. Such buffering is necessary in order to preserve the notion of the stream. Our group is proposing to separate the stream-orientation from the notion of reliability in a Reliable Independent Packet Protocol (RIPP). Such decoupling is critically important for supporting the needs of interactive visualization applications like Visapult.

As bandwidth-hungry fixed data rate methods like network video, SABUL and Tsunami become more pervasive, we must consider ways to provide global mediation of their flow rates. It is clear that fixed-rate implementations of both reliable and unreliable protocols can be disruptive to commodity networks[8], and are most appropriate for use on dedicated network links, Private Virtual Circuits (PVC's), Experimental Networks, or even for scheduled access. While it is unreasonable to assume that all WAN connections will be dedicated links, it is quite reasonable to target an architecture where high performance dedicated or schedulable links will exist between supercomputing centers and satellite data analysis centers on high-performance production and experimental network backbones. More dynamic shared environments require continuous adjustment of the data rates. In this context, there are some concerns that selecting an appropriate packet rate for UDP-based methods to minimize loss is too tedious to be practical.

[8] During our SC00 Bandwidth Challenge run, the commercial service provider of the link between NTON and Dallas asked us to please not exceed 1.5Gbps in transfer rates, lest we excessively interfere with commercial traffic. Our performance numbers during the SC00 run do not reflect any attempt on our part to limit bandwidth consumption.

Ideally, the network switching fabric should provide detailed QoS hints through informational packets to the endpoint hosts to indicate ideal send rates as per the MIT's XCP (eXplicit Congestion control Protocol) proposal [Katabi02], CSFQ (Core-Stateless Fair Queue)[Stoica98], or ECN (Explicit Congestion Notification) [Floyd94, RFC3168]. We could, for instance, have a TCP or UDP implementation that uses these hints to ignore packet loss if the switching fabric says that it is non-congestive, but defaults to the standard congestion avoidance algorithm when no such hints are available. In lieu of the wide availability of an intelligent network infrastructure, we are working on Grid-based bandwidth brokers that monitor current network conditions through SNMPv3 and provide feedback about network congestion through the Globus Metacomputing Directory Service (MDS) so as to avoid congestive loss. Even without intelligent switching fabric or explicit hints about congestion avoidance, we could create a system of peer-to-peer feedback/auto-negotiation by having end-points multicast their path and current packet-rate information on a fixed set of designated paths. This allows hosts to negotiate amongst themselves for appropriate packet rates rather than involving a third-party, like a bandwidth broker or the switching fabric itself. Ultimately, it is time to explore methods of coordinating fixed-data-rate flows as an alternative to current congestion-avoidance methods that attempt to infer congestion from packet loss statistics. The latter methods have clearly reached their scalability limit!

The primary area of growth in considering custom UDP protocols is in the development of fault-tolerant/fault-resilient encoding techniques. The simplest approach provides fault-tolerance by copying data from the previous time step to fill in lost data. A more advanced methodology could use a wavelet or frequency domain encoding of the data so that any loss is hidden in missing spatial frequencies (similar to JPEG compression). For transport of geometric models, we can look at packet encodings that support progressively refined meshes using triangle bisection [Hoppe96]. Such techniques make packet loss less visually distracting and eliminate the need for data retention on the sending side. Any reliable technique requires data to be retained at the source until its receipt is acknowledged. Given the large bandwidth-delay-products involved for future "terabit" networks, the window sizes necessary for reliable transport will be considerable. The buffering required to support TCP retransmission and large windows creates noticeable lag in the responsiveness of remote visualization applications and produces low bandwidth utilization rates. Fast response times are essential for creating the illusion of locality so low-latency connectionless techniques will be essential for Grid visualization and collaborative interfaces. Overall, there are many avenues to consider for information encoding that make performance enhancing, unreliable delivery methods offer graceful degradation of visual quality in response of packet loss rather than simply settling for degradation in interactivity.

As mainstream visualization systems start to move outside the confines of the desktop workstation, we must begin to consider issues of the emerging global computing infrastructure—the emerging "Grid." Like most current RDV applications to date, ours is a one-off prototype that has little chance of interoperating with components or network protocols that we haven't explicitly programmed it to understand. Such interoperability requires standards for interfaces and protocols through community consensus. There must be considerable work on basic architectural frameworks that enable seamless movement from the desktop to the grid and that support sharing of components across RDV component systems and services developed by different groups who may not be working on the same code base. Without such a framework, the entire community will be so mired in issues of Grid management, there will be little time left to spend on Visualization.

For instance, how does one actually go about the task of launching remote components in the emerging Grid infrastructure? In the case of our custom-built RDV applications, we typically use a manual process of "staging" the components on the machines we want to use for a distributed application. This approach is entirely impractical in Grid environments where there may be hundreds or thousands of resources that constitute a single application. There must be a notion of a database or directory service that can distribute and keeps track of components, both executables and running instances, on heterogeneous resources. Similarly, many contemporary distributed applications employ custom methods that signal one another to indicate state changes, and to move data. In order to make use of distributed resources, which could be dynamic in location and availability, visualization frameworks must make use of Grid services that automate the process of resource location and component launching. Fledgeling services of this form are

provided in Grids based upon the Globus[9] architecture in the form of Grid Information Services (GIS), which are essentially an LDAP-based hierarchical directory of resources for a given Grid. Such object directories and indexing infrastructure is necessary to support minimal automation for launching distributed components for a Grid-based visualization architecture.

As we consider how components are managed for distributed environments, one key observation about moving to a Grid architecture is that it isn't just about components. Grid based applications typically involve a fabric of components and persistent services woven together into complete applications. Services are loosely defined as persistent software components that are shared by more than one application or user. For example, an offscreen hardware renderer could be a component of a visualization that is time-shared by multiple users because there is only one of them available on the Grid.

Integrating the concept of services into RDV applications leads to hard problems in mediating access and security. Should you have an external "broker" to ensure fair usage of available visualization services in a Grid or should this be a function of the service itself? What do you do when the service fails to deliver its promised performance – a situation referred to as "contract violation"? Services must have an internal authorization and access control model in order to ensure that these access rules and behaviors are even minimally enforceable! Visualization people are not accustomed to dealing with these issues.

The issues described above are merely a starting point for attacking issues involved in hiding the incredible complexity of the Grid. Indeed, we are moving rapidly away from a computing model where you have complete and dedicated control of all of your computing resources (its all on the motherboard to your computer). The Grid is constantly changing in performance and capabilities, even as you are using it. Imagine what it would be like if while you were trying to run a visualization application on some platform, that the system administrator constantly removing and adding components, even while you are working. That approximates your "Grid" experience. The pieces are distributed over the wide area and your scope of control over them is extremely limited relative to your desktop. If we spend all of our time focusing on "stovepipe" visualization system designs, we will never be able to share any common infrastructure that is needed to tackle the problem of managing Grid resources! Now, more than ever, the visualization community must come together to work towards this common goal. One venue for discussing these topics is the Global Grid Forum (www.gridforum.org) where groups have the opportunity to discuss and ratify community standards that support compatibility between disparate RDV implementations and services. Participation by the visualization community in these forums is absolutely necessary to make wide area visualization and supercomputing possible.

Conclusion

Like many research projects, the scope of the Visapult effort has changed over time in response to new and unanticipated challenges. Visapult began as an effort to provide highly efficient and scalable software tools for data visualization to research scientists. In the relentless pursuit of ever-increasing levels of efficiency and performance, we learned a lot more about networking technology than would have been predicted at the outset. Our accomplishments in this project include winning the SC Bandwidth Challenge for three years in a row, achieving unheard-of levels of bandwidth performance for a single application. Despite these successes, Visapult is still an "island of capability" in the larger "ocean" of remote and distributed visualization. Much work in remote and distributed visualization is still needed before the research efforts of different programs can begin to work together as a coherent whole.

References

[Agarwal02] Agawal, Deborah**, "Self-Configuring Network Monitor Project: an Infrastructure for Passive Network Monitoring** Presented by Deb Agarwal at the CITRIS NorCal Network Research Meeting, Berkeley, CA, March 1, 2002.

[9] www.globus.org

[Bethel00] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau, "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization," in Proceedings of the IEEE/ACM 2000 Conference on Supercomputing (CDROM), November 04-10, 2000, Dallas, Texas, USA.

[Bethel01] W. Bethel, R. Frank, J. D. Brederson. "Combining a Multithreaded Scene Graph System with a Tiled Display Environment." In the Proceedings of the 2002 IS&T/SPIE Conference on Electronic Imaging and Technology, The Engineering Reality of Virtual Reality, San Jose, CA, January 2002.

[DMF] http://www-didc.lbl.gov/DMF/.

[Dunigan02] Dunigan, T. Mathis, M., Tierney, B. ""A TCP TuningDaemon", Proceedings of IEEE Supercomputing 2002 Conference, Nov 2002, LBNL-51022.

[Floyd94] Floyd, S., TCP and Explicit Congestion Notification (compressed postscript, pdf). *ACM Computer Communication Review*, V. 24 N. 5, October 1994, p. 10-23.

[Gu2002] Gu, Y., Hong, X., Mazzucco,M. Grossman, R. "SABUL: A high performance data transfer protocol" Submitted for publication, 2002.

[Hoppe96] Hoppe, H. "Progressive Meshes," *Proc. 23rd Int'l. Conf. on Computer Graphics and Interactive Techniques SIGGRAPH '96*, ACM, New York, NY, 1996, pp. 99-108.

[Katabi02] Katabi, D., Handley, M., "Congestion Control for High Bandwidth-Delay Product Networks," Proceedings of ACM Sigcomm 2002. (http://www.ana.lcs.mit.edu/dina/XCP/)

[Madhavi97] Madhavi, J., Floyd, S. "TCP-Friendly UDP Rate-Based Flow Control" Technical Note, Jan 8, 1997. (http://www.psc.edu/networking/papers/tcp_friendly.html)

[Meiss02] http://www.anml.iu.edu/anmlresearch.html

[Müller99] K. Müller, N. Shareef, J. Huang, R. Crawfis, "IBR-Assisted Volume Rendering." In Late Breaking Hot Topics Proceedings of IEEE Visualization 99, IEEE Press, October 27-29, 1999, San Francisco, CA, USA. pp 5-8.

[Shalf03] J. Shalf and E. Wes. Bethel, "Cactus and Visapult: An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols." To appear in the IEEE CG&A Mar/April 2003 special issue on Grid Visualization.

[Stoica98] I.Stoica, H.Zhang and S. Shenker , *"Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks" proceedings of ACM SIGCOMM 98.*

[Tierney98] Tierney, B., W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter.The NetLogger Methodology for High Performance Distributed Systems Performance Analysis, Proceeding of IEEE High Performance Distributed Computing conference (HPDC-7), July 1998, LBNL-42611

[Tierney99] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., "A Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceedings of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999

[VonLaszewski02]Von Laszewski, G. **,** Russell, M., Allen, G. , Daues, G., Foster, I., Seidel, E., Novotny, J. ,Shalf, J.**, "The Community Software Development with the Astrophysics Simulation Collaboratory"** Concurrency in Computation: Practice and Experience, Spring 2002.