An Implementation of Sort-Last Volume Rendering over an Optical Network Backplane

Yasuhiro Kirihata

Department of Computer Science

University of Illinois at Chicago

1. Introduction

Recently, as the price of the PC is going down, constructing a cluster system with off-the-shelf PCs to realize the high-performance computing is one of the most focused technologies in the parallel computing paradigm. On the cluster system, each node is connected to each other by the high-speed communication network with the bandwidth reaching 1~10 Gbps. The cluster system takes the MIMD (Multiple Instruction stream Multiple Data stream) architecture on which each node can deal with its own data set on its own memory and execute the programs in parallel. Comparing with the SIMD (Single Instruction stream Multiple Data stream) architecture, it is scalable in low cost and utilization of computing resource is more efficient.

When we consider the efficiency of the parallel algorithm over the cluster, we should take one of the major overheads into account, i.e., communication overhead among processing elements. To minimize the communication cost, we need to (1) communicate in bulk, (2) minimize the size of transferred data, and (3) minimize the distance of data transfer. The first and second points are for minimizing the start up time and transmission time, respectively. The third point depends on the topology of the cluster system and the mapping of the parallel programs. Since the propagation time over the medium of the network is usually very small on the cluster, we can ignore the distance among the nodes. We need to take (1) and (2) to optimize the communication among the processing elements. If the communication data becomes larger such as that on the multimedia application, it is not easy to realize (1) and (2) at the same time, because if we would like to send data in bulk, the size of each message become large, and if the size of each message is small, we need to send messages more often. Minimizing the transmission time with the huge bandwidth network is the most effective way to realize both of them together. Therefore, constructing the cluster over the high bandwidth network such as the optical network is the one of the effective solutions to handle the large data set on the cluster.

The OptIPuter [1] is the project which is progressed in Electric Visualization Laboratory (EVL). It is a computing model which uses the optical network as the backplane to connect the computer peripherals each other and provides the high-level computing model. On the scheme, computer peripherals such as computing system, memory system, data storage system, visualization system, and display system are connected together with the optical network and corroborate to realize the high performance computing. The Gigabit Ethernet switch which supports the optical fiber connection changes the optical signal into the electrical signal to realize the packet switching internally in the conventional way. Although the achievable bandwidth of the optical fiber is over 50 Tbps, the practical limitation of the throughput is about 1~10 Gbps through the Gigabit Ethernet

Kirihata, Y., "An Implementation of Sort-Last Volume Rendering over an Optical Network Backplane", Electronic Visualization Laboratory Technical Report: 20040411_kirihata, MS Project, University of Illinois at Chicago, April 2004.

switch. This is due to the response time of a photodiode. The typical photodiode converts one signal in 1 nsec, that is, the limit of data rate is about 1 Gbps. The limitation of the signal sampling causes the limitation of the traditional optical electrical network switch. Meanwhile, the optical switch adopts totally different architecture for switching network. It uses the all-optical MEMS devices to switch the connection inside. The optical signal incoming via the inbound fiber is routed to the outbound fiber with the micro-mirrors and lenses in the silicon. There are no signal changes from optical to electrical. This technology can avoid the bottleneck of the optical-electrical converting signal and make utilize of the optical fiber's bandwidth possible up to the upper limitation. The advantage of the optical switch based cluster is that the bandwidth of the interconnection among the cluster nodes could be over 1000 times larger than the traditional Gigabit Ethernet cluster.

However, there is a serious drawback on the optical switch based cluster. The switching delay takes about 1~2 second. If the cluster nodes changes the connection each other frequently, this drawback affects the performance of the parallel computation. We can expect the performance improvement of the parallel computation when the connection among the cluster nodes does not change frequently comparing with the processing time for the assigned task on each node. Especially, the parallel algorithm in which the data flow is static among the nodes and going to the single node like a tree-structured connection, the switching does not happen after the initial connection establishment. We can hide the drawback and get the benefit of the optical switch based cluster.

The volume rendering is a method to visualize the volume data which is sampled by CT (Computer Tomography) or MRI (Magnetic Resonance Imaging) scanner. The sampled data has the scalar value for each point in the 3 dimensional spatial data. Several methods are proposed to visualize the volume data. The representative methods are ray casting, splatting, shear-warp and hardware-assisted 3D texture mapping. Ray casting is the algorithm calculating the projected ray affected and attenuated through the volume data. The quality of the calculated image is best in these methods but takes much processing time. On the other hand, the hardware-assisted 3D texture mapping method is simple to implement. Rendering procedure is at first slicing the volume data and mapping the sliced image on the polygon. The polygons are located in an array order and opacity is added to combine the texture-mapped volume slices. In these methods, shear-warp and hardware-assisted 3D texture mapping are faster rendering algorithm than others [5]. Especially, hardware-assisted 3D texture mapping is the simple and easy to apply the sort-last rendering. Thus, we adapted the hardware assisted 3D texture mapping method to visualize the volume data on the parallel system.

In this paper, we discuss the sort-last volume rendering system on the optical switch based cluster and describe the implementation of the system. The contents are as follows; first of all, we explain the sort-last rendering and its application to the volume rendering in Section 2. We discuss the design and implementation of our system in Section 3. Finally, we will provide the experimental results and analysis of the system in Section 4.

2. Sort-Last Rendering

2.1. Overview of the Sort-Last Rendering

There are three well-known parallel rendering algorithms, sort-first, sort-middle, and sort-last rendering. Their differences are characterized by the time when the primitives are distributed to several processors in the graphic pipeline. The following figure illustrates the taxonomy of the parallel rendering architecture.

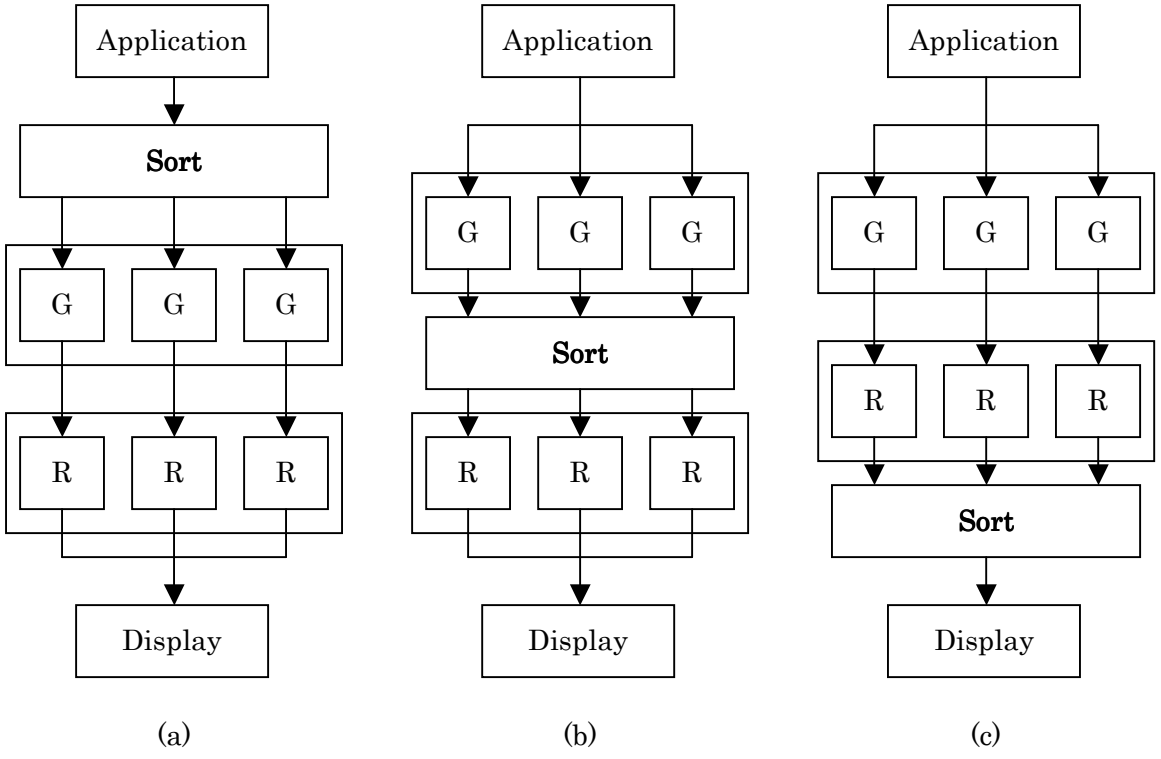

Figure 2.1. The taxonomy of the parallel rendering architecture.
(a) Sort-First (b) Sort-Middle (c) Sort-Last.

The graphic pipeline has three stages, the application processing, the geometry processing, and the rasterization. At the geometry processing stage, each geometry unit G processes the geometry to be rendered. At the rasterization stage, each rasterizer unit R handles the pixel calculations. In the sort-first rendering, the "raw" primitives are distributed early to each processor during the geometry processing stage. Each processor is assigned to a part of the entire display which is divided into disjoint regions, and it renders the assigned primitives individually.

In the sort-middle rendering, the distribution of the work is arbitrary and even among the geometry units. Each rasterizer unit is responsible for a screen space region. After the geometry processing, each primitive is allocated to the corresponding rasterizer unit that is responsible for the screen space location of the primitive.

The sort-last rendering, on the other hand, defers sorting primitives until the end of the rendering pipeline, i.e.

after primitives have been rasterized into pixels. Each processor is assigned a subset of primitives and renders them no matter where they locate on the screen. After rendering, processors communicate each other to composite those pixels to generate the final entire image. In order to handle the real-time high quality image rendering, the high data rates over the internetwork among the rendering processors is required. This is one of the reasons why we target on the parallel computing system over the high bandwidth optical network.

There are some techniques to optimize the data transfer in the sort-last rendering. One is the bounding rectangle method. It is also called SL-sparse. It minimizes the data transfer by only sending the pixels with actual data (active pixels). In order to encode the active pixels, (1) you find a smallest rectangle which contains all actual pixels in the rendered image, (2) take coordinates of upper left and lower right points, and (3) pack these coordinates and the image data inside the rectangle as the buffer to send. When the original image is sparse, the optimization is done efficiently.

At the composition stage of the sort-last rendering, because the composition of active pixel and non-active pixel is the active pixel, we should only compose the overlapping region of two rectangles. This composition technique reduces the time to compose two images.

Another optimization technique is the run-length encoding method. In the method, each pixel is classified into two kinds of pixels, active pixel and non-active pixel. Counting the continuously locating non-active pixels and encode the count as the integer into the sending buffer, the total size of pixels shrinks. Combining these two methods, the sort-last rendering system can optimize the data transfer rate and improve its performance.

2.2 Sort-Last Rendering for 3D Texture Mapping Method

The 3D texture-mapping method is one of the representative volume rendering methods proposed by Cabral. It can achieve a good performance comparing with other rendering methods. The method is executed along with the following step. First of all, the volume data is sliced and loaded as a set of textures into the memory. The hardware rasterizes the sliced texture-mapped polygons parallel to the view plane. The texture-mapped polygons are blended front to back to generate the volume image.

The application of sort-last rendering algorithm to the 3D texture-mapping method is as follows. At first, it divides the volume data into even size of several parts, and then allocates them to rendering nodes. The number of parts is equal to the number of rendering nodes. Then, each rendering node renders the assigned part of the volume data. The number of polygons is almost same on every node because the assigned part of volume data has almost same size in each rendering node. After rendering the assigned part, each rendering node sends the output image to the composition node. The composition node composes the parts of the volume image and output the complete one.

3. System Design and Implementation

3.1. Photonic Computing Environment

The Photonic Computing Environment provides the high performance computing mechanism over the optical switch based cluster system. It constructs the pipelines among the cluster nodes and manages the computation flow. In order to use the optical switch to construct the rendering cluster, the cluster application needs to use the Photonic Domain Controller (PDC) [2] to generate the pipeline connection among the cluster nodes. Because the current library for parallel programming such as MPICH does not support the manipulation of the connection inside the optical switch, it is necessary to implement the network application which generates the network pipeline among the cluster nodes over the optical switch. The following figure shows the architecture of the network application to construct the cluster over the optical switch.

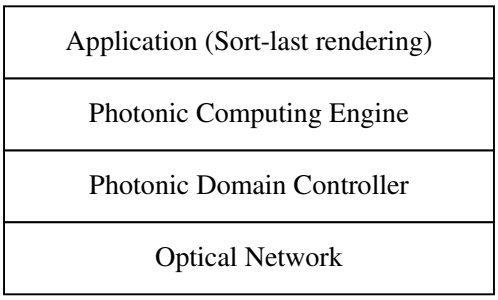

Figure 3.1. The architecture of Photonic Computing Environment

The PDC provides the interface to create the link inside the optical switch. The network application that uses the PDC at first invokes the PDC's interface to establish the connection between two nodes. After generating the connection, those nodes can communicate each other with any protocols such as TCP and UDP. The connection can occupy the whole bandwidth allocated at the initialization time. It is disconnected when the communicating node explicitly invoke the disconnect function on the PDC. The Photonic Computing Engine handles the establishment of the connection among the cluster nodes and provides the functionality to synchronize the messages to the adaptive calculation module such as image rendering module and image composition module.

The Photonic Computing Engine has the two types of data transfer mechanism, pull-up mode and push-out mode. In the pull-up mode, the client send a request to the Photonic Computing Engine and it returns the results as the C/S system. In the sort-last rendering case, the viewer on the client send rendering request to the Photonic Computing Engine each time when it needs to change the view. On the other hand, the outputs of calculations on the Photonic Computing Engine are generated as much as possible and sent to the client in the push-out mode. The push-out mode is useful if the computation results are automatically generated like animations and movies.

3.2. Architecture of the Sort-Last Rendering System over PCE

The Photonic Computing Engine is the application that provides the network pipeline among the optical switch based cluster nodes and synchronization mechanism to realize the sort-last rendering. The following figure shows the architecture of the Photonic Computing Engine with 7 nodes for the sort-last rendering.

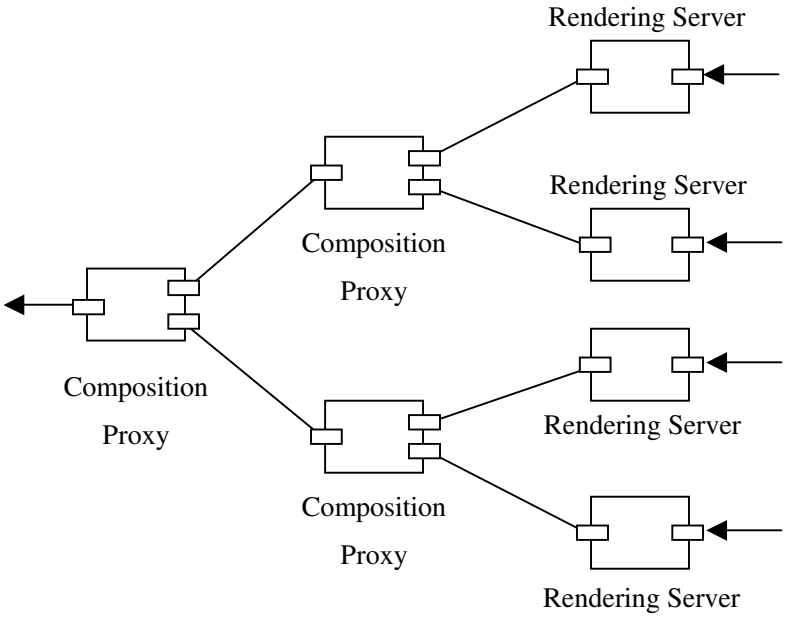

Figure 3.2. Architecture of the 7 node sort-last rendering cluster

On the each node, the Photonic Computing Unit (PCU) is running and generates the pipeline. The client application accesses to the root PCU to get the computing result. In the sort-last volume rendering system, the PCU takes two types of roles, the composition proxy and the rendering server. Each rendering server fetches the allocated part of volume data and renders the image. After rendering the image, the rendering server sends to the composition proxy, which is a parent node of it. At the composition proxy, it synchronizes the output images and composes them.

3.3. Implementation of the Photonic Computing Engine

In this section, we will describe the actual implementation of the Photonic Computing Engine. Photonic Computing Engine has basically the following functionalities, message transferring, message queuing, flow control, and module add-in.

(1) Message transferring

The Photonic Computing Engine generates the cluster as a tree. When the client sends the request to the Photonic Computing Engine, the root node has to propagate the request message to the computing nodes such as the rendering servers. Since switching the connection among the nodes in the optical switch takes much cost,

the Photonic Computing Engine does not change the connection pattern. The message needs to be passed along the tree connection. Therefore, the each PCU has the message transferring mechanism from the parent node to the child nodes.

(2) Message queuing

On the intermediate PCU, the synchronization mechanism is required because the intermediate PCU might use the both results sent from two child PCUs. Each message sent from the child PCUs has a sequential number and it is used to synchronize the output results. Since the output messages from the child PCUs are sent to the intermediate PCU asynchronously, it needs to store the messages in a queue to synchronize them.

(3) Flow control

In the push-out mode, the rendering server sends output image to the composition proxy. If the output message rate of the rendering server is better than that of the other rendering servers or ability of message processing at the composition proxy, the queue could overflow for the message burst. Therefore, the flow control mechanism is required in the composition proxy. In order to control the flow, we use the socket buffer and TCP flow control mechanism. If the socket buffer is full, the sender process is blocked on a TCP connection. Thus, if the length of the queue becomes maximum, the composition proxy blocks the receiving process until a queue element is consumed by another process. The blocking of the receiving process on the proxy is propagated to the child node and stop sending data.

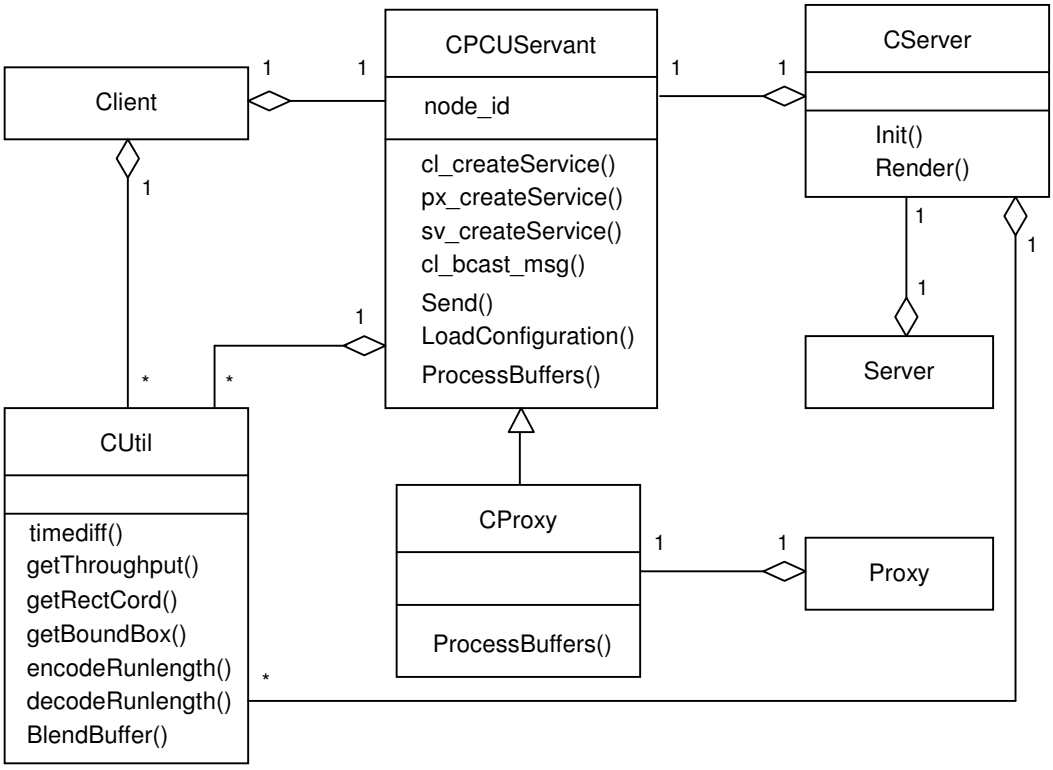

Figure 3.3. Class diagram of the PCE

(4) Module add-in

Besides the message routing mechanism, the Photonic Computing Engine provides the computation add-in mechanism, that is, you can replace the composition and rendering part of implementation to other one like the add-in module. You can easily change the computation algorithm on the Photonic Computing Engine by overwriting the computation part of composition nodes and rendering nodes. For example, the proxy provides the callback function that is invoked when all output data from child nodes reach at the proxy. You can overwrite the callback function that handles the output buffers to implement other composition algorithms. Also, the rendering server provides the display function as the callback function. If you would like to implement the other rendering algorithms, you can modify the display function to realize it. The class diagram of the PCE is depicted in the Figure 3.3. You can overwrite CServer::Render() and CPCUServant::ProcessBuffers() to implement other composition and rendering algorithms.

We explain the implementation of the rendering server, composition proxy, and client viewer. The rendering server renders the part of the volume data with the 3D texture mapping method. After rendering the assigned part of volume data, it fetches the image data from the frame buffer. The fetched image is cut into the smallest rectangle which includes the active part of the image, encoded by the Run Length Encoding algorithm, and sent to the composition proxy.

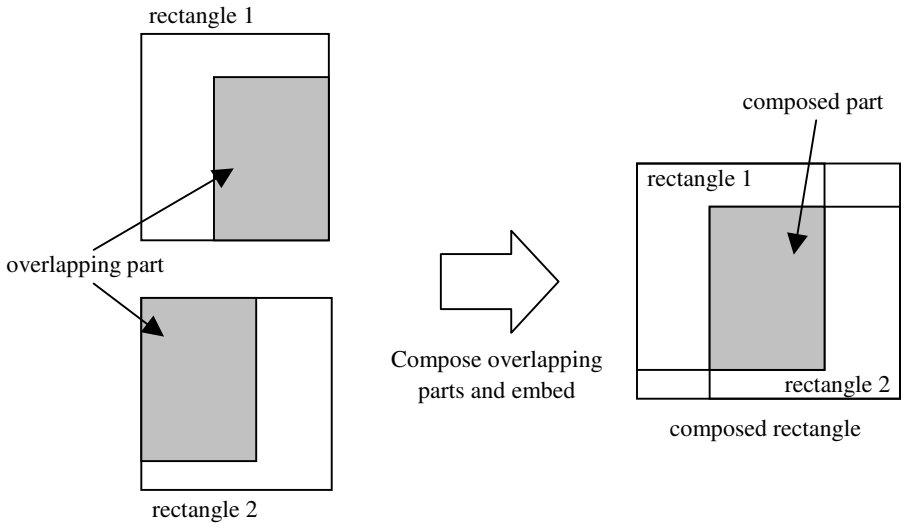

Figure 3.4. Composition of two rectangles

The composition proxy receives the encoded images from the child nodes such as the rendering server and other composition proxy. The encoded images are decoded and checked whether the two image rectangles have overlapping part or not. If so, overlapping part of two image rectangles are fetched and composed. Then, the composed part is embedded into the image rectangle which includes two image rectangles inside. The Figure 3.4 shows the composition algorithm. Composing the overlapping part of the rectangles, we can omit the other

redundant composition such as the composition with the blank part of pixels. After finishing the composition, it packs the data as the message with appropriate header and sends the packed message to the parent node.

The client viewer also has a queue to store the image data sent from the composition proxy. It fetches the encoded image and pushes it into the queue. The display routine of the client viewer popes the image data from the queue, decoding data, embedding it into the original size of blank rectangle, and maps it onto the square polygon. It also has an interface to change the argument of the volume image. When you drag the mouse over the display window, the bounding box rotating on the window and send the request message to the composition proxy when you release the mouse button. We can switch pull-up mode or burst image mode with the client viewer.

Setting the configuration file, one can specify the tree structure of the cluster. In the configuration file, The network communication information can be described such as the port numbers and network addresses of parent node, child nodes and message transfer service for each node. Each node has the ID specified by the command line argument at the beginning of the execution.

Unfortunately, there is a serious problem on the implementation. Now, the cluster in EVL consists of the nodes on which Linux is running and Linux cannot recognize more than 2 optical NICs. Therefore, the proxy cannot have 3 NICs to construct the data processing pipeline in the current situation. The actual system does not have the function to construct the connections over the optical switch with PDC. However, once the pipeline is constructed with PDC, the communication overhead in terms of the PDC does not happen during the calculation with the system. Additionally, the bandwidth of the optical NIC and regular Gigabit NIC are similar to each other (both have around 1 Gbps). We can simulate and evaluate the performance of PCE somehow with the current implementation.

4. Experimental Results

EVL has the cluster that has 16 nodes, 1 master and 15 slaves. Each node has dual Xeons 1.8 GHz and 1.5 GB memory. The graphics card is PNY Quadro FX3000 and the Gigabit Ethernet card is equipped on each node. All nodes are connected to the Gigabit Ethernet Switch to construct a cluster. In the experiment, we executed the 7 node sort-last parallel rendering system on the cluster for three sample volume data, protein.raw, hydrogen.raw and foot.raw, and measured the time intervals inside the system. The sample volume data, which are uploaded on the web site [6], are listed on the following table. The volume data have sizes 64x64x64, 128x128x128, and 256x256x256 respectively.

	Dimension	Data size
Protein.raw	64x64x64	256 KB
Hydrogen.raw	128x128x128	2 MB
Foot.raw	256x256x256	16 MB

Table 4.1. Sample volume data

We took several trials for image resolutions 128x128, 256x256 and 512x512, and measured time intervals on the client viewer, the composition proxy, and the rendering server. The measured time intervals are the total delay, queuing time, blending time, bounding rectangle calculation time and so on. The results are summarized in the following tables. Each time is the average of 30 trials for the arbitrary rotated volume images.

In the tables, total delay means how long it takes from the start of sending request message to final image displaying on the client viewer. The queuing time is the time interval that the final image spent inside the queue. It is related to the rendering rate of the client viewer and pushing/popping speed on the queue. Image embedding time is the time to embed the partial rectangle image into the original size of blank image to generate final one.

[Client viewer]

Resolution	Total delay (sec)	Queuing time (sec)	Queue push (sec)	Queue pop (sec)	Image embedding (sec)
128x128	0.1203	0.0203	0.01	0.0052	0.0026
256x256	0.1271	0.0184	0.0101	0.0045	0.0021
512x512	0.1765	0.0181	0.0097	0.0039	0.0016

Table 4.2. Performance of the Client Viewer for Protein.raw (64 polygons)

Resolution	Total delay (sec)	Queuing time (sec)	Queue push (sec)	Queue pop (sec)	Image embedding (sec)
128x128	0.1331	0.0202	0.0095	0.0044	0.0025
256x256	0.1320	0.0215	0.0091	0.0052	0.0029
512x512	0.1639	0.0192	0.0099	0.0044	0.0024

Table 4.3. Performance of the Client Viewer for Hydrogen.raw (128 polygons)

Resolution	Total delay (sec)	Queuing time (sec)	Queue push (sec)	Queue pop (sec)	Image embedding (sec)
128x128	0.1325	0.0190	0.0095	0.0042	0.0016
256x256	0.1461	0.0168	0.0103	0.0041	0.0006
512x512	0.2882	0.0191	0.0099	0.0037	0.0027

Table 4.4. Performance of the Client Viewer for Foot.raw (256 polygons)

As can be seen in these tables, the total delay increases as the resolution size increases. The number of polygons are not affected the performance explicitly, since the performance of the rendering server is so much better comparing with the processing performance inside the proxies.

In the 7 node cluster case, there are two types of composition proxies, the one at the root node of the tree connection and the intermediate nodes which have the rendering servers as the child nodes. The following tables show the spending time on the two types of composition proxies. Blending time is the composition processing time for two received images. Rectangle calc + Blending means the total processing time for blending two images and handling the rectangle calculations such as checking overlapping part, fetching it and embedding the composed image to the final rectangle. Synchronization time is the time to take for synchronizing the received data, that is, the time interval from the arrival of the first image to the arrival of the final image. It is actually the time the data spent in the queue until it is popped out. Queue pushing/popping time is the time to push and pop the data in the queue respectively. Queue data is stored in the shared memory. Attaching, detaching, reading and writing data to the shared memory is the main processes of the queue handling.

[Composition proxy (root node)]

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0013	0.0019	0.0185	0.0084	0.0061
256x256	0.0004	0.0013	0.0192	0.0064	0.0059
512x512	0.0045	0.0056	0.0387	0.0064	0.0060

Table 4.5. Performance of the Composition Proxy for Protein.raw

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0044	0.0054	0.0207	0.0073	0.0060
256x256	0.0013	0.0024	0.0190	0.0062	0.0060
512x512	0.0056	0.0066	0.0207	0.0064	0.0060

Table 4.6. Performance of the Composition Proxy for Hydrogen.raw

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0015	0.0027	0.0235	0.0056	0.0061
256x256	0.0058	0.0070	0.0222	0.0074	0.0061
512x512	0.0246	0.0261	0.0835	0.0068	0.0061

Table 4.7. Performance of the Composition Proxy for Foot.raw

[Composition proxy (intermediate node)]

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0008	0.0016	0.0200	0.0065	0.0061
256x256	0.0003	0.0010	0.0182	0.0060	0.0060
512x512	0.0017	0.0025	0.0188	0.0068	0.0059

Table 4.8. Performance of the Composition Proxy for Protein.raw

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0013	0.0018	0.0192	0.0069	0.0060
256x256	0.0215	0.0222	0.0187	0.0064	0.006
512x512	0.005	0.0059	0.0192	0.0065	0.0060

Table 4.9. Performance of the Composition Proxy for Hydrogen.raw

Resolution	Blending (sec)	Rectangle calc + Blending (sec)	Synchronization (sec)	Queue push (sec)	Queue pop (sec)
128x128	0.0014	0.002	0.0185	0.0062	0.0060
256x256	0.0067	0.0075	0.0191	0.0064	0.0062
512x512	0.0255	0.027	0.0223	0.0075	0.0061

Table 4.10. Performance of the Composition Proxy for Foot.raw

From these results, we can see that the blending time increases as the size of the resolution becomes larger. The reason is like this. If the resolution is larger, the overlapping part of the two rectangles on the composition process becomes larger. The blending calculation spends more time as the size of the overlapping part of the two rectangles increases.

Other time intervals such as synchronization, queue push and queue pop do not change explicitly in this experiment. However, the synchronization time can increase if the transferred data size is getting larger, since it includes the data receiving time. Thus, we can say that the transmission time affects the synchronization time and total performance of the frame rate on the client viewer significantly.

The next tables show the frame rate of the rendering server. The performance of the rendering getting worse as the resolution and number of polygon increases. Comparing this table with the tables of client viewer performance, it can be seen that the rendering speed does not affect the total delay. For example, the frame rate in Protein.raw 128x128 case is about 10 times faster than the 512x512. But the total delays of both cases do not have much difference.

	128x128	256x256	512x512
Protein.raw	1829.645	668.054	175.583
Hydrogen.raw	1821.282	566.927	166.8
Foot.raw	722.9	400.333	167.2

Table 4.11. Frame rate on the Rendering Server

The following table shows the average size of sent data and throughput of the sending rate on the root node proxy. The throughput is calculated by dividing the payload size by the time to write whole payload to the socket. Because the bandwidth of NIC is so large, the time interval for the small size payload cannot be measured accurately. Thus the records for small payload such as the Protein.raw case do not indicate the correct throughput. Other cases such as Hydrogen.raw 256x256, Hydrogen.raw 512x512, Foot.raw 256x256 and Foot.raw 512x512 show the appropriate values. These results are collected in the pull-up mode. Thus the utilized network bandwidth is close to maximum. However, in the push-out mode, the available network bandwidth will be almost 1/3 on the current situation, because the Scylla node does not have 3 Gigabit NICs and the composition proxy uses only 1 Gigabit NIC.

	128x128		256x256		512x512	
	Throughput (Mbps)	Trans data (byte)	Throughput (Mbps)	Trans data (byte)	Throughput (Mbps)	Trans data (byte)
Protein.raw	57.1	7140	72.7	9082	503	83603
Hydrogen.raw	312	38996.93	864	226725.81	985	137000
Foot.raw	324	40496.5	877	134161.68	1100	539374

Table 4.12. Network throughput for data transmission on the composition proxy

The following tables show the frame rate on the client viewer when the system pushes out the output image as fast as possible or keeps the sending rate in a certain speed, such as 10 FPS, 15 FPS, and 20 FPS. In order to keep the sending rate, the rendering server takes sleep for appropriate time in the redraw routine.

[Actual frame rate (Push-out mode)]

	128x128		256x256		512x512	
	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)
Protein.raw	2.948	3.333	3.303	6.333	4.290	7.333
Hydrogen.raw	2.598	6.333	3.587	4	6.633	14.333
Foot.raw	3.040	3.667	3.263	5.667	9.104	10

Table 4.13 Frame rate when server sends data as fast as possible

	128x128		256x256		512x512	
	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)
Protein.raw	8.567	9.333	8.593	9.333	7.200	7.667
Hydrogen.raw	7.958	9.333	8.499	9.333	7.139	7.667
Foot.raw	7.733	9.333	7.757	8.333	6.583	7.333

Table 4.14 Frame rate when server sends data at rate 10 FPS

	128x128		256x256		512x512	
	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)
Protein.raw	14.333	17	14.222	16	11.143	12.667
Hydrogen.raw	12.889	16.333	14.167	16.667	10.708	12.667
Foot.raw	12.143	16	11.857	14.667	8.233	10.667

Table 4.15 Frame rate when server sends data at rate 15 FPS

	128x128		256x256		512x512	
	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)	Avg (FPS)	Max (FPS)
Protein.raw	10.810	18	12.600	17.666	11	12.333
Hydrogen.raw	13.944	21.333	11.095	18.667	13.444	14.667
Foot.raw	8.091	18.333	10.952	16.333	9	10.333

Table 4.16 Frame rate when server sends data at rate 20 FPS

If the rendering server sends the data as much as possible, the actual frame rate is not good, because the data flow in the cluster is not smooth. When the message sending rate at the rendering servers is too high, the queues in the composition proxies can be full easily and frequently because once one rendering server's performance get worse, the other one send messages during the time and the many messages which cannot be synchronized arrive at the composition proxy. Controlling the output of the rendering server, the data flow inside the cluster get smooth and the frame rate is improved as you can see in the other sending rate cases. What is the optimal message sending rate on the rendering servers is the significant problem in order to maximize the performance of the system.

Finally, we can expect some rendering performance improvement if the number of polygons to be rendered is huge and the rendering speed on each rendering server is close to the optimal one in the push-out mode. In this case, the frame rate on the single machine is lower than that of each rendering server in our sort-last volume rendering system because the frame processing ability in the composition proxy depends on not the number of polygons to visualize the volume data but the spatial distribution of active pixels and resolution.

5. Conclusions and Future Works

We designed the sort-last rendering cluster system with optical switch over the optical fiber network and implemented the system to simulate and see its performance. The performance of the sort-last parallel rendering system is mainly affected by the image blending time and synchronization time. The synchronization time increases if the message transmission takes much time or the loads on the rendering servers are not balanced. The frame processing ability of the composition proxy is related to the resolution and the density of the active pixel. It does not affected by the number of polygons rendered on the rendering server. Thus we can expect the improvement of the frame rate in the push-out mode if the rendered image on the rendering server consists of lots of polygons and make a pressure to render on the single machine.

When the rendering servers generate the images as much as possible and the message sending rate exceed the ability to process them on the composition proxy, the frame rate on the client viewer gets worse. The optimal message sending rate on the rendering server depends on the blending time and synchronization time, that is, the distribution of the active pixels in the volume data, network bandwidth and load balancing. To realize the optimal data flow inside the cluster, it is necessary to realize the flow control mechanism which calculates the

optimal message sending rate from the current situation and feedback to the child nodes to control the message sending rate. That is the future research problem.

6. Reference

[1] Jason Leigh, et al., An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization.

[2] Eric He, et al., QUANTA: A Toolkit for High Performance Data Delivery over Photonic Networks.

[3] Steven Molnar, Michael Cox, David Ellsworth, Henry Fuchs, A Sorting Classification of Parallel Rendering.

[4] Don-Lin Yang, Jen-Chih Yu, Yeh-Ching Chung, Effecient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers.

[5] Michael Meissner, Jian Huang, Dirk Bartz, Klaus Mueller, Roger Crawfis, A Practical Evaluation of Popular Volume Rendering Algorithms.

[6] http://www.gris.uni-tuebingen.de/~bartz/ (volume data set)

[7] Brian Cabral, et al., Accelerated Volume Rendering and Tomographic Reconstruction using texture mapping hardware, ACM SIGGRAPH (Oct. 1994)