Grid-Distributed Visualizations Using Connectionless Protocols

E. Wes Bethel and John Shalf
Lawrence Berkeley National Laboratory

We discuss the advantages of using connectionless protocols to greatly accelerate network throughput of grid-based distributed applications through our experience with Visapult and Cactus.

Over the past decade, growth in memory and data storage capabilities has outpaced Moore's law. This has led to a significant data management and analysis problem. We can expect more dramatic growth of the quantity of available data in the observational sciences as high-resolution feeds from remotely operated network and grid-connected observatories and experimental equipment come online.[1] While statistical methods and feature detection/extraction algorithms have been proposed to automate the mining of useful information from these enormous data stores, a strong need still exists for a human component to the visualization process. It's 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 for effective interactive analysis of remote data stores and remote monitoring of simulation code runs that can span multiple days, or in some cases, weeks. Such tools must balance the competing interests of interactivity and fidelity to fit within the limits imposed by the available infrastructure. The emerging grid infrastructure ties these resources together into a global fabric, integrating distributed research teams and virtual organizations with the remote research hardware, software, and services that carry out their work.

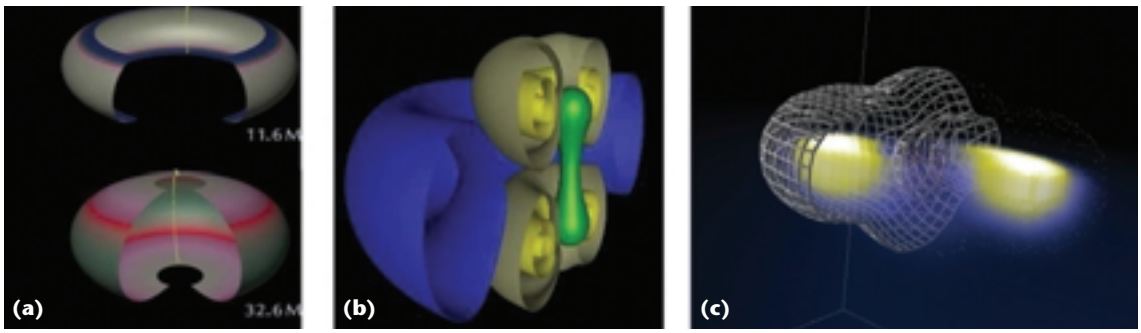
The NERSC and Lawrence Berkeley National Laboratory (LBNL) visualization group has developed the Visapult[2] tool to attack these sorts of grand challenge problems. Visapult is a distributed, parallel, volume-rendering application that leverages parallel computating and high-performance networking resources that are on the same scale as the supercomputers generating the data. Volume rendering is an important visualization technique for exploring complex 3D data sets, but its high computation and bandwidth requirements limit its applicability for interactive visualization unless we develop powerful hardware and software methods that can handle these demands. Traditional serial ray-tracing 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 technique called image-based rendering-assisted volume rendering (IBRAVR). The IBRAVR algorithm, described more completely elsewhere,[2,3] lets Visapult trade additional computational power in exchange for reduced bandwidth requirements.

We've improved Visapult's effectiveness using aggressive network tuning and network protocol modifications. In particular, we used a new connectionless user datagram protocol (UDP) to improve network efficiency from a 25 to 88 percent line rate increase for multigigabit networks. This connectionless protocol also dramatically reduces the latency of network event delivery, improving the responsiveness of wide-area distributed interactive graphics applications as compared to transmission control protocol (TCP) streams. We believe that this UDP protocol, as well as transport encodings and algorithms that can tolerate loss gracefully, will become a fundamental component of future grid visualization architectures.

Numerical relativity

A motivating application for this kind of grid-distributed visualization capability is the modeling of black hole collisions. One of the most challenging problems in astrophysics is the numerical simulation of Einstein's equations to explore his theory of general relativity. These equations are among the most complex in the world of physics; a set of nonlinear, hyperbolic, elliptic,

Published by the IEEE Computer Society



1 Images of black hole simulations. (a) The event horizon of the Schwartzchild spinning black hole colorized by its extrinsic curvature. (b) The Misner visualization shows the real component of the gravitational potential as blue, gray, and yellow isosurfaces (cut open so you can see inside), and the event horizon of the merging black holes as a green surface. (c) 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.

coupled equations containing thousands of terms when fully expanded. The General Relativity Group at the Albert Einstein Institute in Potsdam, Germany, developed the Cactus code[4] to solve these equations on supercomputers. The group's ultimate aim was 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 5-teraflop IBM SP (named seaborg) that has more than 3,000 processors. Large-scale massively parallel processors (MPPs) like seaborg must reach the target regime of the physics that they are used to explore, such as the spiraling merger of two black holes. Simulations that require comparatively small amounts of memory (approximately 1 Terabyte) allow accurate modeling of simpler phenomena like the Schwartzchild (spinning black hole) and Misner (head-on collision of two black holes) cases (see Figure 1). While simulating these cases is important for validating and calibrating the solvers used by the code, they're of little consequence to the endeavor of gravitational wave characterization and detection. The spiraling merger of two black holes offers a more compelling and astrophysically relevant source for detectable gravitational waves. At a minimum, simulating a basic spiraling collision requires 1.5 Tbytes of memory using bitant symmetry (reflection along the plane of rotation). A full 3D evolution requires a minimum of 3 to 5 Tbytes of RAM. It requires even more memory to move the outer boundaries far enough away from the event to perform the accurate gravitational wave extractions necessary to assist in the signal processing performed on Laser Interferometric Gravitational Observatory (LIGO) data.

Cactus' extremely advanced grid-computing capabilities reap the benefits of a globally distributed supercomputing infrastructure that serves a globally distributed virtual organization of astrophysicists. In one grid-computing paradigm, we can distribute the Cactus simulation across multiple computing sites to form a virtual supercomputer far larger than any individual machine. Cactus employs runtime adaptive metacomputing techniques that adjust the number of ghostzones on boundaries that cross the wide area network (WAN), as well as dynamic adjustment of the compression level that transports data for the boundary. In one experiment that captured a 2001 Gordon Bell Award, Cactus used more than 3,000 processors distributed across supercomputing sites on multiple continents and employed runtime adaptive techniques to improve the Cactus speedup from 15 to nearly 80 percent of its peak. In another scenario, the Cactus worm, a nomadic application, can discover more capable resources on the grid, then checkpoint itself and migrate to better resources using grid protocols and information services. The migratory, worldwide, and distributed nature of this application requires a visualization and analysis infrastructure that's similarly grid savvy.

These simulations are so large that it's often impossible to use traditional visualization tools to see and understand their results. Because gravitational physicists don't know the nature of the gravity wave signals they're studying a priori, data mining and feature-extraction techniques are inappropriate for this activity. It isn't feasible to download all of the data from the supercomputer center to a remote site for later analysis. Consider that a 3- to 5-Tbyte data set would take 11 to 18 hours to download if your system could reliably sustain a 1-Gbit stream using a single TCP stream (which is unlikely). Even preloading the data sets on your local visualization system is a dubious proposition because of the computed data sets' large size. Few high-end visualization systems have more than a few Gbytes of memory, or more than a few processors. The platform needed to analyze the data must be similar in scale to that used to produce the data, unless a scientist can employ significant data reduction or noninteractive out-of-core methods. Otherwise, for data analysis to be practical, some form of real-time dynamic data reduction would be needed before the data gets delivered to the scientist's desktop. This sort of data reduction is a natural consequence of many visualization methods.

Visapult allows the scientists using Cactus to view their simulation results in situ on the supercomputer as they're being produced, with the results made available to a remote viewer. Such remote monitoring improves the effective use of supercomputers by letting Cactus

users cut short runs that have gone awry, or 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 WAN throughput and interactivity approximating that of locally hosted applications.

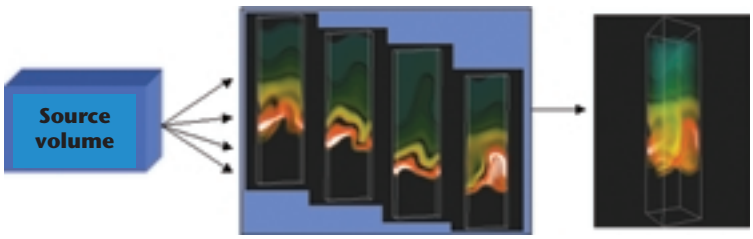
Direct volume rendering

Visapult has 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 subdomain. The system requires relatively little communication between the back-end processes, so the system scales well to large numbers of processors. The back end then transmits each of these images to a viewer, where they're texture-mapped onto static geometry and rendered using OpenGL. This process takes advantage of both the computing horsepower of MPPs 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 a desktop viewer is a high-performance extension to the IBRAVR algorithm.

Scientific data can be read into the Visapult back end in a domain-decomposed fashion, where each processing element of the back-end volume renders its block of data in software, and sends the resulting image to the viewer. This architecture results in drastically different I/O requirements along the processing pipeline. Whereas the bandwidth requirement to read data into Visapult's back end is N^3 (where N is the size of one axis of the computational grid), the bandwidth requirement between the back end and viewer is only (N^2). During the Department of Energy's Next-Generation Internet (NGI) Combustion Corridor project, our objective was to use network-centric resources—such as network-based storage and compute farms—to solve a single scientific problem. For this reason, we studied the performance of moving data across the WAN into the Visapult back end (see Figure 2). The $O(N^3)$ to $O(N^2)$ reduction in bandwidth requirements between data, back end, and viewer ideally suit Visapult for deployment on network topologies in which high-speed links connect resources, but where links to the viewer are over slower links. The multistreaming I/O of Visapult's back-end reader—combined with the flexibility to extend it to new types of data sources—has proven effective at using network resources and Visapult has won the Supercomputing/SCinet bandwidth challenge three times.

Performance

The various components of Visapult must communicate with one another using Internet protocol connections over a local area network (LAN) or WAN, depending on component deployment. To achieve maximum efficiency in the back end, data loading is performed asynchronously, so that I/O is overlapped with software rendering. 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's parameters let us reduce I/O use 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.

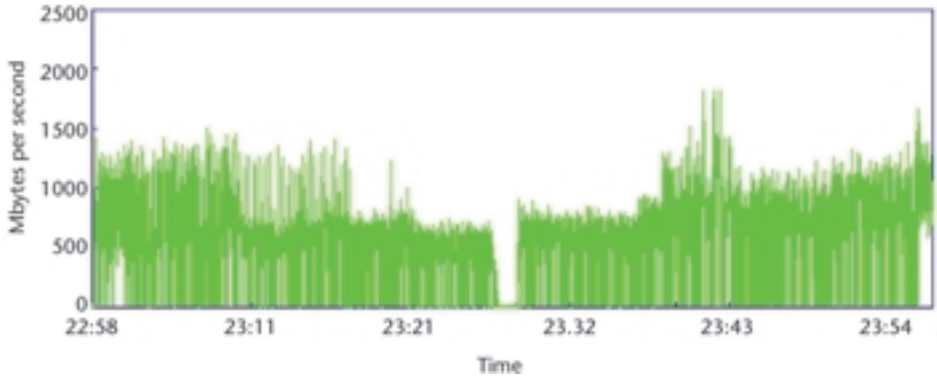


2 Visapult processing pipeline. The data source (either a running Cactus simulation or distributed parallel storage system) feeds the Visapult back-end using multiple parallel network data streams. Each process of the Visapult back-end volume renders its portion of the spatially decomposed domain into an image. These images are sent to a client application running on the user's workstation that uses the texture-mapping capabilities of commodity hardware-accelerated graphics cards to composite these images into a single scene at interactive rates.

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. Indeed, the preamble to Sally Floyd's High Speed TCP request for comments (RFC) draft (http://www.ietf.org/internet-drafts/draft-floyd-tcp-highspeed-00.txt) points out that to achieve steady-state 10 Gbit/sec throughput using standard TCP implementations with 1,500 maximum transmission units and 100 ms round-trip latency, you must have at most only one congestion event for every 5 billion transmitted packets. That means only one lost packet every 1 2/3 hours to reach this bandwidth. This is clearly not practical under any circumstance and yet wide-area, distributed supercomputing systems like the Teragrid that use multiple transcontinental 10-Gbit links are already coming online. We must employ alternative methods to make effective use of these expensive assets.

TCP multistreaming

The typical solution to work around TCP congestion avoidance is to use multiple simultaneous TCP connections. This is an old solution most commonly employed in Web browsers dating back to xmosaic, and is described in Stevens' classic book on network programming.[5] This technique works because streams don't share packet loss and window-size information with one another, so a lost packet will only cause one of the streams to back off



3 Graph of network throughput for Visapult during the SC 2000 bandwidth challenge. The performance was erratic over a one-hour 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.

rather than all of them. The effect is to make the hypersensitive TCP congestion avoidance algorithm artificially less sensitive to packet loss, but this undoes the fairness implicit in the congestion avoidance algorithm. In effect, a multistreaming application becomes a network bully that competes unfairly with its peers. Moreover, it's difficult to tune the parameters of TCP multistreaming to maximize performance. If you wanted to use this technique to reach 10 Gbits, as per the Floyd example, you would have to employ approximately 4,000 simultaneous TCP streams to reach full-line rate. We typically implement this kind of protocol arrangement at the application level in an ad hoc manner, but GridFTP (http://www.globus.org/datagrid/gridftp.html) formalizes this methodology into a standard protocol.

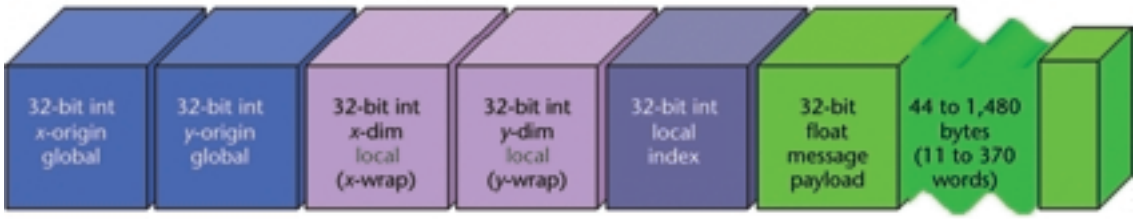
Visapult's TCP implementation

We used this multistreaming method for the first implementation of Visapult, developed under the NGI initiative's combustion corridor project. The distributed parallel storage system (DPSS), developed by LBNL's Distributed Computing Group, provides a data source that improved multistreamed TCP performance by striping the network streams across multiple hosts and network adaptors.[6] We carefully tuned the TCP window sizes, buffer sizes, and blocking parameters to maximize performance. To achieve complete overlap of computation and I/O, the network performance must be extremely high; hence, the aggressive use of TCP multistreaming. At the SC 2000 bandwidth challenge competition, we achieved a peak throughput rate of 1.5 Gbps on an OC-48 link and achieved a sustained throughput rate of about 660 Mbps over a 60-minute window. 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 percent (peak) and 25 percent (sustained) of the theoretical line rate, respectively, of the OC-48 link. The network throughput was extremely erratic, as Figure 3 shows. The gap in Figure 3 (about halfway through the run) occurred as a result of an application crash, which required a restart. On a shared WAN, more than 50 percent use is still admirable, but these were dedicated links that were entirely uncongested. The networking research community has surmised that the deficiencies in TCP performance may simply be a product of an application programmer's limited skill in tuning network applications compared to networking experts (the so-called "wizard gap"). However, the TCP throughput remained inefficient despite aggressive tuning of the TCP parameters by scientists from LBNL's networking research programs. This indicates that we can't simply blame the wizard gap. Therefore, even with tuning and a dedicated link, we can't efficiently exploit the link bandwidth using the TCP protocol.

Moving to unreliable protocols

For data transfer and replication, data integrity is paramount. Response time and performance is of comparable importance to data integrity for visualization applications. Visualization tools almost invariably use reliable transport protocols to connect distributed components because of the general concern that lifting the guarantee of data integrity would compromise the effectiveness of the data analysis. However, visualization researchers find other forms of lossy data compression acceptable, like JPEG, wavelet compression, and data resampling. Such acceptance is perhaps because degradation in visual quality is well behaved in these cases. Networking experts have long turned to connectionless/UDP protocols when an application requires maximum responsiveness and low latency. An unreliable transport mechanism that deals with packet loss gracefully without extreme visual artifacts could 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.

Visualization applications also require extremely low-latency transport to maintain interactive responsiveness. Consider that the response time for TCP is twice the roundtrip time (RTT) plus the additional overhead of processing the information on the hosts involved in the transfer. The TCP responsiveness gets far worse when retransmission and data buffering occur. During retransmissions induced by packet loss, stream-oriented protocols will block the data stream until a successful retransmission occurs. Retransmissions result in huge variations in responsiveness and throughput, such as those observed during the SC 2000 implementation. Application scientists often mistakenly attribute this erratic responsiveness (caused by the TCP algorithm) to network jitter. However, the network jitter is a layer 2 (transport layer in Ethernet jargon) phenomenon that's caused by the nonuniform delay of individual packets—on the order of milliseconds—caused by buffering inside



4 Diagram of the UDP packet format.

the network-switching hardware. Conversely, the erratic behavior of TCP (layer 3 in Ethernet jargon) and the application (layer 4) is the real culprit here and typically manifests itself on much larger time scales with much broader and more noticeable effects to the person using the application. A well-designed UDP protocol, by contrast, provides the lowest possible latencies over the WAN with little OS buffering or protocol-induced delays and nearly immediate response.

Visapult implementation using unreliable transport

After the lessons learned during the SC 2000 bandwidth challenge, we focused on performance improvements that would result from employing more aggressive use of network protocols. We modified Visapult's band end to connect 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, we can use Visapult as a component of an interactive remote steering system composed of grid-distributed components.[4]

For the SC 2001 and 2002 bandwidth challenges, we modified the back end of Visapult to work with our own custom UDP transport protocol. The thesis of our work was that using a connectionless protocol would produce dramatically more efficient bandwidth use as well as more consistent and rapid responsiveness. The SC 2000 Visapult reader used a TCP protocol and requested data from the network in a specific order; TCP guarantees in-order delivery of data. In contrast, UDP makes no such guarantees, so each UDP packet must contain information indicating the location in the domain-decomposed array where the Visapult back end must place the data payload (see Figure 4). This ensures that the system can treat each packet independently so that packet ordering and loss have minimal effect on destination processing. We modified the Visapult back end to render continuously rather than waiting for all packets in a given frame to arrive. The visual effect of this choice is an immediate response involving a coarser representation with 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.

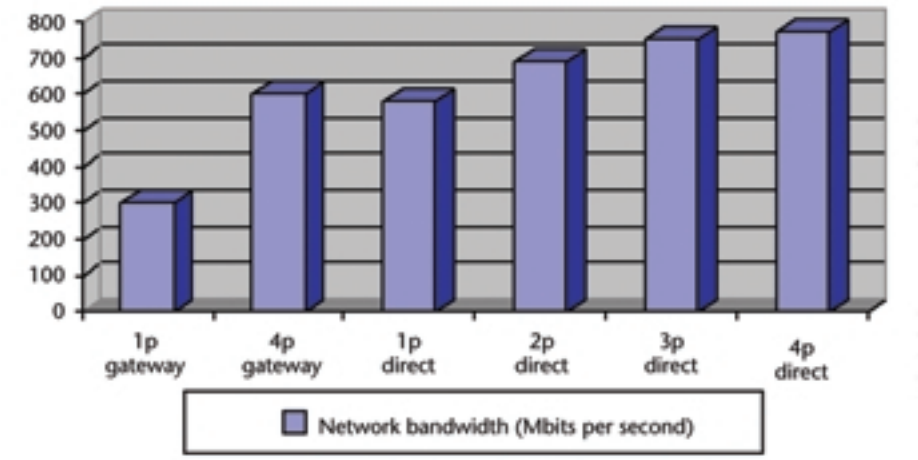
We used a 20-byte header in the Figure 4 implementation to locate the payload of each packet in the destination domain independently of one another. Assuming a full 3D block domain decomposition, the origin and dimension information is superfluous in the z-dimension. The packet header contains the following information:

- *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).
- *Local index*: the offset in cells counted from the start of the local domain decomposed block in the data source.

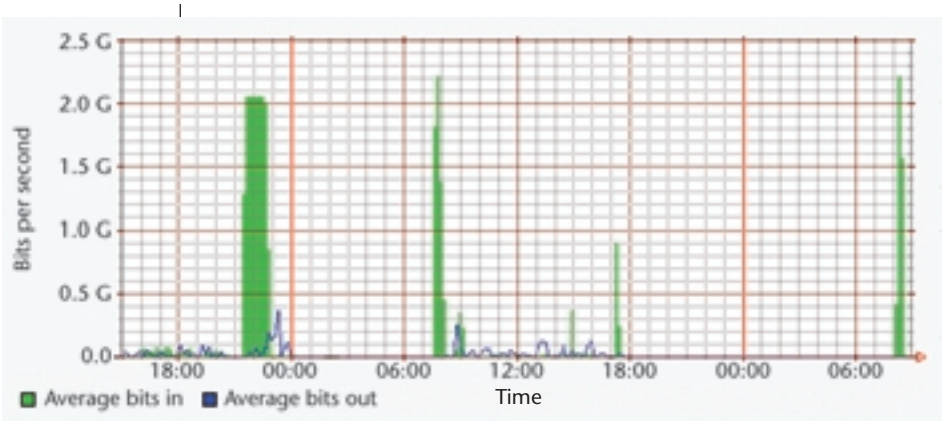
This source-based data indexing lets the component sending the data be ignorant of the domain decomposition at the destination, but provides enough information at the destination to support unambiguous data reassembly.

We contemplated a multibuffering scheme to let all possible packets arrive before rendering. In practice, such a technique would provide improvements in visual quality when the data in successive frames changes rapidly. However, because of the slow rate of data evolution for our remote monitoring application, the single-buffered approach proved more effective for the visualization tool. We continue to investigate this sort of multibuffering.

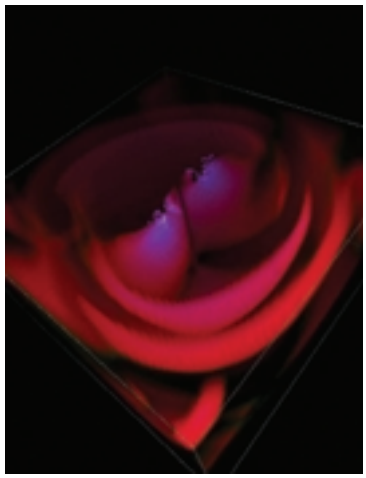
The Cactus/Visapult thorn (AlphaThorns/ShmServ) buffers data in a shared memory staging area created when the code begins. The simulation code spawns a set of background worker processes (which we term *NetWorkers*) at startup time that are dedicated to reading data out of the shared memory region and sending it over the network to the Visapult back end. We implemented the NetWorkers as processes rather than threads to get around the performance problems and scheduling overhead associated with some particularly poor vendor implementations of Posix standard threads. The NetWorkers require a dedicated CPU so that they don't interfere with scheduling and execution of the primary simulation code's processes. Filling a Gigabit Ethernet connection to capacity will fully engage even a powerful CPU. The NetWorkers have parameters that let the code set a fixed packet rate that we can tune at runtime 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 fill the Gigabit Ethernet interface on larger summetric multiprocessors like the 16-way IBM Nighthawk II SP nodes (see Figure 5). For our dual-CPU Linux hosts, we dedicated one processor to network I/O while the other per-



5 UDP performance on the IBM Nighthawk II SP using the internal network sending through Gigabit Ethernet on the gateway node (1 p-gateway and 4 p-gateway mean one processor on one node and four processors on four nodes routed through the gateway node, respectively) and one to four NetWorker processors (1p to 4p direct) on 16 CPU Nighthawk nodes with directly attached Gigabit Ethernet NICS.



6 The daily network traffic log from the SCinet Core1 router during the 2001 bandwidth challenge that monitored the OC-48 traffic from NERSC. All three network traffic peaks that exceeded 2.0 Gbits are from Visapult testing (the last being the actual bandwidth challenge run).



7 Volume rendering of the real component of gravitational potential.

formed computing functions. Under this arrangement, we could achieve 960 Mbits per full-line-rate per node.

In 2002, we demonstrated full use of a 10-Gbit Ethernet pipe using a modestly sized (12 node) Linux cluster for the Visapult backend component (http://www.supercomputingonline.com/nl.php?sid=2252).

Bandwidth challenge results

We previously discussed the SC 2000 results in the section "Visapult's TCP implementation." These results motivated us to completely rewrite the networking code from scratch (dropping the TCP in favor of our own custom connectionless protocol). At the SC 2001 high-performance Visapult run, we used 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 six nodes of NERSC's IBM SP system. We also ran a related apparent horizon finder on a 128-node Origin 2000 system at NCSA. At NERSC, the Gigabit Ethernet interfaces on five of the SP 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 ran on an eight-node Linux cluster on the conference's show floor that was connected to a 10-Gbit Force10 switch. At NCSA, a single Gigabit Ethernet on the host fed the NCSA OC-12 uplink and fed a 32-way Sun Starfire symmetric multiprocessor system in the Sun booth on the show floor.

The NERSC system reached throughput rates of 2.38 Gbps on the OC-48 link. This is 96 percent of its theoretical capacity, and it reached this rate without the typical ramp up associated with TCP-based applications (see Figure 6). The total aggregate throughput of the application from all sources was 3.3 Gbits. 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 wasn't compromised by the unreliable transport method. The gains in interactive performance were dramatic. While there were some lost packets, the loss rate was easily tailored to be less than 1 percent by regulating the packet-sending rate. When packets were lost, the resulting visual artifacts were minor to the casual observer, but quite obvious to one familiar with the science. However, the high bandwidth efficiency resulted in rapid recovery from these artifacts. Overall, the move to unreliable transport

greatly enhanced the effectiveness of the application for remote analysis of extremely large or dynamic data sets (Figure 7).

We used a very similar software configuration with some protocol improvements that offered more accurate rate control for the SC 2002 bandwidth challenge. The 2002 effort involved a global grid testbed collaboration that assembled a globally distributed set of resources to win the challenge with a sustained data rate of 16.8 Gbits per second. This constitutes a more than five-fold improvement over the bandwidth record set in the previous year and demonstrates the importance of network protocol improvements relative to advances in network hardware performance.

User impact

While visualization researchers have become quite comfortable with data compression and reduction methods, we have a concern that scientists might balk at any artifacts or scintillation in the displayed images. It's too early to fully evaluate the impact on our scientific users given the prototype nature of the current tool. As we move this technology into production, we're considering many options to respond to these potential concerns. The current implementation can recover from loss quite rapidly by sending redundant packets, which greatly reduces the probability of gaps in the final representation. We can also offer a reliable UDP or lower-performance TCP back channel that operates in parallel with the primary data delivery stream. This ensures that we can generate a final, uncompromised visual representation after a suitable delay. In addition to our own application-specific implementation, a number of reliable UDP implementations are available in the networking community, including Reliable UDP, Rapid, and SABUL.

Future directions

The UDP method we have described relies on pacing packets to meet, but not exceed, network capacity. Indeed, there's no reason that we can't apply these fixed-data-rate methods trivially to reliable transport protocols like TCP. It's clear that fixed-rate implementations of both reliable and unreliable protocols can be disruptive to commodity networks, and are most appropriate for use on dedicated network links, private virtual circuits, experimental networks, or even for scheduled access. While it's unreasonable to assume that all WAN connections will be dedicated links, it's 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. Such dedicated bandwidth is more typical in high-performance networks for the sciences like ESNet, Abilene, and Canarie. Rate-limited flows will prove essential for data visualization corridors and ultrascale grid visualization architectures of the future to take advantage of the capability offered by these forthcoming ultra-high performance backbones.

Rate-limited flows will prove essential for data visualization corridors and ultrascale grid visualization architectures of the future.

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. However, this problem isn't unique to UDP since the same tuning must occur even for multistream TCP implementations. Multistreaming 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. Using too many TCP streams can quickly lead to the same congestion situations that occur with UDP.[7] However, unlike the UDP method, no straightforward method exists to regulate bandwidth use of multistream TCP flows. In this respect, multistream TCP is superfluous because it's possible to craft a congestion avoidance algorithm for any single-stream UDP-based protocol that models the behavior of single-host multistream TCP by simply modifying its response to loss. The Web100 workaround daemon uses an artificially large virtual maximum transmission unit to recover more quickly from packet loss. (More information on this Web100 concept paper is available at http://www.web100.org/docs.concept_paper.php.)

Even with raw UDP, we can incorporate TCP-friendly methods described by Mahdavi and Floyd[8] for interactive rate control and provide guidelines for appropriate packet rates to minimize loss and limit impact on other network users. However, the TCP-friendly assumption that any packet loss is indicative of congestion is the very characteristic that leads to TCP's poor performance on high-bandwidth pipes.

Considerable research exists on traffic-shaping methods (active queue management) and statistical methods to improve the quality of this assumption. Even the most effective of these methods (random early detection) turns out to be quite difficult to tune. However, we can hold the extreme position that only the switching fabric can provide the necessary information to help the end points differentiate congestive from noncongestive packet loss.

Ideally, the network-switching fabric should provide detailed quality of service (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's noncongestive, but would default to the standard congestion avoidance algorithm when no such hints are available. Proposed methods like explicit control protocol (XCP), explicit congestion notification (ECN), and core-stateless fair

queueing (CSFQ) advocate this approach. These methods require significant modifications to existing router- and network-switching hardware designs, which is a distant prospect.

Grid-based bandwidth brokers or networking-monitoring services, such as LBNL's Network Control System (NCS) or those proposed for the Quanta QoS project (read more about this testbed project at http://www.evl.uic.edu/cavern/teranode/quanta.html), offer a means to provide direct feedback about network congestion in the absence of new network hardware designs. These methods can use existing QoS protocols built into network equipment or post feedback information on a Globus Metacomputing Directory Service (MDS) to manage fixed-rate transport services on these sorts of higher-performance network backbones so as to avoid congestive loss. Even without intelligent switching fabric or QoS, we could create a system of peer-to-peer feedback/autonegotiation by having end points multicast their path and current packet-rate information on a fixed set of designated paths. This would let hosts negotiate between themselves for appropriate packet rates rather than involving a third party, such as a bandwidth broker or the switching fabric itself.

Ultimately, it's 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 and 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 data encoding 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.[9] Such techniques make packet loss less visually distracting and eliminate the need for data retention on the sending side. Any reliable technique requires data retention 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 use 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 to 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 grid-enabled visualization tools and distributed visualization component architectures. Such aggressive methods are necessary to overcome the limitations of the aging TCP protocol for high throughput applications on high-speed WANs. Using such techniques has produced a 300 percent improvement in I/O efficiency over the best available tuned, multistreaming TCP methods. While packet delivery isn't guaranteed, the results are comparable to other lossy data-reduction techniques commonly employed in visualization. In addition, connectionless methods offer much lower latency and better responsiveness than TCP streams under the same conditions. Ultimately, grid architects must reevaluate exclusive reliance on TCP-based reliable transport for distributed interactive applications like visualization, because this greatly impedes our ability to exploit high-performance network-interconnected grid resources. ■

Acknowledgements

The Director, Office of Science, of the US Department of Energy under contract DE-AC03-76SF00098 and the NERSC supported this work. This work was carried out in collaboration with the Cactus team at the Albert Einstein Institute and the Max Planck Institute, and we especially thank Ed Seidel, Gabrielle Allen, Denis Pollney, Thomas Radke, and Peter Deiner. Syskonnect and Force10 Networks (particularly Raju Shah) provided key equipment and expertise for these experiments. The following institutions contributed computing and networking resources to the testbed: NERSC, NCSA, Masaryk University in the Czech Republic, University of Amsterdam, Argonne National Laboratory, and Pittsburgh Supercomputing Center. Also thanks to SCinet, CESNET, StarLight/StarTap, Level-3, Abilene, LBLNet, ESNet, and Qwest for setting up the network infrastructure for the bandwidth challenges. And finally thanks to our LBNL bandwidth challenge team for many late nights of sweat and toil: Mike Bennett, John Christman, Eli Dart, Jason Lee, Chip Smith, and Brian Tierney.

References

1. T. DeFanti and M. Brown, eds., *Report to the National Science Foundation Directorate for Computer and Information Science and Engineering (CISE) Advanced Networks Infrastructure and Research Division*, tech. report, Electronic Visualization Laboratory, Univ. of Illinois, Chicago, Dec. 2001, http://www.evl.uic.edu/activity/NSF/final.html.
2. W. Bethel et al., "Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization," *Proc. IEEE/ACM 2000 Conf. Supercomputing*, CD-ROM, 2000.
3. K. Meuller et al., "IBR-Assisted Volume Rendering," *Proc.*

IEEE Visualization 99, IEEE CS Press, 1999, pp. 5-8.
4. G. Allen et al., "Cactus Tools for Grid Applications," *Cluster Computing 4*, Kluwer Academic Publishers, 2001, pp. 179-188, http://www.cactuscode.org/Showcase/Publications.html.
5. W.R. Stevens, *Unix Network Programming, Volume 1: Networking APIs*, Prentice Hall, 1997.
6. B. Tierney et al., "A Network-Aware Distributed Storage Cache for Data Intensive Environments," *Proc. IEEE High-Performance Distributed Computing Conf.* (HPDC-8), IEEE CS Press, 1999.
7. T.J. Hacker, B.D. Athey, and B.D. Noble, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," *Proc. 16th Int'l Parallel and Distributed Processing Symp.*, 2002, http://mobility.eecs.umich.edu/papers/ipdps02.pdf.
8. J. Mahdavi and S. Floyd, *TCP-Friendly Unicast UDP Rate-Based Flow Control*, tech. note, 8 Jan. 1997, http://www.psc.edu/networking/papers/tcp_friendly.html.
9. H. Hoppe, "Progressive Meshes," *Proc. 23rd Int'l. Conf. Computer Graphics and Interactive Techniques* (Proc. Siggraph 96), ACM Press, 1996, pp. 99-108.

E. Wes Bethel *is a staff scientist at Lawrence Berkeley National Laboratory, where he is a member of the Scientific Visualization group. His research interests include computer graphics and visualization software architecture, remote and distributed visualization algorithms, and latency-tolerant and parallel graphics techniques. He received an MS in computer science from the University of Tulsa. He is a member of the IEEE and ACM.*

John Shalf *is a staff scientist at Lawrence Berkeley National Laboratory. His research interests include distributed/remote visualization, grid portal technology, high-performance networking, and computer architecture. He's a member of the technical advisory board for the EU GridLab project, which seeks to create application-oriented programming interfaces and frameworks for grid computing.*

Readers may contact E. Wes Bethel or John Shalf at Lawrence Berkeley National Laboratory, 1 Cyclotron Rd., Mail Stop 50F, Berkeley, CA 94720, email {jshalf, ewbethel}@lbl.gov.