Cactus and Visapult:
A Case Study of Ultra-High Performance Distributed Visualization Using Connectionless Protocols

John Shalf[†] and E. Wes Bethel
Lawrence Berkeley National Laboratory
National Energy Research Scientific Computing Center
University of California
Berkeley, CA 94720

Abstract
This past decade has seen rapid growth in the size, resolution, and complexity of Grand Challenge simulation codes. Many such problems still require interactive visualization tools to make sense of multi-terabyte data stores. Visapult is a parallel volume rendering tool that employs distributed components, latency tolerant visualization and graphics algorithms, along with high performance network I/O in order to achieve effective remote visualization of massive datasets. In this paper we discuss improvements to network bandwidth utilization of the Visapult application that result from using connectionless protocols to move data payload between the distributed Visapult components and a Cactus-enabled, high-performance physics simulation used to study gravitational waveforms of colliding black holes. These improvements have boosted Visapult's network efficiency to 88-98% of the maximum theoretical available bandwidth on multi-gigabit Wide Area Networks.

Introduction
Over the past decade, growth in the memory and data storage capabilities of the largest supercomputing installations in the world has outpaced Moore's law. This has lead to a significant data management and data analysis problem. Even more dramatic growth is expected in the observational sciences as high-resolution data feeds from remotely-operated network/Grid-connected observatories and experimental equipment come online [21]. While statistical methods and feature detection/extraction algorithms have been proposed to automate the mining of useful information from these enormous data stores, there is still a strong need for a human component to the visualization process. It is not realistic to expect that every numerical simulation domain will have suitable automated methods to mine and detect features in large data sets. Detection of unexpected phenomena is still most often performed using interactive visualization.

Consequently, supercomputing centers, such as the National Energy Research Supercomputing Center (NERSC), are motivated to provide visualization tools that enable effective interactive analysis of remotely located data stores and remote monitoring of simulation codes runs that can span multiple days, or in some cases, weeks. Such tools must balance the competing interests of interactivity and fidelity in order to fit within the limits imposed by available infrastructure. A naive example of this tradeoff is downsizing large datasets in order to reduce the amount of processing that needs to be done by the visualization application, or to reduce the bandwidth requirements of moving data to a remote location.

The NERSC/LBNL Visualization group has developed the Visapult [1] tool to attack these sorts of "Grand Challenge" problems. Visapult is a distributed, parallel volume rendering application that allows us to use parallel computation and high performance networking resources that are on the same scale as the supercomputers that generated the data. Volume rendering is a very important visualization technique for exploration of complex 3D datasets, but is extremely compute- and bandwidth-intensive. Traditional serial raytracing volume renderers can take many minutes or hours to render a single frame. Visapult supports volume rendering at interactive rates by employing network distributed components, a high degree of parallelism, and a rendering technique called Image Based Rendering Assisted Volume Rendering (IBRAVR) that was originally described in a paper by Mueller & Crawfis in 1999 [7]. The IBRAVR algorithm, described more completely in other papers [1,2,7], allows Visapult to trade additional

[†] Authors' contact information: jshalf@lbl.gov and ewbethel@lbl.gov, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Mail Stop 50F, Berkeley CA, 94720.

computational power in exchange for reduced bandwidth requirements. In this paper, we will describe the high performance remote visualization tool called Visapult and our efforts to improve its effectiveness using aggressive network tuning and network protocol modifications. In particular we will describe how we used a new connectionless UDP protocol to improve network utilization efficiency from 25% of line-rate to 88% of line-rate for multiple gigabit networks.

Driving Application (General Relativity)

A motivating application for this kind of remote visualization capability is the modeling of black hole collisions. One of the most challenging problems in science is the numerical simulation of Einstein's equations in order to explore Einstein's Theory of General Relativity (GR). These equations are among the most complex in the world of physics; a set of nonlinear, hyperbolic, elliptic coupled equations containing millions of terms when fully expanded. The General Relativity Group at the Albert Einstein Institute in Potsdam Germany developed the Cactus code [3,4] to solve these equations from first principles on supercomputers in order to simulate astrophysical phenomena with extremely high gravitational fluxes, such as the collision of two black holes and the gravitational waves that radiate from that event.

The Cactus simulation code scales well on some of the largest supercomputers in the world, including NERSC's IBM SP2 (seaborg). Large-scale MPP's like seaborg are required to reach the target regime of physics they would like to explore; the "inspiraling" merger of two black holes. This is the very sort of event that gravitational wave observatories such as LIGO and VIRGO are designed to detect. These observatories can barely discern real gravitational wave signals from background noise, so the gravitational wave signatures derived from simulations of these events are a critical means to improving the Signal to Noise Ratio of these devices. Simulations that require comparatively small amounts of memory (~1Tbyte) allow accurate simulations for simple phenomena like the Schwartzchild (spinning black hole) and Misner (head-on-collision of two black holes) cases (*see figure 1)*. While simulating of these cases is important for validating and calibrating the solvers used by the code, they are of little consequence to the endeavor of gravitational wave characterization and detection. At a minimum, 1.5TB of memory is required to simulate a basic inspiraling collision using bitant symmetry (reflection along the plane of rotation). A full 3D evolution requires a minimum of 3-5TB of RAM. Still more memory is required to move the outer boundaries far enough away from the event to perform accurate gravitational wave extractions necessary to assist in the signal processing performed on LIGO data.

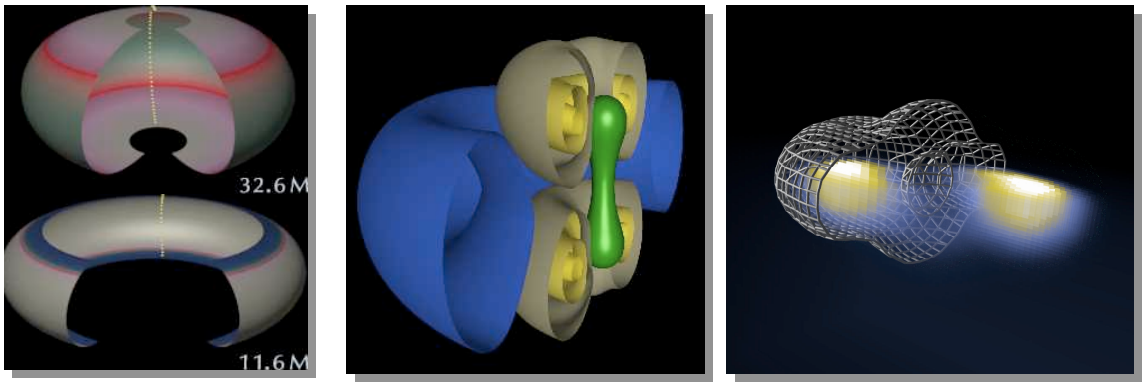

Figure 1*: Images of black hole simulations showing the Schwartzchild, Misner, and inspiraling merger cases respectively. The first image shows the event horizon of the Schwartzchild spinning black hole colorized by its extrinsic curvature. The Misner visualization in the center shows the real component of the gravitational potential (psi4 of the Weyl tensor) as blue, grey, and yellow isosurfaces (cut open so you can see inside) and the event horizon of the merging black holes as a green surface. The last image is a volume rendering of the lapse with a cut-open wireframe isosurface of the real part of the gravitational waves emanating from an offset inspiraling merger of two black holes.*

These simulations are so large that it is impossible to use traditional visualization tools to see and understand their results. Since gravitational physicists do not know, a-priori, the nature of the gravity wave

signals they are studying, data mining/feature extraction techniques are inappropriate for this activity. It is not feasible to download all of the data from the supercomputer center to a remote site for later analysis. Due to the large size of the computed data sets, the platform needed to analyze the data must be similar in scale to that used to produce the data, unless significant data reduction or non-interactive out-of-core methods are employed. Typically, visualization is performed at a particular supercomputer in-situ with the running simulation, with the results made available to a remote viewer located, possibly, halfway around the world. Such remote monitoring improves the effective use of supercomputers by allowing the Cactus users to cut short runs that have gone awry, or to steer the code or restart it with more appropriate parameters or initial conditions. The interactive, remote visualization and monitoring methodology requires latency-tolerant visualization algorithms with an emphasis on maximizing Wide Area Network (WAN) throughput.

Visapult and the Latency-Tolerant Direct Volume Rendering Algorithm

As described in earlier papers [1,2], Visapult is composed of three components: a raw data source, a viewer, and an MPI-based back end (*see figure 2*). The back end first reads data from the raw data source, which may be either local or remote. Next, the back end domain decomposes the volume of data into smaller units, each of which is direct volume rendered to produce a single image of that particular sub-domain. Relatively little communication is required between the back end processes, so the system scales very well to large numbers of processors. Each of these images is then transmitted to a viewer, where they are texture mapped onto static geometry, and rendered using OpenGL. This process takes advantage of both the computing horsepower of MPP's for partial rendering as well as the benefits of hardware-accelerated rendering available on many low-cost client workstations. It also decouples interactivity on the desktop from the latencies involved in moving data over the network. Our implementation of this technique using large MPPs, high performance networks and desktop viewer is a high-performance extension to the IBRAVR algorithm described in previous literature [7], and was presented in previous publications [2].

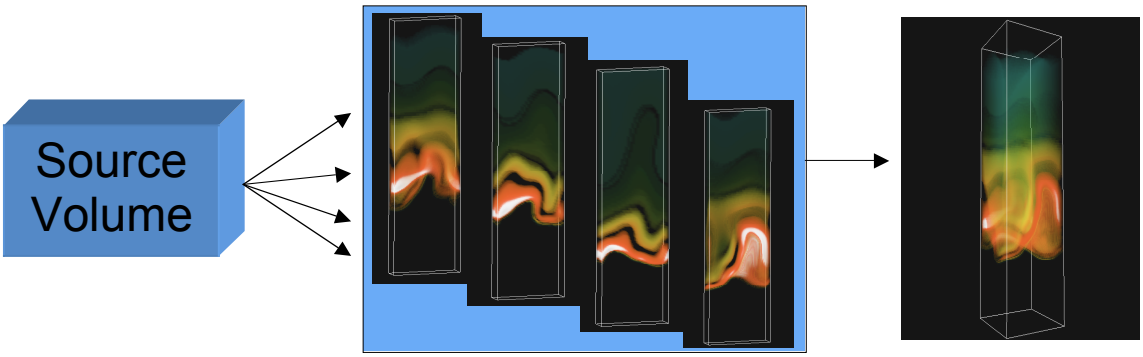

Figure 2*: Diagram of the Visapult processing pipeline. The data source (either a running Cactus simulation or DPSS), feeds the Visapult back-end using multiple parallel network data streams. Each process of the Visapult back-end volume renders its portion of the spatially decomposed domain into an image. These images are in turn sent to a client application running on the user's workstation that uses 3D hardware-accelerated graphics/textures to composite these images into a single scene at interactive rates.*

TCP Performance on the WAN

The various components of Visapult must communicate with one-another using IP connections over a LAN or WAN, depending on the deployment of the components. I/O between data source and back-end is performed asynchronously so that it overlaps the volume rendering computations (*see figure 3*). Despite this, the effective performance of the application when operating with maximum fidelity is network-I/O bound unless the network communications are extremely efficient. Visapult has parameters that allow us to reduce I/O utilization in order to fit within the network-imposed limits, but this requires a commensurate trade-off in visual fidelity. Therefore, improvements in the effectiveness of network I/O are critical to maximizing application performance.

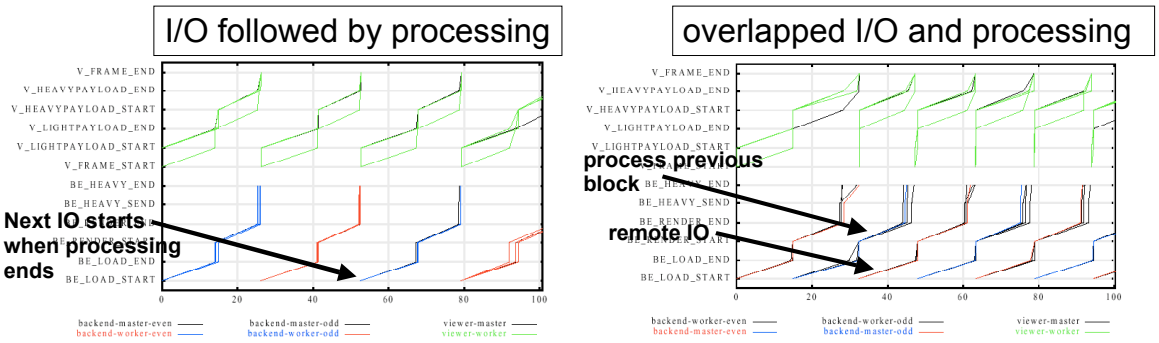

***Figure 3**: By placing network I/O into a background thread, the visapult volume rendering computations could be overlapped with the data transfers, thereby doubling the duty cycle of the application. This overlap fails if network I/O takes longer than the volume rendering calculation, so optimizing network performance is critical to improving overall application efficiency.*

Single-stream TCP performance on the WAN is often disappointing. Even with aggressive tuning of the TCP window size, buffer sizes, and chunking of transfers, typical performance is still a fraction of the available bandwidth on the WAN on OC-12 or faster links. While there are a number of factors involved, the behavior of the TCP congestion avoidance algorithm has been implicated as a leading cause of this performance deficit.

TCP congestion avoidance algorithms assume that any packet loss is due to congestion, which we define to be oversubscription of bandwidth on any switch or link along the path the packet stream takes on the WAN. However, it is increasingly the case that packet loss is caused by events that are unrelated to congestion. The sensitivity of TCP to loss is further exacerbated as the bandwidth of the network is increased, so solutions to remedy poor TCP performance will be increasingly important to distributed computing applications on the WAN. Simulations of the TCP protocol performed by Jacobson and Floyd show a high sensitivity to loss, but also demonstrate the fact that from a control theory standpoint, that the TCP congestion avoidance algorithm results in periodic fluctuations that resonate with the deterministic control mechanisms of switching fabric in a highly non-linear and unstable fashion [5]. The default "tail-dropping" behavior of packet switch input-queues leads to significant degradation in performance. One recommended cure for this deficiency is traffic shaping using Active Queue Management (AQM) schemes like Random Early Detection (RED), but a study by Hollot et al. shows that the parameters for RED must be chosen carefully in order to provide both stability and throughput. Deployment of AQM will require considerable modification to our network infrastructure and cannot be affected at an application level. These non-congestive losses are further explored in other papers [5,6,8,9,11]. It is our conclusion that TCP congestion control, while adequate for the 10megabit networks it was originally designed for, is failing completely on today's multiple gigabit networks and multiple efforts are underway to work around these problems. Loss-tolerant UDP-based protocols will play an increasingly important role in high throughput network applications of the near future.

TCP Multistreaming

The typical solution to work around TCP congestion avoidance is to use multiple simultaneous TCP connections. This is actually a very old solution that is most commonly employed in web-browsers dating back to xmosaic[12]. This technique works because streams don't share packet loss/window-size information with one-another, so a lost packet will only cause one of the streams to back off rather than all of them. The effect is to make the hypersensitive TCP congestion avoidance algorithm artificially less sensitive to frame loss, but this undoes some of the "fairness" implicit in the congestion avoidance algorithm. In effect, a multistreaming application becomes a network "bully" that competes rather unfairly with its peers. This kind of protocol arrangement is typically implemented at the application level in an ad-hoc application-level manner [15] but GridFTP [10] formalizes this methodology into a standard protocol.

TCP Implementation of Visapult

This multi-streaming method was used for the first implementation of Visapult, developed under the NGI program for the Combustion Corridor project. The Distributed Parallel Storage System (DPSS), developed by LBL's Distributed Computing Group, provides a data source that improved multi-streamed TCP performance by striping the network streams was across multiple hosts and network adaptors [13]. The TCP window sizes, buffer sizes, and blocking parameters were carefully tuned to maximize performance. Visapult network I/O receives occur in the background, while Visapult is simultaneously volume rendering its current data payload. In order to achieve complete overlap of computation and I/O, the network performance must be extremely high; hence the aggressive use of TCP multi-streaming. At SC00 in Dallas Texas, we were able to achieve peak throughput rates of 1.5Gigabits/sec on an OC48, with a sustained rate of about 660Mbps over a 60-minute window (*see Figure 4*). While this was sufficient to win the SC 2000 Bandwidth Challenge with nearly twice the performance of the nearest competitor, this was still only 60% (peak) and 25% (sustained) of the theoretical line rate, respectively, of the OC-48 link used during the contest. On a shared WAN, >50% utilization is still admirable, but these were dedicated links that were entirely uncongested. Furthermore, the aggressive tuning of the TCP parameters indicates that we cannot idly blame the "wizard gap" [14] for this performance deficit. So even with tuning and a dedicated link, we cannot fully exploit the link bandwidth using the TCP protocol.

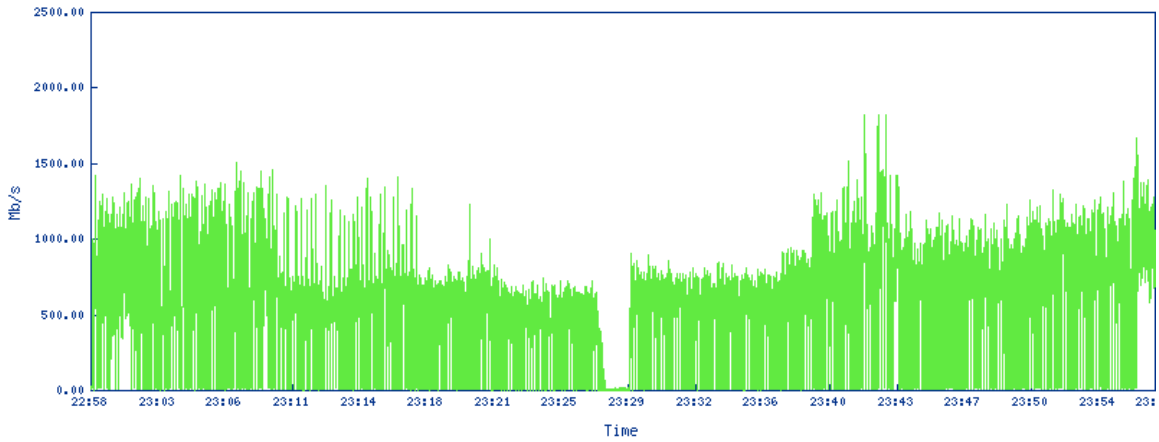

Figure 4*: Graph of network throughput for Visapult during the SC2000 Bandwidth Challenge. The performance was extremely erratic over a sixty minute time span, despite a nominally dedicated network connection and use of 32 parallel TCP streams to mitigate the effects of packet loss. This points to the extreme sensitivity and instability of TCP congestion control in practice.*

Moving to Unreliable Protocols

While testing WAN links, we have consistently noticed that UDP "packet blasting" tests show frame loss rates that are so small that our testing tool, *iperf*, was often unable to represent them in its standard printed floating point format. However, these same losses were still sufficient to make the TCP-based *iperf* tests slow to a fraction of maximum line rate. With custom tools, we find that UDP packet loss rates are consistently low until you reach a critical limit, which is the available bandwidth of the slowest/most congested link in the network path. At the critical point, when the slowest link in the path across the WAN becomes congested, the loss rate climbs almost linearly in proportion to the increase in output rate. It is interesting to note that frame loss rates, even at low data rates, exhibit some background loss. This is counterintuitive, as switches should be less congested when exposed to the slower stream, based on the models of network loss assumed by TCP congestion avoidance algorithm.

For data transfer and replication, file integrity is paramount. Response time and performance is of comparable importance to file integrity for visualization applications. Visualization tools almost invariably use reliable transport protocols to connect distributed components, since there is a general concern that lifting the guarantee of data integrity would compromise the effectiveness of the data analysis. However, visualization researchers find acceptable other forms of lossy data compression like JPEG, wavelet compression and even data resampling. Acceptance may be due to the fact that degradation in visual quality

is well behaved in these cases. Therefore, an unreliable transport mechanism that deals with packet loss gracefully and doesn't exhibit extreme visual artifacts will compete well with other well-accepted data reduction techniques. Furthermore, when tuned to fit within the available bandwidth of a dedicated network connection, the loss rates for unreliable transport are extremely small - a few tenths of a percent of all packets sent if the packets are paced to stay within the limits of the slowest link in the network path.

Visapult Implementation Using Unreliable Transport

For the SC01 Bandwidth Challenge, we modified the back end of Visapult to work with our own custom UDP transport protocol. Each packet contains encoded information about where to place its data payload in the destination array (*see figure 5*). This ensures that each packet can be treated independently so that packet ordering and packet loss have minimal effect on destination processing. The Visapult back end was modified to render continuously rather than waiting for all packets in a given frame to arrive. The visual effect of this choice is a progressive refinement of the image over time. Data from the previous frame was used to "prime" the receive buffer and fill in gaps where packets were lost. We contemplated a multi-buffering scheme to allow all possible packets to arrive before rendering. In practice, such a technique would provide improvements in visual quality when the data in successive frames is changes rapidly. However, due to the slow rate of evolution of the data for our remote monitoring application, the single-buffered approach proved sufficient. This sort of multi-buffering continues to be an area of investigation.

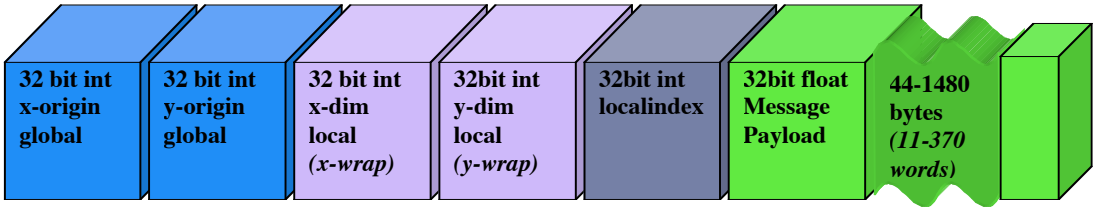

Figure 5*: Diagram of the UDP packet format. A 20-byte header was used to locate the payload of each packet in the destination domain completely independently of one-another. Assuming a full 3D block domain decomposition, the origin and dimension information is superfluous in the z-dimension.* ***X/Y origin global*** *=> the offset in grid coordinates with respect to the global data dimensions of the origin of the domain decomposed block (local domain) in the data source.* ***X/Y dim local*** *=> The x/y dimensions of the local domain decomposed chunk in the data source (used to control wrap-around of data as it is written into the destination data array).* ***Localindex*** *=> The offset in cells counted from the start of the local domain decomposed block in the data source. This source-based data indexing allows the component sending the data to be ignorant of the domain decomposition at the destination, but provides enough information at the destination to support unambiguous data re-assembly.*

Rather than using static data stored on the DPSS as the data source as in our SC00 implementation, we chose to connect Visapult directly to the Cactus code. Cactus' modular code components are referred to as "thorns". We developed a custom "thorn" for Cactus that sends data to the Visapult back end to support visual remote monitoring of executing codes. Combined with the Cactus web-based remote steering interface, Visapult can be used as a visualization component of an interactive remote steering system [3]. The Cactus/Visapult thorn (AlphaThorns/ShmServ) buffers data in a shared memory, staging data that is created when the code starts up. A set of background worker processes (NetWorkers) are spawned at startup time that are dedicated to reading data out of the shared memory region and sending it over the network to the Visapult back end. We chose to implement the NetWorkers as processes rather than threads in order to get around the performance problems and scheduling overhead associated with some particularly poor vendor implementations of pthreads. The NetWorkers require a dedicated CPU so that they don't interfere with scheduling and execution of the primary simulation code's processes. The NetWorkers have parameters that allow the code set a fixed packet rate that can be tuned to prevent oversubscription of resources and thereby minimize packet loss. At least one asynchronous NetWorker is required per network interface card (NIC), but we also found that multiple NetWorkers per NIC were required to maximize GigE NIC utilization on larger SMP's like the 16-way IBM Nighthawk SP nodes.

On dual-CPU x86 Linux hosts with GigE NICs, we were able to send at approximately 560Megabits/sec using this method. Linux performance can be improved to 970Megabits/sec per NIC using the VETH/Yumo patch for Syskonnect Linux Kernel drivers, which enables aggressive interrupt coalescing (intended for link aggregation, but improves single-NIC performance as well) [24]. This is the same rate that we were able to achieve with 'iperf –u' on these hosts. The SP2 is configured with a single node with 4 GigE NICS operating as a "gateway" to system IP traffic. The IP traffic from interior nodes must be channeled through the internal network to the gateway. We could only reach approximately 300Megabits/sec when channeling IP traffic through the SP's internal communication network. The cause of this deficiency is that the 1500byte IP packets made inefficient use of the SP's internal 64Kbyte packet format. Higher aggregate rates could be achieved using multiple source and destination addresses because the SP load-balances packet streams across the 4 interfaces by XOR'ing the 2 LSB's of the source and destination addresses, but it was still insufficient for this application. Furthermore, there was insufficient internal bandwidth to drive the 4-way striped Gigabit Ethernet adaptors from a single node and some packet-blasting experiments had the ability to disrupt many important internal components of the SP2 including the GPFS shared filesystem, the batch scheduler, and even the ability to login to individual nodes to kill the errant packet-blasting process that was causing the difficulties. By moving to dedicated GigE NICs placed on each node, we were able to reach 520Megabits/sec per node with a single NetWorker and a peak of 760Megabits/sec per node when employing 4 or more NetWorker processes.

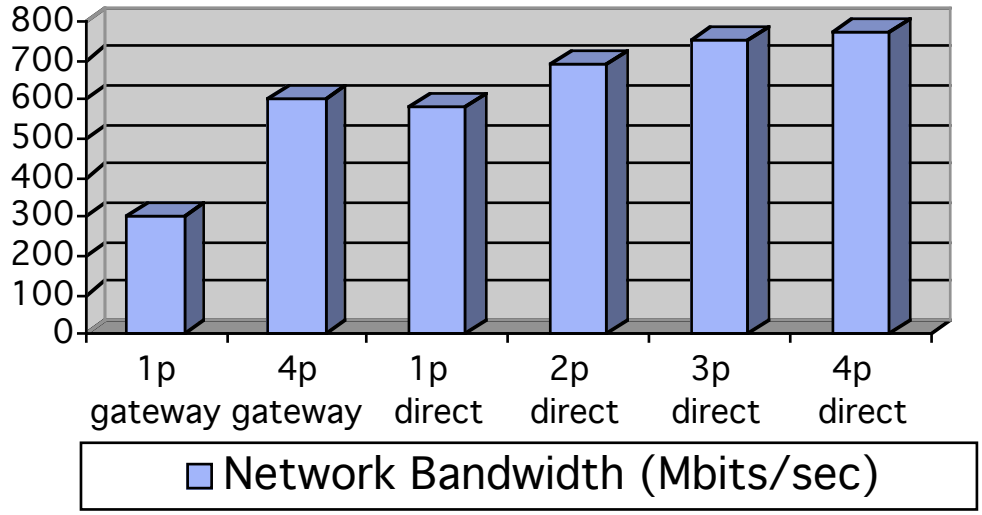

Figure 6: UDP performance on the SP2 using the internal network (sending through GigE on the Gateway node (1p-gatway and 4p-gateway mean 1processor on one node and 4 processes on 4 nodes routed throught the gateway node respectively) and 1-4 NetWorker processes (1p-4p direct) on 16 CPU Nighthawk nodes with directly-attached GigE NICS. There is considerable benefit in using directly attached GigE NICs and some moderate benefit to feeding the network interface using multiple processes on each node.

SC01 Bandwidth Challenge Results

This past year at SC01 in Denver, we conducted another high performance Visapult run as part of the Bandwidth Challenge. This run made use of the new UDP transport protocol between the data source and the back end of the Visapult application. For the data source, we ran a binary black-hole merger calculation using the Cactus code on 6 nodes of NERSC's IBM SP-2 system. We also ran a related apparent horizon finder on a 128node Origin 2000 system at NCSA. At NERSC, the GigE interfaces on 5 of the SP-2 nodes connected to a dedicated OC-48 link provided by Qwest and the remaining NIC connected to an OC-12 link provided by ESNet. The Visapult application itself ran on an 8-node Linux

cluster on the show floor that was connected to a 10Gigabit Force10 switch. At NCSA, a single GigE on the host fed the NCSA OC-12 uplink and fed a 32way Sun Starfire system in the Sun booth on the showfloor.

The NERSC system alone was able to reach throughput rates of 2.38Gbps on the OC-48. That is 96% of its theoretical capacity and reached this rate without the typical ramp-up associated with TCP-based applications (*see figure 6*). The total aggregate throughput of the application was 3.3Gigabits; once again winning the Bandwidth Challenge. The visual quality and interactivity of the application was greatly improved by the high average throughput sustained by the application's data feed. The visual quality of the volume visualization was not compromised by the unreliable transport method (*see figure 7*). The gains in interactive performance were very dramatic. While there were many lost packets, the loss rate was easily tailored to be less than 1% by throttling the packet-sending rate. The resulting artifacts when packets were lost were rather innocuous, but obvious to users who might study the image for them, however, the extremely high bandwidth efficiency resulted in rapid recovery from these artifacts when they did indeed occur. Overall, the move to unreliable transport greatly enhanced the effectiveness of the application for remote analysis of extremely large/dynamic datasets.

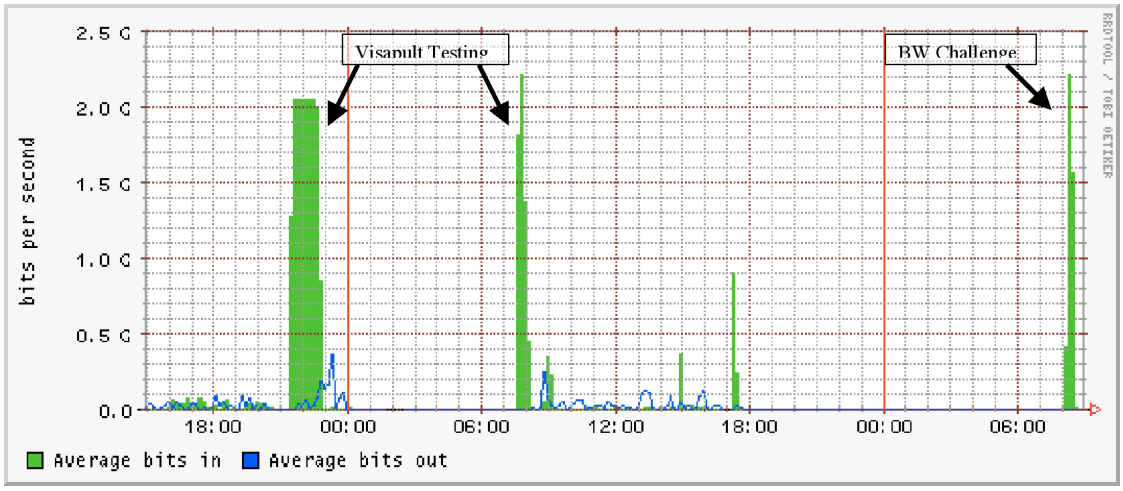

Figure 7: *The daily network traffic log from SciNET Core1 router during the Bandwidth Challenge that monitored the OC-48 traffic from NERSC. All three network traffic peaks that exceeded 2.0 Gigabits are from Visapult testing (the last being the actual bandwidth challenge run). This traffic, combined with another OC-12 from NERSC and an OC-12 from NCSA via Abilene resulted in a sustained performance exceeding 3 gigabits.*

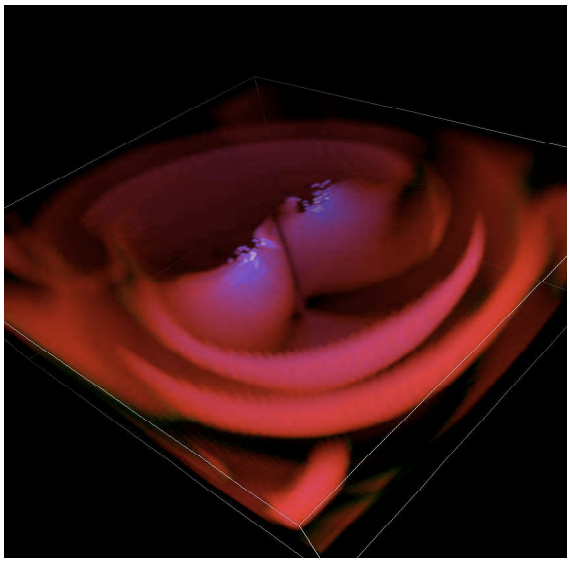

Figure 8:*Volume rendering of the real component of gravitational potential (psi4 of the Weyl tensor).*

Discussion / Future Directions

The UDP method we have described so far relies on pacing of packets to meet, but not exceed, network capacity. The are some concerns that selecting an appropriate packet rate for UDP-based methods is too tedious to be practical, but this ignores the fact that the same tuning must occur even for multi-stream TCP implementations. Using too many TCP streams can quickly lead to the same congestion situations that occur with UDP [18] Even with UDP, we can incorporate TCP-friendly methods described by Mahdavi & Floyd [17] to provide guidelines for appropriate packet rates that limit impact on other network users.

There is no reason that these methods cannot be applied to reliable transport protocols like TCP as well. We refer to TCP/reliable transport without congestion control as "TCP Brooklyn" (a TCP implementation that doesn't back-off in the face of packet loss). It is clear that fixed-rate implementations of both reliable and unreliable protocols would be disruptive to commodity networks, and are most appropriate for use on dedicated network links, Private Virtual Circuits (PVC's), Experimental Networks, or even for scheduled access. Such dedicated bandwidth is more typical in high performance networks for the sciences like ESNet, Abilene, and CANARIE. We believe that Grid-based bandwidth brokers such as those proposed for the Quanta QoS project [20] can be used to expand fixed-rate transport services to a wider variety of configurations; posting QoS feedback on an MDS (Metacomputing Directory Service)[19].

More dynamic environments require continuous adjustment of the data rates. Multi-streaming TCP techniques offer no advantage over UDP protocols in this regard as their performance advantage comes from responding slowly to packet loss; even in cases of actual congestion. We argue that it is possible to craft a congestion control algorithm that models the behavior of multi-stream TCP using a only single stream by simply modifying its response to loss or, as in the case of Web100 Work-Around-Daemon (WAD), uses artificially large virtual-MTU's to cause it to recover more quickly from packet loss [22] Ideally, we would like to see our network switching fabric provide detailed QoS hints through informational packets to the endpoint hosts to indicate ideal send rates. 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 in lieu of hints. Even without intelligent switching fabric, 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.

The primary area of growth in considering custom UDP protocols is in the development of fault-tolerant/fault-resilient encoding techniques. The technique described here provides very naïve fault-

tolerance by copying data from the previous time step to fill in lost data. A more advanced methodology, could for example, 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 [23]. Such techniques make packet loss less visually distracting and eliminate the need for data retention on the sending side. Any reliable technique requires that data be retained until its receipt at the destination is acknowledged. Given the large bandwidth-delay-products involved for future "terabit" networks, the window sizes necessary for reliable transport will be considerable. 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.

Conclusions

The movement to well-behaved fault-tolerant UDP-based protocols is a significant area of exploration for the future development of effective remote visualization tools. Such aggressive methods are necessary to overcome the limitations of the aging TCP protocol for high throughput applications on high speed Wide Area Networks. The technique we have described has been used to obtain better than 300% improvement in I/O efficiency over the best available tuned, multi-streaming TCP methods. While packet delivery is not guaranteed, the results are comparable to other lossy data reduction techniques commonly employed in visualization. Ultimately, visualization researchers must re-evaluate exclusive reliance on TCP-based reliable transport for distributed visualization applications because it is greatly impeding our ability to exploit networking resources.

Acknowledgements

This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The Cactus Team: Especially Ed Seidel, Gabrielle Allen, and Peter Diener; Chip Smith for setting up the cluster. Force10 Networks for making such an awesome network switch and Raju Shah for making it work. Mike Bennett and John Christman of LBLNet for finding and testing this switch. John Towns and Wayne-Louis Hoyenga at NCSA-University of Illinois for use of their O2k array. ESNet and Qwest for setting up the OC-48 for the bandwidth challenge. SciNET and Eli Dart who worked tirelessly to rewire the NOC (multiple times) over the course of Supercomputing 2001. And finally to NERSC for use of their enormous SP2 supercomputer and David Paul for helping fix the system to do our bidding.

Bibliography

[1] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau, Visapult - A Prototype Remote and Distributed Visualization Application and Framework. (LBNL-45215) In Siggraph 2000, Applications and Sketches, New Orleans, LA.

[2] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau.,"Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization" (LBNL-45365). In "Proceedings of SC00", November 2000.

[3] Gabrielle Allen, Werner Benger, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, André Merzky, Thomas Radke, Edward Seidel, John Shalf, "Cactus Tools for Grid Applications", Cluster Computing **4**, 179-188, 2001. (http://www.cactuscode.org/Showcase/Publications.html)

[4] Cactus Information at http://www.cactuscode.org

[5] S. Floyd, V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways", Computer Communications Review, V.21, N.2, April 1991.

[6] C.V. Hollot, V. Misra, D. Townsley, W-B Gong, "A Control Theoretic Analysis of RED", to appear in Proceedings of IEEE Infocom, 2001.

[7] K. Meuller, N. Shareef, J. Huang, and R. Crafis, “IBR-Assisted Volume Rendering,” Proceedings of IEEE Visualization 1999, Late Breaking Hot Topics, October 1999.

[8] J. Stone, C. Partridge, “When the CRC and TCP Checksums Disagree,” Proceedings of ACM SigComm 2000, Sept 2000

[9] R. Seifert, “Gigabit Ethernet,” Addison Wesley Longman, Inc. 1998.

[10] GridFTP information: http://www.globus.org/datagrid/gridftp.html

[11] D.M. Chiu, R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” Computer Networks and ISDM Systems, Vol. 17, 1989.

[12] xmosaic web browser information: http://archive.ncsa.uiuc.edu/SDG/Software/XMosaic/

[13] B. Tierney, J. Lee, B Crowley, M. Holding, J. Hylton, F.Drake,” "A Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceedings of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896

[14] M. Mathis,"Pushing Up Performance for Everyone", NLANR Workshop Talk, 1999. (http://ncne.nlanr.net/training/techs/1999/991205/Talks/mathis_991205_TCP_Auto-tuning/)

[15] H. Sivakumar, S. Bailey, R.L. Grossman,” Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks,” Proceedings of Supercomputing 2000, 2000.

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

[18] T. J. Hacker and B. D. Athey and B. D. Noble.□“The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network”.□To appear in the *Proceedings of the 16th International Parallel & Distributed Processing Symposium,* April, 2002, Fort Lauderdale, FL. (http://mobility.eecs.umich.edu/papers/ipdps02.pdf)

[19] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman,” Grid Information Services for Distributed Resource Sharing. “ *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.Globus MDS Information: http://www.globus.org/mds

[20] Quanta Testbed: QoS on high performance optical networks: http://www.evl.uic.edu/cavern/teranode/quanta.html

[21] T. DeFanti, M. Brown editors, “Report to the National Science Foundation Directorate for Computer and Information Science and Engineering (CISE) Advanced Networks Infrastructure & Research Division,” Technical Report, Chicago, Dec 2001. Text of the final report available at http://www.evl.uic.edu/activity/NSF/final.html

[22] Web100 Concept Paper: http://www.web100.org/docs/concept_paper.php

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

[24] VETH link aggregation patch at: http://www.st.rim.or.jp/~yumo/#veth
Whitepaper on VETH/LACP performance at http://dit.lbl.gov/Tmp/iperf_3.pdf