A PC-based Grid for Medical Visualization

Yu Kang Xiang-guo Yan Chong-xun Zheng Chi-yu Zheng
The Key Laboratory of Biomedical Information Engineering of Ministry of Education
Xi'an Jiaotong University, Xi'an 710049, China
ky98132@163.com xgyan@mail.xjtu.edu.cn cxzheng@mail.xjtu.edu.cn zcyu180@163.com

Abstract

With grid's potential computing capability and flexibility, many applications needing to run on HPC (High Performance Computing) platform previously can be transplanted on it. To validate the feasibility of a PC-based grid for medical visualization, a hybrid P2P PC-based grid testbed is presented in this paper. The sharp-warp based volume rendering algorithm is adopted for volume rendering. The rendering algorithm is decentralized for the grid. A load balancing method is introduced to improve the rendering efficiency. Two volume data sets are used for experiments. The results show that the computing capability of the P2P mode grid is almost the same as a cluster with similar computing resources. The load balancing mechanism can make the whole visualization procedure more efficient. Finally, to improve the performance further, several possible approaches are addressed for future work.

1. Introduction

With the rapid developments on information technologies and imaging sensor technologies, visualization technique is used as an important mean by doctors and surgeons in hospital. With the help of medical visualization, surgeons can investigate the virtual model of the anatomy of a patient that will greatly benefit the preoperative plans and provide a navigational guidance during surgery [1] and also doctors can make much accurate diagnoses. Normally such visualization procedures require HPC approaches, for example a cluster system will be necessary. So the associated funds and space for a HPC platform will need to be considered. However, widely spreading applications of information technologies in a hospital make personal computers (PC) popular and these PCs are often idle during operating time. All the computing resources could be integrated, the total capability would fully satisfy the needs for visualization. Grid is just such a technology for such demands.

Grid computing has been rapidly spreading as a technical foundation in many areas. A grid integrates all kinds of resources (computing resource, storage resource, network, online hardware, etc.) to satisfy various needs. One of grid's important goals is to play as an internet cluster for science computing. The main idea is called peer-to peer(P2P) [2],[3] technique that is to composite (not just sum) all the resources, especially computing power currently in free time at their owner sides as a whole to serve others. Napster, a website used to let music fans share music files from all over the world, and SETI@home [4], which uses vast available computer power at home and office to analyze radio signals from space, are such systems using P2P. The situation in a hospital is matching the needs for constructing a grid that will not only save funds and space but also provide a convenient way for further integration of different medical information systems.

In order to realize this grid-based visualization system by compositing ordinary PCs of different ownership in a hospital, the P2P mode architecture and parallel visualization technique [5] were used. To validate this intended grid is efficient for substituting the cluster for visualization, we developed a grid testbed. Additionally a cluster (with the same number of nodes) is built for comparing the visualization performance differences between them and we try to prove this grid-based attempt is feasible.

This paper is organized as follows. Section 2 reviews related works of grid-enabled visualization. Section 3 proposes the architecture, volume rendering decentralization, and a load balancing method. The experimental results of the grid based testbed will be presented in Section 4, while Section 5 delivers the discussion of the performance of the PC-based grid and addresses the area of future work. Finally, conclusions are given in Section 6.

2. Related works

There are several works on grid-based visualization. OpenGL Vizserver [6] and VizGrid [7] are grid systems being realized as client-server architecture. In Vizserver, the visualization requests are sent to a remote HPC server to make volume rendering and then send images back with data compression to the clients to decompress and display. VizGrid provides interactive visualization of data. Patras/ITBL [8] grid-based visualization system added more clusters. Users can use a Web browser to login to the system and select one HPC platform and corresponding functions. In these systems all the computing tasks are completed on server's side which makes the number of users limited and requires high-bandwidth data connections between server and clients. Grid visualization system designed by Alan Norton and Alyn Rockwood [9] has a novel improvement to reduce the amount of data communication between server and client. In this system volume data is preprocessed by a server using wavelet data compression technique and stored in a data server. The rendering action is taken place on client's side. When the client needs new data the wavelet-encoded data will be transferred from the data server and decoded at the client side. However, the system still has the limited number of users because the maximum capability of a server or servers is fixed.

Theoretically P2P mode can support more users than client-server mode because every user added means a server participated at the same time. In [10], Taylor, I., Shields, M., Wang, I., and Philp, R, introduce a distribute P2P computing for galaxy visualization test and in [11], C. Zunino and A. Sanna, introduce a rendezvous-edge peer mode for visualization test. Currently we could not find publications that describe P2P mode visualization applications in a hospital with normal PCs of different ownership.

3. Methods

3.1. Architecture Overview

We use a hybrid P2P model (see figure 1) for our grid testbed. There are about six steps to complete one task.

- Data pre-partition: an initial peer, as a task requestor, communicates with information server to get the target data source information (data node location and target data size) and resource information of available peers in the index service. And then using these information to decide how to divide the target data, but not really to divide it;
- Parameter transmission: the initial peer send those parameters, that will tell a peer where to get the target data, what size should a peer be download and how to run the sub-task on the chosen peer;
- Data distribution: every chosen peer downloads the part of the target data according to the parameters received. If user still work on this target data next time, this step will be skipped;
- Sub-task processing: every chosen peer begins running the sub-task until to get an intermediate data;(from this step to the last procedure will be described in more detail in section 3.2)
- Data collection: all the intermediate data is transmitted to the initial peer;
- Intermediate data composition: the initial peer do the final procedure for the intermediate data composition and output the final result.

The grid testbed is built with Globus Toolkit 4.0(GT4) [12],[13] which is an open source software toolkit used for building grid systems and applications. The resource information service is implemented using MDS (the Monitoring and Discovery System). MDS is one of Globus Toolkit components, which is a suite of web services to monitor and discover resources and to provide services on grid. This system allows users to discover what resources are considered and to monitor those resources. The realization pattern is as figure 2. We use Index service, which is a MDS WSRF-based service, and aggregator service as our resource register

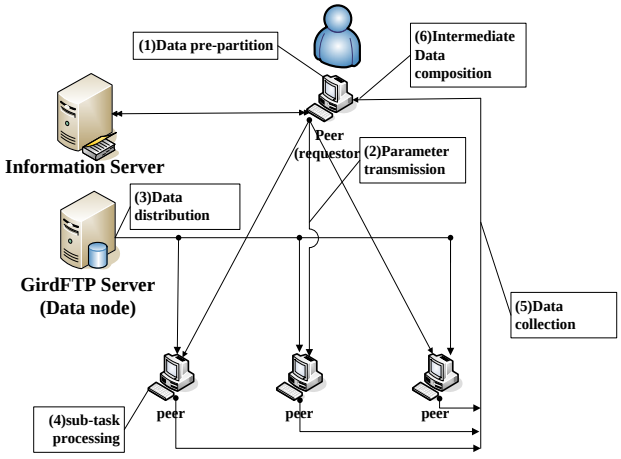

Figure 1. **Hybrid P2P model.**

and data collector to provide users or the job scheduler resource information about every machine available in the grid. The aggregator service collects information via the aggregator source which is a Java class that implements an interface to collect XML-formatted data. An index is self-cleaning that each registration has a lifetime. If a registration is not refreshed before it expires, the associated data and registration itself are

removed from the server. Every machine has its own index service and aggregator source to complete their local resource collection and registry.

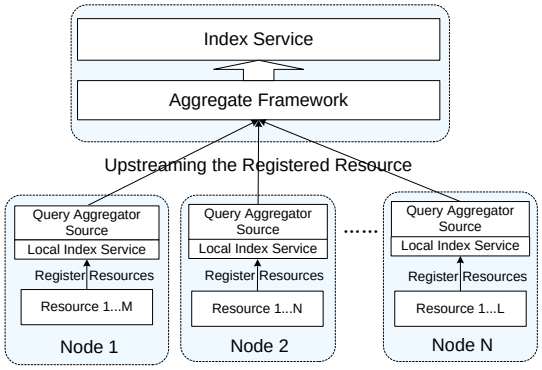

Figure 2. **Resource aggregation pattern.**

For job submission and data transfer, two different systems are used for performance comparison in grid application category. One is GRAM, the execution management service of GT4. The other is a grid visualization service which we call Visual-Grid service, programmed by us based on web service technique. The data transfer function in these two systems are based on Gridftp both.

In this project, the peer(requestor) initiating a visualization request can extract computing resource information from index service and select some of them that fit their application request to submit jobs. According to the nodes' resource information (CPU rate, free memory, and disk space) extracted from index service, the visualization request will be divided into several parts. During tasks distribution, only the parameters of task size, its starting position in the volume data array, and steering message are sent. The distributed computing will transform 3-D volume data into 2-D intermediate image data and sent back to the requesting peer for compositing and generating final image. In this architecture every peer can be a visualization requestor.

3.2. Volume Rendering Decentralization

Volpack is a portable software library for volume rendering written by Philippe Lacroute. Its main idea is based on a factorization of the viewing matrix into a 3D shear parallel to the slices of the volume data, a projection to form a distorted intermediate image, and a 2D warp to produce the final image [14]. The algorithm chooses a coordinate system called "sheared object space" in which all viewing rays are parallel to the third coordinate axis. Mapping from the object coordinate system, the "sheared object space" allows efficient projection to a 2D image. So the algorithm of intermediate image rendering can be described as follows[14]:

```
for z0 = 1 to VolumeDepth
  for yi = 1 to ImageHeight
    for xi = 1 to ImageWidth
      foreach y0 in ResamplingFilter(xi,yi)
        foreach x0 in ResamplingFilter(xi,yi)
  add contribution of Voxel[x0,y0,z0]
toImagePixel[xi,yi]                          (1)
```

At last the 2D intermediate image is transformed into final image. This procedure is less expensive because the data is 2D image which is much smaller than 3D volume data.

We use data parallelism technique here making a large dataset being partitioned into many independent subsets that can be processed in parallel. We apply data parallelism to the procedure when the volume data are projected to a distorted intermediate 2D image. The volume data are divided along z-axis into M parts. So the formulation 1 is modified like this:

```
for z0 = Part_beg(j) to Part_end(j)
  for yi = 1 to ImageHeight
    for xi = 1 to ImageWidth
      foreach y0 in ResamplingFilter(xi,yi)
        foreach x0 in ResamplingFilter(xi,yi)
          add contribution of Voxel[x0,y0,z0] to
              IntermediateImagePixel[xi,yi],     (2)
```

where Part_beg(j) and Part_end(j) represent the jth begin-index and the jth end-index respectively along z-axis, j is from 0 to M-1. The M data subsets will be processed in different nodes. The "over operation" [14] is used for volume rendering. The "over operation" is written as follows:

$$\begin{aligned} L(x) &= \sum_{i=0}^{n-1} c_i * \prod_{j=0}^{i-1} 1-\alpha_j \\ &= c_0 + c_1(1-\alpha_0) + c_2(1-\alpha_0)(1-\alpha_1) + \ldots \\ &+ c_{n-1}(1-\alpha_0)\ldots(1-\alpha_{n-2}) \\ &= c_0 \; over \; c_1 \; over \; c_2 \; over \ldots over \; c_{n-1} \end{aligned} \quad (3).$$

α_i is the opacity of sampling i. If $color_i$ is defined as the color of sampling i, c_i is the premultiplied color which is equal to α_i multiplying $color_i$. $L(x)$ is the combination result alone a ray. This equation is used in two stages. Every node uses it for volume rendering operation to create an intermediate image data and then these intermediate image data are collected to a node on which this equation is reused for intermediate image composition. Finally the composite image is transformed into final image.

3.3. Load Balancing

Generally the performance of the nodes in a grid is different and their resources available are time-varying. A Load balancing method is used in our Visual-Grid service to improve the visualization efficiency. This load balancing schema is used at "data pre-partition" stage (see section 3.1). We define

$$C_i(t) = R(t)CPU^i_{idle} \quad (4)$$

as the rendering capability of the *ith* node on the time t. In Equation (4):

$$CPU^i_{idle} = (1 - CPU^i_{usage})cpuspeed_i$$

where CPU^i_{usage} is the mean CPU usage rate in 1 minute on node *i* and the sampling rate is one second. The $cpuspeed_i$ is the speed of CPU computing float point per second on node *i*. $R(t)$ is the weight of CPU^i_{idle}. $R(t)$ will be changed according to history record, which will be described later. A visualization job will be divided according to $C_i(t)$. The percentage of sub-job to the total on every node is

$$p_i(t) = \frac{C_i(t)}{\sum_{i=1}^{i=n(t)} C_i(t)}, \quad (5)$$

where *n(t)* is the number of nodes currently chosen to do a visualization job. We define

$$MaxMemory_i = FreeMemory_i - 3S_i$$

as the maximum free physical memory can be used on node *i*. $FreeMemory_i$ is the mean value of the size of free physical memory and S_i is the sample standard deviation of memory usage on node *i* in 1 minute and the sampling rate is one second. When the target data distributed according to $p_i(t)$, three situations will occur:

- If the memory needed for target data on every node dose not exceed $MaxMemory_i$, the nodes will begin to download the data;
- If $MaxMemory_i$ of some nodes, assuming *j* nodes, do not satisfy the memory need, the total memory needed size of the job, $Totalmemory(t)$, will cut off $\sum_{i=1}^{i=j} MaxMemory_i$ and use Equation (5) to re-compute the portion among the rest nodes until all nodes satisfy the memory need. Otherwise the finally remained data will again be divided among all the chosen nodes according to Equation (5);
- If the memory needs on all the chosen nodes exceed the available physical memory, the nodes will begin to download the data and the virtual memory on the node may be used.

We also define

$$K_i(t) = \frac{RT_i(t)}{p_i(t)Total(t)} \quad (6)$$

where $RT_i(t)$ is the rendering time used on node *i* in the time t. $Total(t)$ is the total size of the job in the time t. Every time the total size of a job submitted may be different. We use $K_i(t)$ to adjust $R(t)$.

$$R_1(t) = \begin{cases} (1-\alpha)R_1(t-1), if \\ |K_i(t-1) - K_i(t-2)| > THR \cap K_i(t-2) < K_i(t-1) \\ (1+\alpha)R_1(t-1), if \\ |K_i(t-1) - K_i(t-2)| > THR \cap K_i(t-2) > K_i(t-1) \end{cases} \quad (7)$$

Where THR is a threshold, α ($0 < \alpha < 1$) is the adjusting coefficients for $R(t)$.

$|K_i(t-1) - K_i(t-2)| > THR \cap K_i(t-2) < K_i(t-1)$ means that the load on the node i is heavy. So the $R(t)$ is decreased, which causes $C_i(t)$ decreased. And smaller sub-job will be distributed on node i.

$|K_i(t-1) - K_i(t-2)| > THR \cap K_i(t-2) > K_i(t-1)$ means that the load on the node *i* is light. So the $R(t)$ is increased, which causes $C_i(t)$ increased. And bigger sub-job will be distributed on node *i*.

When a user wants a visualization for a certain image set, he often works on a image set with rather long time, which will result in many continually task submissions for a same data set. So this load balancing schema is used for this situation and α will be decreased to zero when the time interval between two submissions is rather long.

3.4. Security

Security is a primary concern in the context of any grid computing applications. The security of the PC-based grid presented covers not only system itself but also the privacy of patients. So the system security involves privacy of patients, authentication, communication privacy and integrity, and authorization.

Privacy of patients is guaranteed by the anonymous data mechanism. Before the data distributed among the nodes, all the privacy related information such as name, address is shield off, and that having someone eavesdrop on a communication does not pose a privacy risk.

GRAM uses GSI Transport schema for system security.

For our Visual-Grid service, considering that a user may work with the same data set repeatedly for certain time interval, such that a doctor may want to observe a 3D image of a patient from different angles of view. So the GSI secure conversation schema is used as the authentication method. This method establishes a context after an initial exchange of messages. All the following messages can reuse that context, resulting in

a better performance. Public-key cryptography based on asymmetric algorithm makes the authentication easily for a user with private-key and almost impossible for an unauthenticated one. And digital signature is one part of public-key system. It is a piece of data which is attached to a message and could be used to find out if the message is tampered with during a conversation for data integrity verification.

Because every node used here has the ownership corresponding to a specified person, a doctor for example, the node must verify that only those authorized ones can use the local visualization service to run other sub-jobs. Currently the Grid-map authorization is used, and only the users added in the grid map file are the authorized ones who can use the visualization service deployed on this node.

4. Experimental results

The cluster and the two kinds of P2P PC-based grid services in the experiment were made up of four 256M RAM 2.80GHz-Pentium4 PCs, and a Dell Precision 650Xeon 2.8GHz 1024M workstation. So the test environment has nodes with different computing capacities.

The implementation of the cluster is distributed-memory mode based on MPICH1.2.7. The nodes are connected with 100Mbit/s Ethernet. The cluster uses SSH (Secure Shell) protocol.

Three kinds of testing method are used in this test. The first is MPI-cluster test. The second is the grid test using GRAM to submit visualization task. The third method is our Visual-Grid service. We have two ways of data distribution. One is mean data distribution that the data is divided equally among nodes. The other is guided by the load balancing schema.

Because a user often works on a certain image set rather long time, the divided data on every node for the sub-job will only distributed in the first time and it will be reused until the user changes to the other data set. So in our experiments the volume data distributing time are not counted. RT_i is the rendering time on node *i*. T_{ij} is the intermediate image data transmission time for data collection from node *i* to node *j*. In the experiment once a user define the final image size (256x256 etc.), the size of intermediate image data will be the same. We assume that bandwidth among nodes is same and all the intermediate image data is collected to one node. In the experiment, the node for final composition and coordinate transformation is always the same one. So the T_{ij} will be a same value, defined as a constant TR when a user submits a visualization job. CT_i is the intermediate images composition and coordinate transformation time on node *i*. So all the figures below show the time:

$$FT = MAX(RT_i) + TR + CT_i \qquad (8)$$

The volume data used are two sets. One is 384*384*252 voxels and the other is 512*512*360 voxels, which all derived from the data set named "brainsmall" in VolPack using upsampling method.

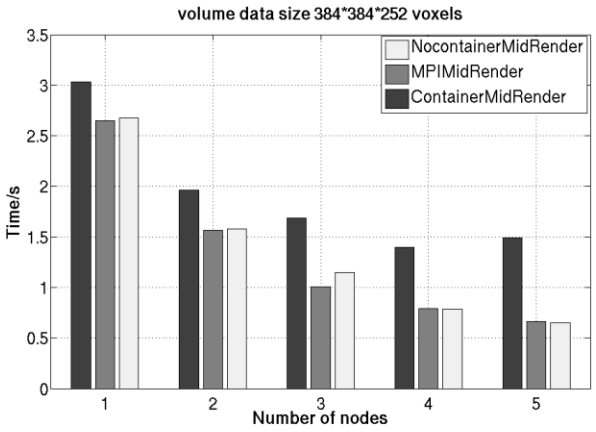

Figure 3. **Time consumption bar plot for 384*384*252 volume data set.**

Three kinds of result bars are displayed in the figure 3 without using load balancing schema. Every node is distributed with one part of jobs evenly. When the number of nodes for visualization is one, the workstation is not chosen here. "MPIMidrender" bar is the time FT in MPI-cluster test and the rest are ones in gird test. "ContainerMidRender" bar means the time FT of visualization managed by GRAM. "NocontainerMidRender" bar is the time FT of visualization managed by our job submission system.

GRAM was used firstly for job submission. But we found that the rendering time increased severely on the nodes with 256M memory while involved in the 512*512*360 volume data test. There are two reasons to bring on this result. One is because GRAM needs more memory consummation during a job execution for job state monitoring than our self-make job submission service. The other is large data loading. These two reasons cause insufficiently available memory and influence the volume rendering procedure. So we only display the test result of the 384*384*252 voxel data set in figure 3. From figure 3, it can be seen that the rendering performance is approximately equal between the cluster and grid with the simple job submission system. And it will take more time when using GRAM for the more memory consumption than the simple one during job processing.

In figure 4, the GRAM was not used for job submission and execution due to insufficiently available memory mentioned above. Comparing

"MPIBigrender" bars with "NocontainerBigRender" ones using mean date distribution schema, we can also find out a similar result as in figure 3. To simulate resources unbalancedness in a PC-based grid, two kinds of PCs are utilized when the number of nodes is 5. The "BalanceNocontainerRender" bar shows that a load balancing strategy adopted can improve the performance effectively.

Figure 5 shows the composition procedure when visualization is decentralized into 5 parts. The first five images are intermediate images created by each node or peer and the last one is the final image.

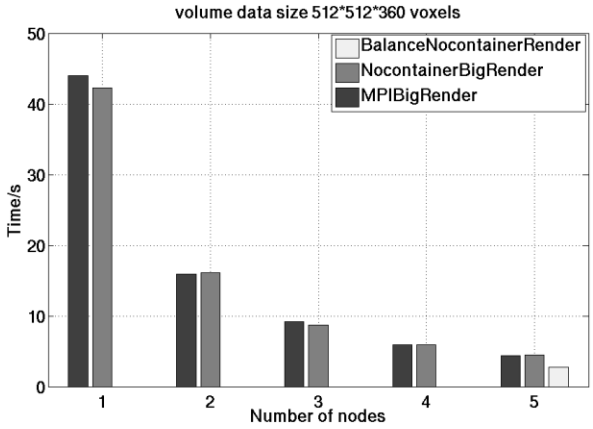

Figure 4. **Time consumption bar plot for 512*512*360 volume data set.**

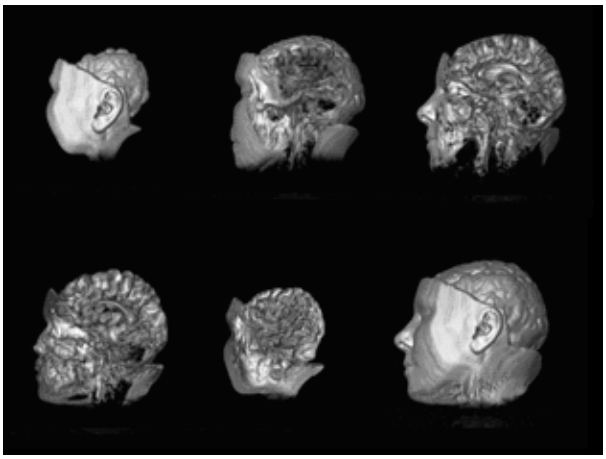

Figure 5. **Composition procedure of 5 intermediate images and final image**

5. Discussions

The experimental results demonstrate that the performance on a P2P PC-based grid is almost the same as a cluster with similar computing resources for our visualization application. The sharp-warp volume rendering algorithm presented by Philippe Lacroute et al is an effective volume rendering method and can be modified for a grid application. Considering that free resources on each PC are time-varying, load balancing is a good choice to achieve better performance.

From the experimental results, we also find that there are several aspects that can make the system much efficiency except those usually considered (such as data transmission speed, decentralizing task and related resources, data compression, etc.):

Predict performance. Because every PC is autonomous, we could not grasp the amount of free computing resources on each peer in the next duration of time even though we get the current states from the information server. In order to achieve the most efficient services as quickly as possible, we can use a performance predicting mechanism that will guide us for fast searching candidate resources based on the analysis of historical records.

Reduce link establishment time. With the number of nodes increasing, one complex function may involve many PCs. Hardware and software conditions may cause an asynchronous job preparation which will influence the whole system efficiency. So reducing this time is very important for improving system performance. We try to reduce this delay in our job submission system in the experiment above. The job submission delay is about 2 seconds in our system.

Keep connected channel during a request processing. This aspect is aiming at minimizing the frequency of link establishment. Owing to various computing resources are hidden behind firewalls with all kinds of rules, keeping connection can also reduce the time of package filtering by firewalls. When visualization especially for a visualization session begins, the peers will keep their connective status until users exit. So the link establishment only occurs at job preparation stage and after that the performance are only related with processing capability and transmission speed.

6. Conclusions

In this paper a hybrid P2P PC-based grid testbed for medical visualization is presented. The grid testbed is established with Globus Toolkit. The resource information service is implemented using MDS4. Index service and aggregator service are used as resource register and data collector to provide users or the job scheduler resource information about each machine available in the grid. The aggregator service collects information via the aggregator source. Privacy, authentication, and data integrity are guaranteed by the anonymous data mechanism, public key cryptography,

and digital signature respectively.

We modify the sharp-warp volume rendering algorithm for parallel visualization. The experimental results show that the computing capability of a P2P PC-based grid is almost the same as a cluster with similar computing resources for our visualization application. Considering the condition of free resources on each PC being time-varying and the nodes' performance being different, the load balancing mechanism can make the whole visualization procedure more efficient even the load balancing method is relatively simple.

7. Acknowledgment

The authors would like to thank Stanford computer graphics laboratory for providing the VolPack and volume test sets.

8. Reference

[1] B. Ma and R. E. Ellis, "Robust registration for computer-integrated orthopedic surgery: Laboratory validation and clinical experience," Med Image Anal., vol. 7, no. 3, Sept. 2003, pp. 237-250.

[2] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu., "Peer-to-peer computing", Technical Report HPL-2002-57, HP Lab, 2002

[3] Shirky, C. (2001). "What is P2P... and what Isn't," An article published on O'Reilly Network. Availible: *www.openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html*.

[4] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and dan werthimer , "SETI@home An Experiment in Public-Resource Computing", COMMUNICATIONS OF THE ACM, vol. 45, no. 11, Nov. 2002, pp. 56-61

[5] J. Ahrens et al., "Large Scale Data Visualization Using Parallel Data Streaming," IEEE Computer Graphics and Applications, vol. 21, no. 4, July/Aug. 2001, pp. 34-41.

[6] Ohazama, C., "OpenGL Vizserver White Paper", Silicon Graphics Inc., USA, CA, 94043 (650) 960-1980, 1999

[7] Matsukura R, Koyamada K, Tan Y, et al., "VizGrid: Collaborative Visualization Grid Environment for Natural Interaction Between Remote Researchers", Fujistu Scientific &Technical Journal, vol.40, no. 2, 2004, pp. 205-216.

[8] Suzuki, Y., Sai, K., Matsumoto, N., and Hazama, O., "Visualization systems on the information-technology-based laboratory", IEEE Computer Graphics and Applications, vol. 23, no. 2, 2003, pp. 32-39.

[9] Alan Norton, Alyn Rockwood., "Enabling View-Dependent Progressive Volume Visualization on the Grid", IEEE Computer Graphics and Applications, vol. 23, no. 2, 2003, pp. 22-31.

[10] Taylor, I., Shields, M., Wang, I., and Philp, R., "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case," IPDPS 2003 Conference, Apr 2003.

[11] C. Zunino and A. Sanna, "A JXTA-based architecture for 3D distributed visualization: D3D," Available: *http://sanna.polito.it/Versioni_Postscript/CCCT_2004.pdf*.

[12] The Global Grid Forum. *http://www.gridforum.org*.

[13] The Globus Project. *http://www.globus.org*.

[14] Philippe G. Lacroute, "Fast volumerendering using a shear-warp factorization of the viewing transformation", Tech. Rep. CSL-TR-95-678, Sep. 1995.