How the Grid will affect the Architecture of Future Visualization Systems

John Shalf [1] and E. Wes Bethel
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

Abstract

The promise of Grid computing, particularly Grid-enabled visualization, is to employ a transparent interconnect fabric to link data sources, computing (visualization) resources and users into widely distributed Virtual Organizations (VOs) for the purposes of tackling increasingly complex problems. However, there exists a wide gulf between current visualization technologies and the vision of global, Grid-enabled visualization capabilities. In this article, we discuss our views of how visualization technology must evolve in order to be truly effective in a Grid environment.

The Vision

Dr. Jane is a geophysics researcher who is studying the effects of how a new material, code-named "zorbs," can mitigate the effects of earthquakes on urban structures. The material works by isolating structures from direct exposure to shock damage through direct injection into porous media upon which structures are built. In the course of her studies, she runs experiments at the molecular scale, where she observes molecule-molecule interactions, and also at the urban scale, where she uses large equipment to induce shock waves into the ground. She has access to instrumentation attached directly to urban structures in Shaky City, where real-time data is collected and stored at a nearby data center. Dr. Jane runs large-scale simulations that spawn off subsimulations around the world, wherever cycles are available. The subsimulations, when complete, report back to the master simulation with the results of parameter studies. Because the simulations routinely generate terabytes of results, data is stored on data caches located close to the computational resource and is later analyzed and visualized. She routinely uses experimental data to seed her simulations, and in turn, uses simulation results to modify her experimental parameters. When an earthquake hits, she is able to immediately connect to a vast network of sensors to observe first-hand how her handiwork has reduced direct shock damage caused by the pressure and shear waves generated by earthquake shaking. Such observation can occur using a desktop workstation, or using a handheld telephone. Dr. Jane is part of a large, multidisciplinary team that is scattered around the world. All collaborators have access to the entire collection of experimental and simulation data, which may reside at any one of the member institutions.

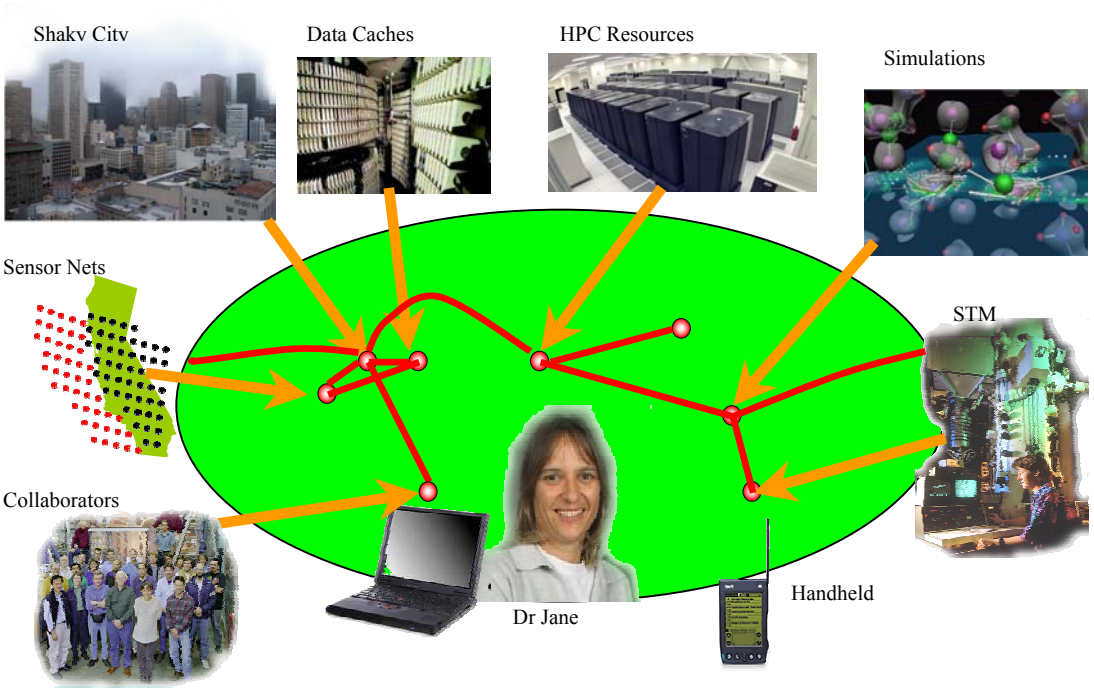


Figure 1. Grid-Based Visualization and Computing Applications Comprised of Heterogeneous Resources

As you may have guessed, this description is fiction. However, the scope of activities performed by Dr. Jane is not far fetched, and is reasonably characteristic of many modern research scientists. Indeed, the promise of "Grids" is the ability to weave together a collection of disparate resources into a single application in exactly the fashion we have suggested. As visualization researchers and developers who work with Dr. Jane's on a routine basis, and who are tasked with designing and building visualization tools that provide such capabilities, it is our position that the current state of visualization software is not Grid-ready. In fact, we believe that a fundamental paradigm shift is needed in order to create fully global

1. The authors can be contacted at jshalf@lbl.gov and ewbethel@lbl.gov.

visualization applications. What is needed is a new framework for distributed visualization that is Grid-aware, that is easy-to-use, and that is modular, extensible, and which permits us to reuse our existing investments in visualization technology.

Getting From Here to There

A number of successful, contemporary visualization systems, such as VTK, AVS and OpenDX, are constructed using a pipelined, component-based architecture. In such an architecture, a user can quickly assemble modular software components into a "finished application." These systems have been successful because they are both flexible and extensible. They are flexible in the sense that components can be combined in a multitude of ways, thereby allowing an application developer to accomplish a wide variety of visualization tasks. They are extensible as they offer the means for developers to add new components to the system, thereby extending the system's functionality.

Whereas the characteristics of such systems deployed on a single machine are relatively well understood, and some even offer the capability of running distributed across a limited number of machines, deploying such systems onto the Grid requires consideration of new conditions not likely to have been anticipated during their initial design and implementation. It is our viewpoint that these additional design considerations, some of which are listed below, are best met through a Distributed Visualization Architecture (DiVA), a completely new framework designed for distributed, component-based, Grid-enabled visualization.

Distributed, Heterogeneous Components. Building upon the architecture of successful, contemporary systems, future Grid-based systems will likely use a component-based architecture. At their core, these distributed components will likely be quite similar to their single-machine cousins – they will perform a well-defined operation on strongly typed input data to produce a result. Currently, modules from one system are most likely incompatible with those of another or are bound to a particular user interface paradigm, which has the effect of creating isolation among users and within the visualization community. Users of visualization systems want to be able to use the best tool for the job, regardless of its source[1]. In order to achieve such a goal, a plethora of engineering issues and cultural differences must be resolved. One of these issues is the nature of a flexible HPC-oriented component interface, which is the subject of active research. For instance, the Common Component Architecture[2]. (CCA) effort seeks to create an Interface Description Language and automatic language wrappers that are oriented towards HPC application requirements. The Advanced Collaborative Environments Research Group[3] (ACE) of the Global Grid Forum[4] (GGF) is defining the security requirements for this sort of component architecture.

Fundamental Graphics and Visualization Algorithms. The hallmark of Grid-based deployments is a focus on remote visualization. Typical approaches to remote visualization in the past have focused on one of two broad approaches. The first approach is to perform all the visualization on the server, and send image data (or X-protocol streams) to the client. The alternative approach has been to transfer subsets of the large scientific data from the server to the client workstation, where visualization is performed completely on the desktop machine. The former approach works best when the size of the dataset is large. The latter approach works best when interactivity is most important. However, these requirements can change dynamically at runtime as we show in the Sidebar. New trends in graphics and visualization have produced a new family of algorithms that will be increasingly important for future Grid-based visualization systems. These *latency tolerant* algorithms seek to maintain interactivity on the desktop, while providing the ability to perform visualization of very large datasets. Visapult[5] is an example of pipelined-parallel IBR-accelerated volume rendering, while the TeraScale Browser[6] uses multiresolution methods and pipelined data caching to achieve interactive, latency tolerant visualization. More algorithms of this type need to be integrated into a common framework in order to perform interactive, desktop visualization of large, remotely located data sources using distributed visualization components.

Distributed Execution and Dynamic Scheduling. Most contemporary component-based visualization systems, which manage the tasks of component launching and data movement, are typically designed for use on a single platform. As such, these systems will likely not prove effective in Grid environments. A common practice is to use a manual "staging" of components on machines into an execution pipeline. This approach is impractical in Grid environments where there may be hundreds or thousands of resources used by a single application. A better approach is the notion of a database or directory service that distributes and keeps track of components, both executables and running instances, on heterogeneous resources. Fledgeling services of this form are provided in Grids based upon the Globus[7] architecture in the form of Monitoring and Discovery Service[8] (MDS), which is essentially an LDAP-based hierarchical directory of resources for a given Virtual Organization (VO).

End-to-end Performance. The practice of static pipeline configuration for distributed resources offers no guarantee of better performance than would result from having all components run on a single machine. While the performance of an individual component in a system is straightforward to quantify, modeling the performance of the overall system of distributed components is needed in order to effectively use all resources. In order to select a reasonable pipeline arrangement, we need to predict the performance of candidate pipelines before they are launched. Such modeling is non-trivial, and will have a profound impact on the architecture of future systems. However, since the underlying resources change dynamically in typical Grid Computing Environments and the performance of the various visualization algorithms is highly dependent on parameter choices and the data they are fed, distributed visualization architectures must have continuous runtime performance monitoring to dynamically select between different resources, work distributions, and algorithm alternatives. The Grid Application Development Software[9] project (GrADS) is starting to grapple with performance prediction for distributed applications. Other efforts like the EU GridLab[10] project seek to create API's that hide the complex dynamic nature of Grid resources from application developers. The impact of such dynamic selection on performance is illustrated in the Sidebar.

Conclusion

Extending current visualization tools so that they are Grid-enabled is a daunting challenge due to the breadth and depth of issues, which we have lightly examined in this column. At one end of the spectrum, careful performance analysis and optimization is needed to make effective use of resource in dynamic environments. At the other end of the spectrum, new fundamental graphics and visualization algorithms are needed in order to implement a family of latency-tolerant tools suitable for use in remote and distributed visualization settings. In this article, we have briefly mentioned only a few of the numerous issues related to implementing and deploying effective Grid-based solutions. The approach we favor leverages upon lessons learned from successful, contemporary visualization packages. Specifically, it is our position that the best solution is for a new framework that "hides" the complexity of implementing fully Grid-enabled distributed visualization tools built from software components. Our vision is for a DiVA, a framework that supports high performance, latency tolerant, and heterogeneous components, with an underlying theme of reuse of existing component-based visualization technology. Such a vision serves to promote growth of visualization technology into Grid environments, and to promote unity within the visualization and computational science communities.

Acknowledgement

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information, and Computational Sciences Division, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References and Further Reading

1. Findings of NERSC Visualization Greenbook Workshop, http://vis.lbl.gov/Publications/2002/VisGreenFindings-LBNL-51699.pdf
2. http://www.cca-forum.org.
3. http://calder.ncsa.uiuc.edu/ACE-grid/main.html.
4. http://www.gridforum.org.
5. 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 SC00, November 2000.
6. V. Pascucci and R. Frank, "Global Static Indexing for Real-time Exploration of Very Large Regular Grids," Proceedings of Supercomputing 2001 Conference, Denver, CO, Nov 10-16 2001. UCRL-JC-144754.
7. http://www.globus.org.
8. http://www.globus.org/mds.
9. http://hipersoft.cs.rice.edu/grads/gradsoft.htm.
10. http://www.gridlab.org.

Sidebar 1: Performance Modeling and Pipeline Partitioning of Distributed Visualization Systems

Most remote and distributed visualization applications use a static partitioning of the visualization pipeline, and hope that such a partitioning works well for all situations. The example in this sidebar serves to illustrate the importance of dynamic partitioning for distributed visualization applications. Figure S1 shows the components of the visualization pipeline used to create an image of an isosurface. The links between the blocks indicate data flow.

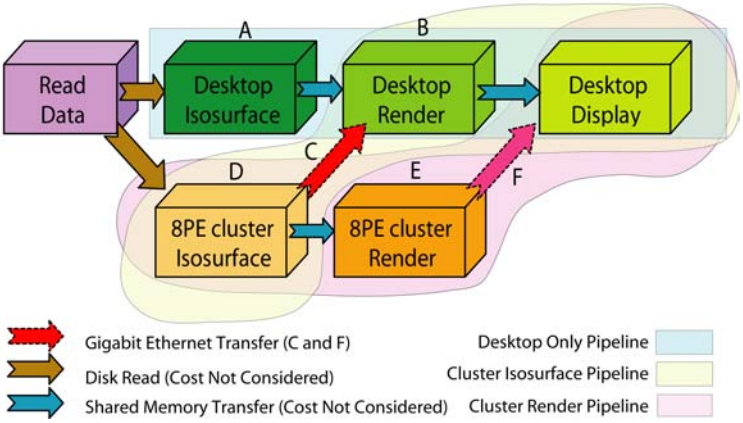


Figure S1: Components and Data Flow for our hypothetical example. Arrows indicate potential data flow paths. There are three potential partitionings in this graph.

In Figure S1, there are three possible ways to partition the visualization pipeline. The *Desktop Only* pipeline places the isosurface and rendering components on the desktop machine. The *Cluster Isosurface* pipeline places a distributed-memory implementation of the isosurface component on the cluster, and transfers triangles over the network to a serial rendering component on the desktop. The *Cluster Render* pipeline performs isosurface extraction and rendering on the cluster, then transfers images to the desktop for display. The colors used in Figure S2 correspond to the groupings of components shown in Figure S1: cyan for the *Desktop Only* pipeline, yellow for the *Cluster Isosurface* pipeline, and magenta for the *Cluster Render* pipeline. In order to measure the performance of each of these pipelines, we make the following optimisting and simplifying assumptions:

- *All triangles produced by the isosurface component are triangle strips. Each incremental triangle of the strip is represented with a single vertex, which consumes twenty four bytes (six, four-byte floats comprising vertex and normal information).*
- *All graphics hardware is capble of rendering 50M triangles/second. An 8-node system can render 400M triangles/second.*
- *Isosurface creation takes 1 second on the desktop, and .125 seconds on the eight node system.*
- *The image transfer assumes a 24-bit High Definition 1920/1080p CIF framebuffer*
- *Interconnects use a 1 Gigabit network with perfect performance.*
- *The performance model does not consider the cost of data reads, the cost of scatter-gather operations, or the cost of displaying cluster-rendered images on the desktop.*

Whereas Figure S2 shows the absolute runtime of each pipeline for varying numbers of triangles, Figure S3 presents the relative runtime of all components for varying numbers of triangles. In Figure S3, the absolute runtime of all components is summed, producing 100%, and the size of each colored segment in each vertical bar shows the relative time consumed by that particular component. Each component's execution time is normalized by the sum of all component execution times so that we can quickly determine which components, or network transfers, will dominate the execution time. Note that some components are used in more than one pipeline configuration. The vertical bars in Figure S3 are color-coded by component: those in shades of green are desktop-resident, those in shades of red are network transfers, and those in shades of orange are cluster-resident. The components are arranged vertically so the reader can visually integrate groups of adjacent components into their respective pipeline partitionings. In the 500K triangles case, the cost of desktop isosurface extraction dominates in the *Desktop Only* pipeline. In contrast, the *Cluster Isosurface* pipeline would perform very well – about six times faster. In the 500M triangles case, the *Cluster Render* pipeline is about five times faster than the *Desktop Only* pipeline, and about eight times faster than the *Cluster Isosurface* pipeline.

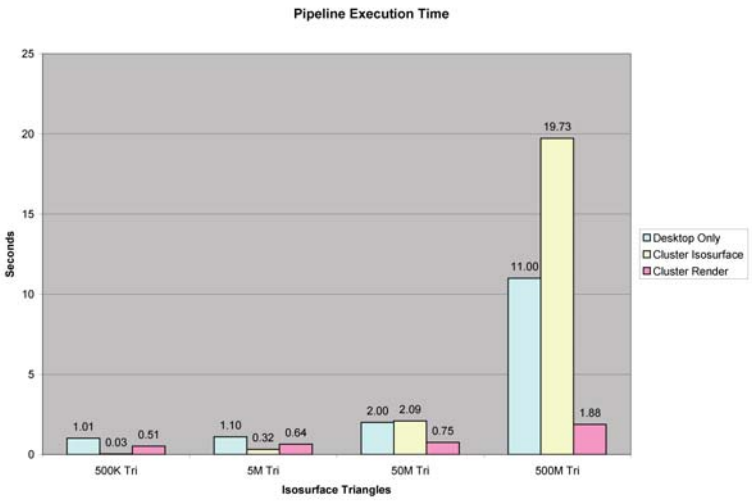
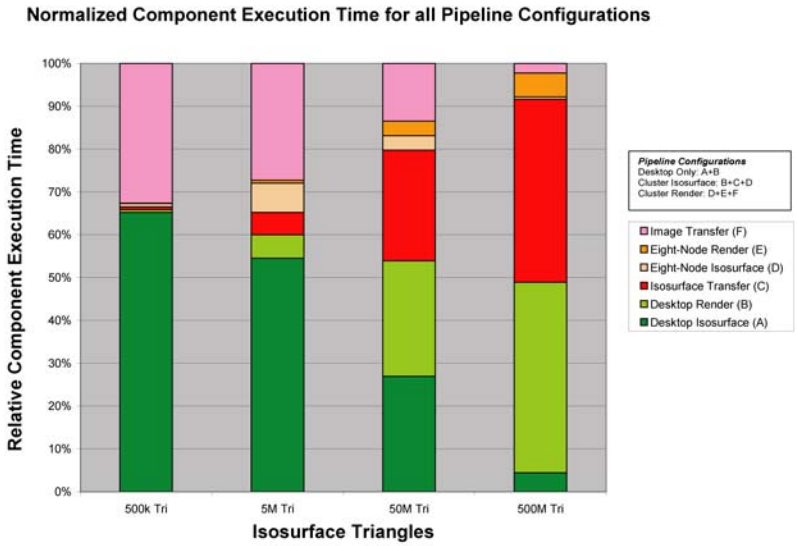


Figure S2. Absolute Runtime for Each Pipeline. *Desktop Only* performance is the sum of components A and B. *Cluster Isosurface* performance is the sum of components D, C and B. *Cluster Render* performance is the sum of components D, E and F.

This example serves to illustrate how optimal pipeline partitioning can change as a function of a simple parameter change. Creating and deploying such dynamic repartitioning strategies on the Grid requires consideration of many more variables than we have discussed in this article, and a more accurate performance model. Grid-based services are intended to help measure and provide such information to applications, but they do not perform the kind of dynamic partitioning needed to support Grid-



based visualization. A substantial body of new infrastructure work is required in order to support effective Grid-based visualization tools.

Figure S3. Relative Performance Components in the Three Pipelines