Performance Modeling for 3D Visualization in a Heterogeneous Computing Environment

Ian Bowman[1,2] John Shalf[2] Kwan-Liu Ma[1] Wes Bethel[2]
[1]University of California at Davis [2]Lawrence Berkeley National Laboratory
{bowman,ma}@cs.ucdavis.edu {ewbethel,jshalf}@lbl.gov

Abstract

The visualization of large, remotely located data sets necessitates the development of a distributed computing pipeline in order to reduce the data, in stages, to a manageable size. The required baseline infrastructure for launching such a distributed pipeline is becoming available, but few services support even marginally optimal resource selection and partitioning of the data analysis workflow. We explore a methodology for building a model of overall application performance using a composition of the analytic models of individual components that comprise the pipeline. The analytic models are shown to be accurate on a testbed of distributed heterogeneous systems. The prediction methodology will form the foundation of a more robust resource management service for future Grid-based visualization applications.

1 Introduction

In scientific analysis and visualization, scientists or researchers would often like to use their desktop workstations to visualize data located at a remote site such as a supercomputer center. With large datasets, it is impractical, or impossible to download the entire dataset to the workstation and visualize it. However, it has been observed that visualization is a process of data reduction—taking large quantities of data and using visual methods to reduce them to their essential, comprehensible qualities. Each stage of the visualization pipeline can potentially be used to reduce the total size of the data in a progressive/staged fashion. The benefits of data reduction on the distributed pipeline must be balanced against the latencies inherent in remotely located components; thus, an optimal partitioning must consider pipeline distributions that are not necessarily a simple client-server division. Therefore, somewhere between the remote site and local workstation a distributed visualization pipeline must be constructed that is optimally partitioned so as to deliver the highest effective performance to the user.

This type of problem is well suited for a Grid-based approach [12] where distributed computational and storage resources are employed to reduce the data movement to a manageable size. Distributed visualization is hardly a new concept, but in practice, most available examples of distributed workflows offer limited flexibility, target a very narrow range of infrastructure, and employ comparatively static distribution of the computation. Manual resource selection for Virtual Organizations (VOs) with even a modest number of services can quickly become impractical. Even with the simplest of workflows, it is very easy to select a workflow partitioning where the performance of the distributed pipeline is actually worse than simply running a monolithic application in a single location. Unless we develop reliable automated methods to select appropriate visualization pipeline distributions, we stand very little chance of deriving any tangible benefit from the Grid computing infrastructure.

The foundation of an effective distributed application manager is the ability to select appropriate resources and accurately rank the predicted performance of various distribution options. Accurate performance prediction requires accurate performance models of the components that comprise a distributed application *before* they are launched. There is considerable work in performance modeling for numerical simulations that employ iterative mesh-based methods [1], but comparatively little work on selecting appropriate performance models for interactive visualization applications. Prior work in this field has identified many approaches to performance modeling including analytic [8], heuristic/statistical [11], and even history-based methodologies [10], but it is not clear which method is best suited for the unique characteristics of distributed visualization applications. Performance modeling for visualization applications poses a special challenge because performance is extremely dynamic and input dependent. For example, assuming we are trying to model the performance of an isosurface algorithm, even with the same dataset, different isolevels can produce isosurfaces with dramatically different numbers of triangles. Different time steps of the same dataset will result in vastly different quantities of extracted triangles even at the same isolevel. The differences in the number of triangles produced results in radically different performance characteristics. This has a very similar effect on distributed memory parallel algo-

rithms because load-balanced input data can produce results that are so load-imbalanced that they obviate the benefits of parallelism. This is very different from performance analysis of simulation codes and mesh partitioning where the workload remains essentially static through the lifetime of a simulation.

In this paper we present a method for modeling and predicting the performance of visualization stages, and ultimately the entire visualization pipeline, using a composite analytic model. Our approach is targeted at modular component-based visualization workflows similar to VTK and OpenDX; therefore, each stage of the visualization pipeline is handled by a component and each component is modeled individually. These components can be distributed across many machines, even ones with different architectures. First we develop a methodology for modeling the performance of these components on our testbed machines using a minimal number of input attributes. Then we combine the performance predictions for individual components with network performance information to facilitate the prediction of overall pipeline performance for the composite application. Our experimental results show that the actual performance of both the individual components and the overall visualization pipeline agrees with the predicted performance to a high degree of accuracy.

2 Related Work

Modeling of mesh-based parallel scientific applications has garnered the bulk of attention from the performance analysis community. The computational load for these applications typically does not change considerably during execution. Only recently has there been considerable effort to model dynamically adaptive applications in heterogeneous environments [1, 2, 3]. Efforts like the Grid Application Development System (GrADS) [8] have been driven by the highly dynamic, heterogeneous and lossy nature of the Grid infrastructure more than by dynamic resource requirements on the part of the applications. The adaptivity of the application, is mostly confined to discrete events like "contract violations" and their associated job migration for tightly-coupled parallel applications or management task-farming engines that support embarrassingly parallel application scenarios.

Visualization, by contrast, offers a considerably more dynamic and complex resource utilization profile. Small changes in the input parameters to some visualization algorithms can result in huge changes in both execution time and amount of generated data. Consequently optimal partitioning for these pipelines can change dramatically during execution. Therefore, existing work on data-parallel partitioning strategies may not be directly relevant to distributed visualization pipelines. It is our intention to evaluate the applicability of existing performance prediction methods to this application scenario. In this paper, we focus almost exclusively on analytic methods. Future work will compare the efficiency, accuracy, and limitations of the three primary performance prediction techniques: analytic, heuristic/statistical, and historical.

2.1 Models

Statistical/heuristic methods attempt to reduce the size of the performance metric space and complexity of the model by employing statistical correlations. Examples of such a methodology is the Dynamic Statistical Projection Pursuit method of Vetter and Reed [11], and UCSD's trace driven MAPS system [15]. Statistical methods tend to overlap with heuristic techniques like sqmat [14] that rely on tunable microbenchmarks to characterize a given machine for a given set of algorithm techniques. First an algorithm is characterized by its pattern of memory references and computational intensity. Then one uses a simplified code that derives a set of parameters that characterize a given architecture. These parameters are then fed into a statistical model that can predict the performance of the original algorithm on an architecture without actually running it there. Much of this work is highly experimental and still under development.

One can make reasonably accurate predictions of component performance using historical logs of performance information. A good example of a history-based performance prediction method is Rich Wolski's Network Weather Service (NWS) [10] which uses historical patterns in network traffic to predict current network congestion conditions. There is often considerable variability in the accuracy of these predictions, but sometimes the historical model can actually perform better than an analytic model because it can take into account unexpected factors such as the social/behavioral patterns of the people who use the computing infrastructure. However, creating a historical model requires monitoring of real usage patterns. In our situation, we would have to instrument a widely used production code in a completely non-invasive manner to collect real historical data in order to build our model—a difficult proposition at best.

Analytic methods, perhaps the most common and direct technique in performance modeling, attempt to derive an equation that can predict an algorithm's performance using a minimal set of input parameters. Examples include the latency-based model employed for modeling the performance of sparse-matrix kernels [13]. In practice, it is quite difficult to find a minimal set of truly independent parameters—resulting in very complex models. A model with too many parameters can be very difficult to validate as well because the parameter space and dependent performance metric space can be very large. The applicability of the results can be quite architecture-specific, thereby limiting their relevance. However, they offer the most direct approach to developing a predictive performance model for our components. By limiting the scope of our experiments to a simple isosurfacer work-

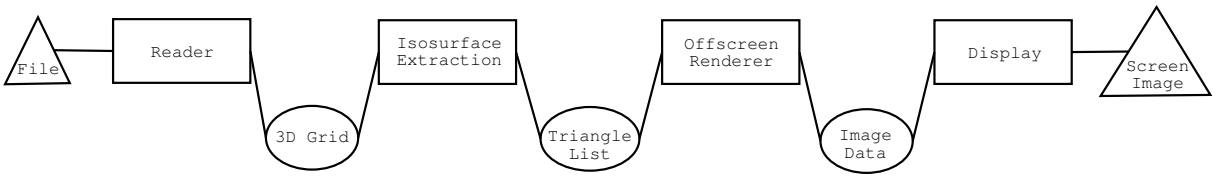

Figure 1: Components and Data flow for visualization pipeline.

flow, we are able to develop an analytic model with the fewest possible degrees of freedom.

2.2 Partitioning

There exist visualization applications that allow the user to distribute the visualization process into a pipeline. The predominant drawback with existing mainstream implementations is that the pipeline partitioning must be done manually. This requires the user to have fairly detailed knowledge of the machines available, in order to know which resource is appropriate to handle a particular component—an impractical requirement for a typical production environment.

Current remote visualization applications typically employ two different partitioning strategies. With one type, all of the visualization computations are handled by the server containing the dataset, and only the image data is sent to the workstation [4]. For the other type, a subset of the dataset or the derived geometry and/or texture data is sent from the server to the workstation, which handles the local visualization [5, 6]. There are two problems with applying this approach to the Grid. One is that the desirability of one pipeline setting over the other may change at run time. The other, more obvious, problem is that with a large number of resources available, it may be impractical to perform the resource selection manually.

3 Application

A popular paradigm for representing visualization workflows is the data-driven dataflow component pipeline. Examples of systems that use this execution paradigm in distributed environments include tools such as AVS, OpenDX, and VTK. For the purpose of this paper, we model the performance of a network-distributed isosurface visualization pipeline. We selected this particular restricted case because of its simplicity, its pervasive use in scientific and engineering applications, and because of its relevance to a variety of similar stencil-based visualization algorithms that produce geometric output. This pipeline has four different stages, which are Reader, Isosurface Extractor, Off-screen Renderer and Display. Figure 2.1 shows such a pipeline. The components are designed such that they can be composed in any topological distribution of network-connected compute resources in a location-independent manner—communicating via the fastest available method. When communicating locally, the components exchange data using a pointer hand-off in the same application space. In the distributed case, data is serialized and sent to the next machine via TCP packets. Associated with the Reader is a 3D grid dataset, associated with the Isosurface Extractor is a Triangle List, and with the Renderer an Image Buffer.

We restrict ourselves to an execution paradigm where each component is activated in sequence along the direction of the forward dataflow dependencies in response to any changes in component inputs. This execution paradigm is typical of visulization workflows. Some visualization applications also support an asynchronous pipelined execution model for out-of-core methods; however, modeling such an execution scenario requires incorporation of queuing theory in addition to the analytic models for each component. Since both execution scenarios must be composed from the performance models of individual components, we have chosen to focus our initial work on the simpler of the two cases. Thus, our results are only applicable to the data-driven sequential execution paradigm, but eventually we will be able to apply the resulting component performance models for future work that employs queuing theory to generalize the performance model enough to accommodate asynchronous systems.

When components are located on the same machine, data is exchanged between the components using a simple pointer hand-off. When the components must communication between different machines, the data is serialized and transferred over the network via a TCP socket and then deserialized at the destination. Assuming all components are distributed among various machines, the component interaction and data flow is as follows. First the Reader opens a dataset file and uses it to initialize a 3D grid dataset. The 3D grid is serialized and its packets are sent to the Isosurface Extractor. The Isosurface Extractor deserializes the 3D grid, and uses it (along with an isovalue) to create the isosurface triangles. These triangles are used to initialize a Triangle List which is serialized as before, and sent to the Renderer. The Renderer renders the triangles in the Triangle List and stores this image to the Image dataset. It then serializes the Image Dataset and sends it to the Display, which simply deserializes and displays the image server data. However, any co-located components will simply pass pointers to data-structures to exchange data

in the most efficient manner possible.

While the objective of our work is to predict the performance of visualization tools that employ data-driven visualization workflows, the actual target systems are far too complex to instrument in an effective and timely manner. Therefore, we developed our own simplified pipeline that provides functionally equivalent operations. The simpler pipeline greatly accelerated our ability to instrument the components and understand the result with a minimum of engineering complexity; however, we are confident that our results can be applied to the more complex applications.

4 Performance Model

Each of the components has a fully parameterized performance model that contain many machine-dependent coefficients. We use a series of benchmarks to approximate the correct values for these coefficients.

Once the performance of the various components can be modeled, additional network information must still be gathered to predict overall pipeline performance. Our technique for collecting this data is discussed below.

We use the following notation. Time is represented by t, a parameter is represented by n, and a constant is represented by a C.

4.1 Reader

The time needed for a file to be read off disk is dominated by the 3D grid size. Hence, the performance model for our reader is simply

$$t_{reader}(n_v) = n_v \times C_{reader}, \tag{1}$$

where $n_v = x \times y \times z$ (x, y and z, represent the dataset dimension, in voxels), and C_{reader} is computed by first simply opening a variety of datasets and recording the time spent for each open. Using the known file sizes, we find an average C_{reader} for each machine.

4.2 Isosurface Extractor

The performance model for isosurface extraction is the most complicated of the component models. We used the Marching Cubes algorithm [7] for our Isosurface Extractor component. There was initially some uncertainty as to whether the isosurface extraction performance would be dominated by the cast-table selection (proportional to the number of cells intersected by the isosurface) or by the number of triangles produced by the case-tables, which can vary from 1-5 triangles. After a series of benchmarks, we determined that the isosurface extraction performance was based on the number of triangles extracted rather than on the number of cells intersected by the surface. This leads to significant difficulty in predicting ahead of time the isosurface extraction performance for a triangle based model. Also, even if no triangles are generated, time is still consumed marching through the dataset inspecting cells. No additional optimization for speeding up cell sending is applied. Hence, there is a base cost, which is determined by the size of the 3D grid dataset. Our performance model is therefore

$$t_{iso}(n_t, n_v) = base(n_v) + n_t \times C_{iso}, \tag{2}$$

where base cost is modeled with $base(n_v) = n_v \times C_{base}$, where C_{base} is computed by first of all using our Isosurface Extractor on datasets of various sizes and using isovalues that do not generate any triangles. We use the times recorded to find an average C_{base} value. We use a similar method to find C_{iso}: We simply record the time spent computing isosurfaces that generate a varying number of triangles. Later we use these times to find an average value for C_{iso}.

4.3 Off-screen Renderer

The off-screen render time is dominated by the number of triangles being rendered. Hence our model is

$$t_{render}(n_t) = n_t \times C_{render} + t_{readback}, \tag{3}$$

where $t_{readback}$ is found by recording the time spent reading the frame buffer after rendering no triangles, and where to find C_{render} we first record the times spent rendering various amounts of triangles, and use those times to find an average value. $t_{readback}$ incorporates both the rasterization time onboard the video card and the cost of reading the framebuffer back into user memory. It can be further parameterized by the image-size, but we will restrict the parameterization in this study to focus on a fixed image size.

4.4 Network

We are interested only in the following three bandwidths, Reader-Isosurfacer (Bri) measured in $bytes/sec$, Isosurfacer-Renderer (Bio) measured in $triangles/sec$, and Renderer-Display (Bod) measured in $pixels/sec$. To measure the network bandwidth between two machines, we have one machine serialize and deserialize a dataset, while the other deserializes and serializes that same dataset. In order to find Bri, we use the above proceedure with a 3D grid dataset. In order to find Bio, we use a Triangle List, and to find Bod we use a triangle list. The network performance model does not account for protocol behavior such as TCP slowstart, loading, or latency effects. While the performance model for the network is not as sophisticated as it could be, it has proven to be sufficient to predict the application performance.

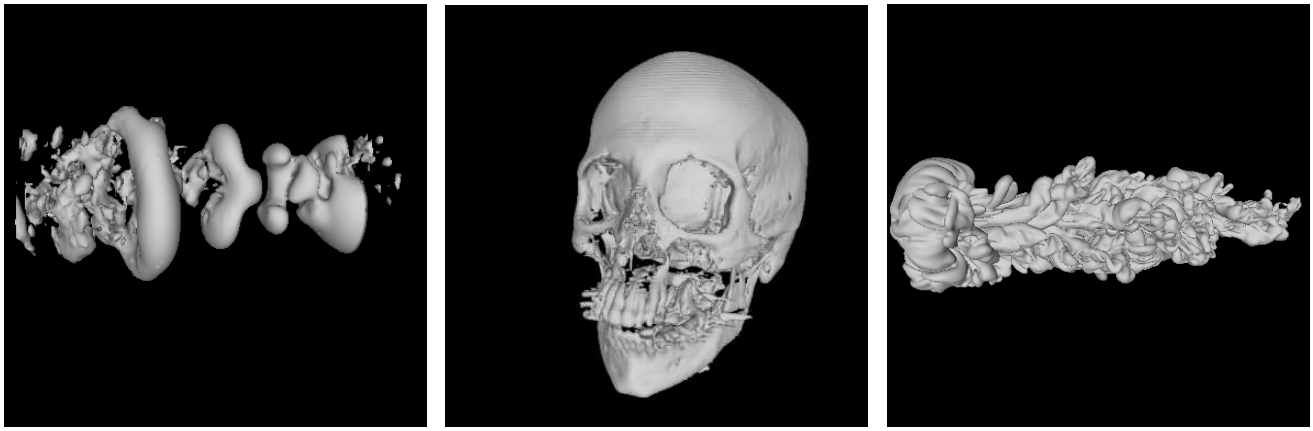

Figure 2: Left: turbulent jet data, $104 \times 129 \times 129$ voxels. Middle: CT head data, $256 \times 256 \times 240$ voxels. Right: Argon bubble data, $640 \times 256 \times 256$ voxels.

5 Performance Model Tests

To test our models we used the following five machines. A single shared node of an IBM SP Nighthawk II running AIX. This node has 16 375MHz Power 3+ processors with 16GB of shared memory. From this point we will refer to this machine as seaborg. We also used a single node of Silicon Graphics Onyx 3400 with 12 600 MHz IP35 processors and 24 GB of memory. We'll refer to this machine as escher. We also used a PC running Linux with 2GB of memory (known as troutlake) that has an Intel Xeon running at 3GHz and a ATI Radeon 9700 graphics card. The other PC (known as millwood) we used has 1.5GB of memory, a Pentium 4 CPU running at 1.9GHz, and an nVidia GeForce 4 graphics card. Finally, xvc is a small PC cluster with 6 nodes each of which has dual 2.4GHz Xeons and a GeForce 5600FX graphics card. Whenever possible, all the available processors are used for parallel isosurface extraction and rendering.

Three datasets were used for our tests. One contains jet flow data with $104 \times 129 \times 129$ voxels. The second is a CT scan of a human head with $256 \times 256 \times 240$ voxels. The third one is an argon bubble dataset with $640 \times 256 \times 256$ voxels. Figure 2 gives sample isosurface renderings of each dataset.

5.1 Reader

The graph in Figure 3 demonstrates that our model for predicting reader performance is adequate. The over-estimation of read time for argon bubbles is likely due to additional efficiencies for continuous reads.

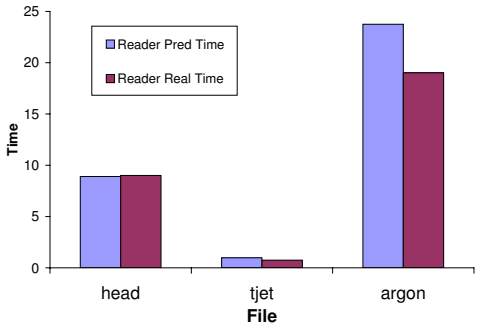

Figure 3: The graph on the left shows the predicted and real times for Reader on troutlake.

5.2 Isosurface Extractor

5.2.1 Analytic Uncertainty

As mentioned in the performance modeling section, the difficult thing about predicting the performance of the Isosurface Extractor is that the performance depends on the number of triangles generated. However, it is very difficult to predict the number of triangles that will be found prior to execution because this is dependent on both the isolevel parameter and the input data characteristics. It is especially important to note that the number of triangles generated are not directly related to dataset size. We are able to model the base isosurface extraction time for each dataset. But, without knowing the number of triangles generated by an isolevel, we only know that the performance will be between the base cost when no surface is found, and the case where every voxel generates the maximum of five triangles. We need to reduce this range of uncertainty.

Our solution is to create a table describing how many cells an isosurface with a certain isovalue will intersect. This can be done at runtime by analyzing the minimum and maximum value of each cell, and incrementing the tables at each value that falls within this region. This allows us to approximate how many cells an isosurface will intersect using a particu-

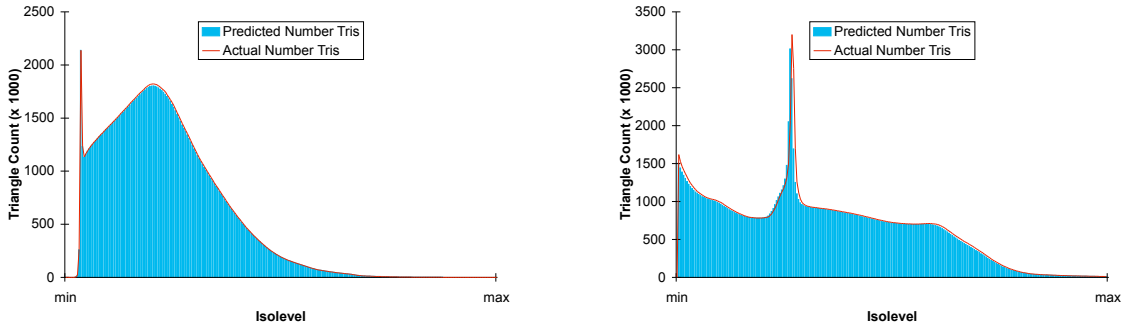

Figure 4: Predicted triangle counts and actual triangle counts for argon bubble(left), and head(right) datasets. Predicted number shown as blue bars and actual number drawn as red curve. 255 samples taken at regular intervals from minimum to maximum. The predicted and actual values are so close that it is hard to differentiate.

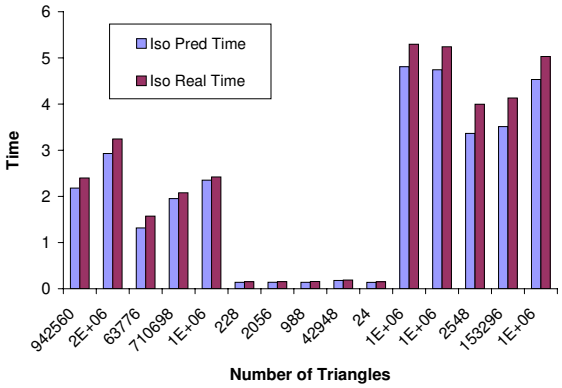

Figure 5: Predicted and real times for isosurface extraction on troutlake.

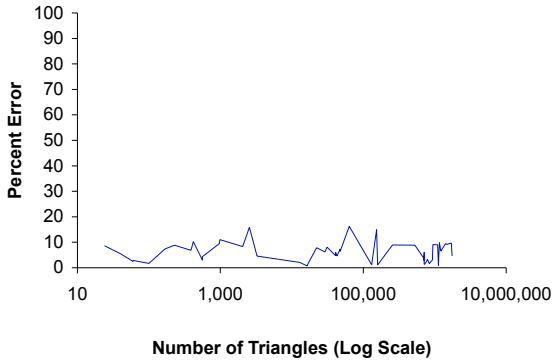

Figure 6: Percentage error of predicted time for the isosurface extraction. Data for graph collected from all machines on which the isosurfacer modeling tests were run.

lar isolevel. The advantage of this technique is that it can be done by the File Reader with little extra cost. However, even when we know how many cells are intersected, we still are not certain how many triangles are generated. That is, assume there are c cells intersected, and 2 triangles per cell on average. Then the approximate number of triangles is $2c$. However, the actual bound on the number of triangles is between c and $4c$, so our uncertainty now lies within this region. We still are left with uncertainty, but it is narrower. Whatever the case, in our results we demonstrate that our technique approximates quite accurately the number of triangles generated by an isolevel, as revealed in Figure 4.

5.2.2 Results

The actual times for the Isosurfacer on troutlake were consistantly longer than the predicted times(Fig. 5). This is likely due to an inaccurate C_{iso} (Eq. 2) measurement for troutlake, since the results from other machines did not share this attribute. Whatever the case, across all machines the average percentage error was 5.95% (Fig. 6).

5.3 Off-screen Renderer

The accuracy of our performance prediction test results for the Off-screen Renderer on troutlake are graphed in Figure 7. Across all machines the average percentage error was 7.27% (Fig. 8). It is difficult to accurately measure rendering times that sometimes take small fractions of a second. The smaller the amount of time being measured, the less accurate the measurement due primarily to timer granularity issues. We will rectify this situation in future work by using CPU hardware performance counters for finer-grained timing of small-scale events.

5.4 Pipeline Performance Prediction

Next we predict the performance for the overall pipeline. We combine networking information gathered as described in

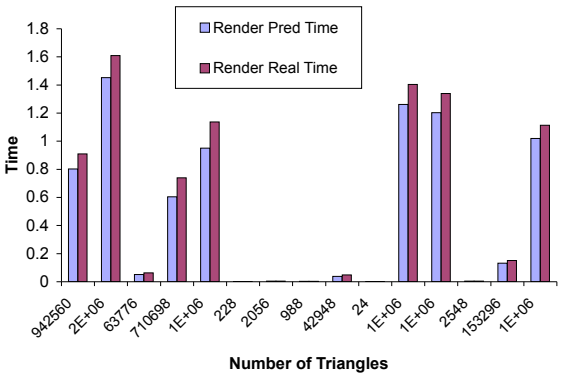

Figure 7: Predicted and real times for off-screen rendering on troutlake.

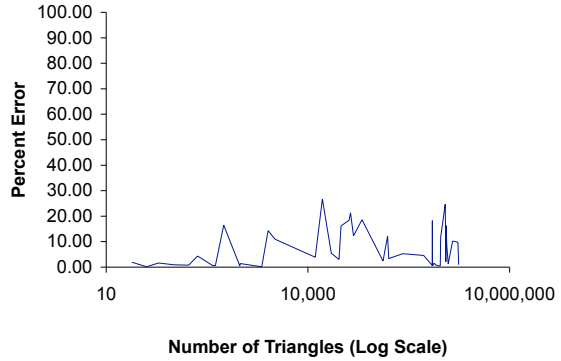

Figure 8: Percentage error of the predicted time for the off-screen renderer. Data for graph collected from all machines on which the off-screen renderer modeling tests were run.

Table 1: Pipeline configurations used for testing.

Pipeline Configurations				
Config	Reader	Isosuf	Renderer	Disp
1	trout	escher	mill	sea
2	escher	mill	trout	sea
3	sea	trout	escher	mill
4	trout	xvc	mill	sea
5	trout	xvc	xvc	sea

Table 2: Predicted times for components. Times for unused combinations omitted.

Predicted Times			
Machine	Reader	Isosuf	Renderer
troutlake	22.70	3.876	.1601
escher	54.17	8.870	19.82
millwood	45.5728	5.783	.04701
seaborg	791.6	371.03	NA
xvc	NA	9.66	1.86

previous sections with our component performance models. Since we assume that local display is fixed, we are only concerned with modeling the performance of the pipeline from the Reader up to deserializing the image data (see Figure 2.1). Therefore, for our tests the machine handling the "display" simply deserialized the image data. Table 1 lists the configurations we used. For all configurations we used the argon bubble dataset and the same isovalue. Also, we used a screen size of 500×500. Thus,

$$n_v = 41,943,040, \quad n_t = 186,854, \text{ and } \quad n_p = 250,000.$$

In order to predict overall pipeline performance we first have to predict performance of the individual components. We used our machine-specific constants to do this and list the results in Table 2. We should point out that none of the pipeline stages was optimized for high performance. For example, both the I/O and cell searching could be optimized through a one-time preprocessing step. However, the methodology is equally applicable to optimized pipeline components. Omitting optimization allows us to concentrate on the larger-scale performance predication methodology. Optimization will become important when we are ready to study the fine-scale matching of different pipeline stages to reach overall high performance.

After we collect the network information we can use the following equation to predict the performance of the pipeline:

$$t_{read} + \frac{n_v}{Bri} + t_{iso} + \frac{n_t}{Bio} + t_{render} + \frac{n_p}{Bod}, \tag{4}$$

Table 3: From top to bottom, predicted and real times for pipeline configurations 1, 2, 3, 4 and 5.

Predicted and Real Times				
Bri	Bio	Bod	Pred	Real
0.50×10^6	0.06×10^6	1.1×10^6	901.9	793.7
1.1×10^6	0.09×10^6	1.1×10^6	71.6	56.2
1.6×10^6	0.06×10^5	0.88×10^6	87.0	83.3
1.6×10^6	0.11×10^6	1.1×10^6	59.22	56.37
1.6×10^6	NA	1.73×10^6	60.34	55.72

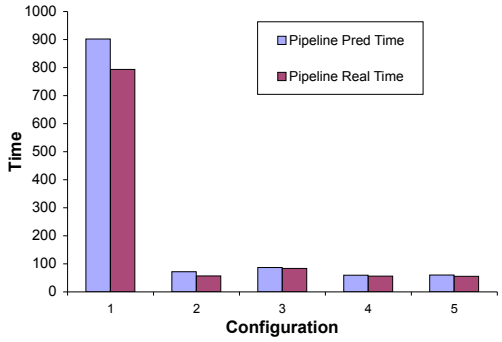

Figure 9: The predicted and real times for visualization pipeline.

where, as introduced in section 4.4, The values for all five configurations are listed in a table in Figure 9, as well as the predicted and real times. Our method correctly ranks the performance of the available pipelines, as can be seen in Table 3 and Figure 9.

6 Conclusions

Large-scale scientific computing is gradually moving toward a Grid-based service model. Thus, appropriate Grid-based visualization tools must be developed to support remote, collaborative data analysis making use of geographically distributed high-performance computing and storage facilities. We have derived and experimentally verified an effective component-based analytic performance model. Such a performance model can be used to automate resource allocation in a Grid-based computing environment. Future work includes studying computational and communication requirements of other visualization methods, parallel visualization pipelines, dynamic repartitioning of visualization pipelines, and other performance prediction techniques. In particular, we will compare this performance modeling technique to the heuristic and historical methodologies. Our ultimate goal is to develop a general purpose framework for managing Grid-based distributed visualization workflows.

Acknowledgments

This research has been sponsored in part by the National Science Foundation under contracts ACI9983641 (PECASE award) and ACI0325934 (ITR); the U.S. Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC program) and NO. B523578 (ASCI VIEWS); and the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The argon bubble dataset was provided by the Center for Computational Science and Engineering at the Lawrence Berkeley National Laboratory. The turbulent jet data set was provided by Dr. Robert Wilson at the University of Iowa.

References

[1] Ripeanu, M., Iamnitchi, A. and Foster, I. Performance Predictions for a Numerical Relativity Package in Grid Environments. International Journal of Scientific Applications, 14 (4).

[2] Gabrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chuang Liu, Thomas Radke, Ed Seidel, and John Shalf. The Cactus Worm: Experiments with dynamic resource discovery and allocation in a grid environment. International Journal of High Performance Computing Applications, 15(4):345–358, 2001. 11

[3] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC11), July 2002.

[4] W. Bethel, J. Shalf. 2003. Grid-Distributed Visualizations Using Connectionless Protocols. IEEE Computer Graphics and Applications. 23(2):51-59.

[5] A. Norton, A. Rockwood. 2003. Enabling View-Dependent Progressive Volume Visualization on the Grid. IEEE Computer Graphics and Applications. 23(2):22-31.

[6] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau. 2000. Using High-Speed WANs and Network Data Caches to Enable Remote and Distributed Visualization. In Proceedings of SC 2000, November 2000.

[7] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. Computer Graphics, 21(4):163–169, 1987.

[8] H. Dail, F. Berman, H. Casanova. 2003. A decoupled scheduling approach to Grid application development environments. Journal of Parallel and Distributed Computing. 63:505-524

[9] R. Raman, M. Livny, M.H. Solomon. 1999. Matchmaking: an extensible framework for distributed resource management. Cluster Computing. 2(2):129-138.

[10] R. Wolski, N.T. Spring, J. Hayes. 1999. The Network Weather Service: a distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems. 15(5-6):757-768.

[11] J.S. Vetter, D.A. Reed. 1999. Managing Performance Analysis with Dynamic Statistical Projection Pursuit. Proceedings of SC 99, Portland, OR.

[12] I. Foster, C. Kesselman(Editors). 1999. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann.

[13] Richard Vuduc, Attila Gyulassy, James W. Demmel, Katherine A. Yelick. Memory Hierarchy Optimizations and Performance Bounds for Sparse A^TAx. 2003. ICCS 2003: Workshop on Parallel Linear Algebra, Melbourne, Australia.

[14] Brian R. Gaeke, Parry Husbands, Xiaoye S. Li, Leonid Oliker, Katherine A. Yelick, Rupak Biswas. 2002. Memory-Intensive Benchmarks: IRAM vs. Cache-Based Machines. International Parallel and Distributed Processing Symposium.

[15] A. Snavely, N. Wolter, L. Carrington. 2001. Modeling Application Performance by Converting Machine Signatures with Application Profiles. IEEE 4th annual Workshop on Workload Characterization, Austin, TX.