A Sort-Last Rendering System over an Optical Backplane

Yasuhiro KIRIHATA, Jason LEIGH, Chaoyue XIONG, and Tadao MURATA
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois 60607, U.S.

ABSTRACT

Sort-Last is a computer graphics technique for rendering extremely large data sets on clusters of computers. Sort-Last works by dividing the data-set into even-sized chunks for parallel rendering and then composing the images to form the final result. Since sort-last rendering requires the movement of large amounts of image data between cluster nodes, the network interconnecting the nodes becomes a major bottleneck. In this paper, we describe a sort-last rendering system implemented on a cluster of computers whose nodes are connected by an all-optical switch. The rendering system introduces the notion of the Photonic Computing Engine, a computing system built dynamically by using the optical switch to create dedicated network connections between cluster nodes. The sort-last volume rendering algorithm was implemented on the Photonic Computing Engine, and its performance is evaluated. Preliminary experiments show that performance is affected by the image composition time and average payload size. In an attempt to stabilize the performance of the system, we have designed a flow control mechanism that uses feedback messages to dynamically adjust the data flow rate within the computing engine.

1. INTRODUCTION

The continual drop in the cost of commodity computers has motivated aggressive research in the development of techniques for realizing and optimizing large scale computation on PC clusters. On a cluster, each node is connected to each other by a high-speed communication network whose bandwidth can reach anywhere between 1~10 Gbps, depending on the technology used. 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. Compared with the SIMD (Single Instruction stream Multiple Data stream) architecture, it is cost effective and utilization of computing resource is more efficient.

When we consider the efficiency of a parallel algorithm over a cluster, we should take one major overhead into account, i.e., communication 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 requirements 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 payload in a system becomes larger, e.g. the multimedia application, it is not easy to realize (1) and (2) at the same time. Because if one would like to send data in bulk, the size of each message become large, and if the size of each message is small, one needs to send messages more frequently. Using a huge bandwidth network can greatly reduce transmission time, and thus is the most effective way to realize these two requirements. Therefore, constructing a cluster over a high bandwidth network such as an optical network is one of the most effective solutions to handle large data sets on clusters.

The OptIPuter [1], a project currently at the Electronic Visualization Laboratory (EVL) and the University of California San Diego, is a computing model which uses optical networking as a backplane to connect clusters of computers that are collectively regarded as large computer peripherals. For example, a cluster of computers with massive RAID disks are thought of as a single large disk drive; and a cluster of computers with advanced graphics cards are thought of as a single giant graphics card. These peripherals are then interconnected with optical networks to form a wide variety of virtual computers that can be specifically customized to meet an application's requirements. The Gigabit Ethernet switch which supports the optical fiber connection converts 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 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 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 seconds. If the switch changes connections among the cluster nodes frequently, the performance of the parallel computation will be degraded. We can expect the performance improvement of the parallel computation when the connection among the cluster nodes does not change frequently compared 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.

In this paper, we discuss the design and implementation of the sort-last rendering system on the optical switch-based cluster. Contents of the paper are as follows; first of all, we explain the sort-last rendering in Section 2. Then, we discuss the design and implementation of our system in Section 3. In Section 4, we will provide the experimental results and analysis of the system. Finally, we propose and discuss the control feedback mechanism to realize the stable burst flow in the cluster in Section 5.

2. 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.

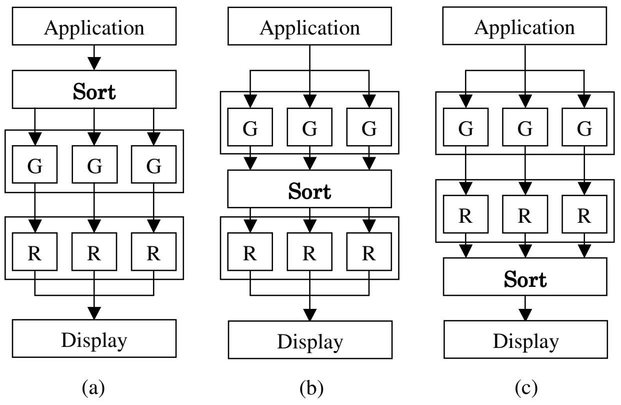

Fig. 1. Taxonomy of the parallel rendering architecture (a) sort-first, (b) sort-middle, and (c) sort-last

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 with 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 pixel, 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, we can optimize the data transfer rate in the sort-last rendering and improve its performance.

3. SYSTEM DESIGN AND IMPLEMENTATION

3.1. PHOTONIC COMPUTING ENGINE

The Photonic Computing Environment provides a 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 existing 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.

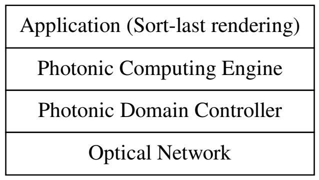
Application (Sort-last rendering)
Photonic Computing Engine
Photonic Domain Controller
Optical Network

Fig. 2. 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 communicating nodes explicitly invoke the disconnect function on the PDC. The Photonic Computing Engine (PCE) handles the establishment of the connection among the cluster nodes and provides the functionality to synchronize messages to the add-in calculation module such as image rendering module and image composition module.

The PCE has the two types of data transfer mechanism, pull-up mode and push-out mode. In the pull-up mode, the client sends a request to the PCE 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 PCE each time when it needs to change the view. On the other hand, the outputs of calculations on the PCE 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 THE PCE

The PCE is the application that provides the network pipeline among the nodes on the optical switch-based cluster and synchronization mechanism to realize the sort-last rendering. Fig. 3 shows the architecture of the PCE with 7 nodes for the sort-last rendering.

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 plays 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.

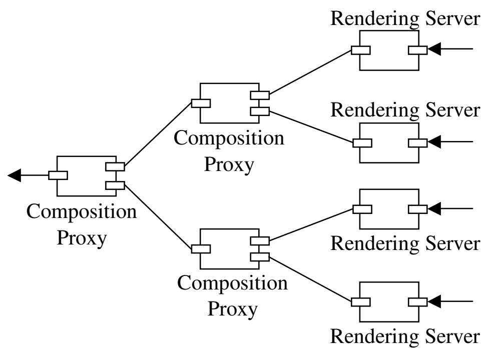

Fig. 3. Architecture of the 7-node sort-last rendering cluster

3.3. IMPLEMENTATION OF PCE

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

(1) Message transferring

The PCE generates the cluster as a tree. When the client sends the request to the PCE, 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 PCE does not change the connection pattern. The message needs to be passed along the tree-structured 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 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.

(4) Module add-in

Besides the message routing mechanism, the PCE 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 PCE 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. One 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 one would like to implement the other rendering algorithms, one can modify the display function to realize it.

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.

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. Fig. 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.

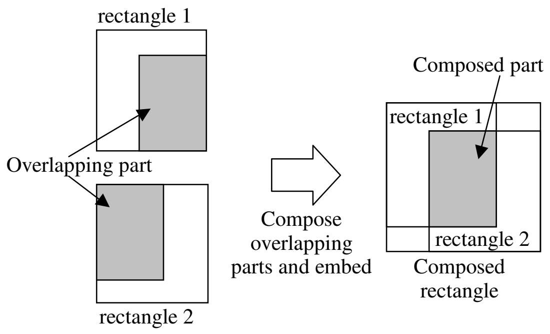

Fig. 4. Composition of the overlapping part of two rectangles

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 information of the network connection can be described by the port numbers and network addresses of parent node, child nodes and message transfer service on each node. Each node has the ID specified by the command line argument at the beginning of the execution. In the configuration file, the set of parameters for each node is separated and identified with the node ID. Each node reads its configuration part from the configuration file according to its ID.

Since the cluster in EVL consists of the nodes on which Linux is running and Linux cannot recognize more than 2 optical NICs, the proxy cannot have 3 NICs to construct the data processing pipeline currently. Therefore, the current implementation 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

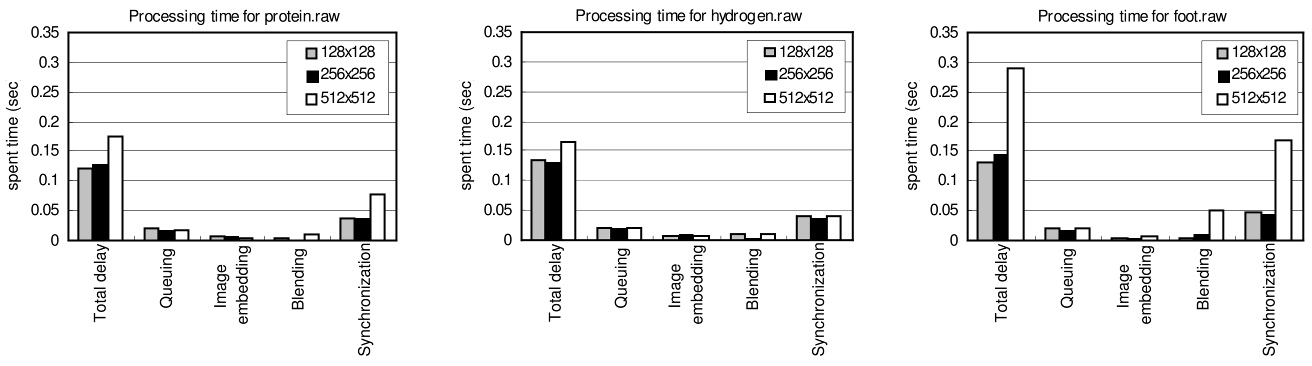

Fig. 5. Total delay and spent time of each processing in the 7-node volume rendering cluster system. (a) Processing times for protein.raw, (b) processing times for hydrogen.raw, and (c) processing times for foot.raw

computation of image rendering. Additionally, bandwidth of an optical NIC and a regular Gigabit NIC are similar to each other (both have around 1 Gbps). We can simulate and evaluate the performance of PCE somehow in the current implementation.

4. EXPERIMENTAL RESULTS

In order to evaluate the PCE, we implemented the sort-last volume rendering system over the PCE and took some experiments. 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. We implemented the 3D texture mapping method to render the volume in the system

We used 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 constructed the 7-node sort-last rendering system on the cluster and rendered the three sample volume data, protein.raw, hydrogen.raw and foot.raw, which have sizes of 64x64x64, 128x128x128, and 256x256x256 respectively.

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 graphs in Fig. 5 show the total delay and spent time for each process in the system. Total delay means how long it takes from the start of sending request message to final image displaying on the client viewer. Image embedding time is the time to embed the partial rectangle image into the original size of blank image to generate final one. Blending time is the composition processing time for two received images. 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. Finally, queuing 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.

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

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.

Fig. 6 shows 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.

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.

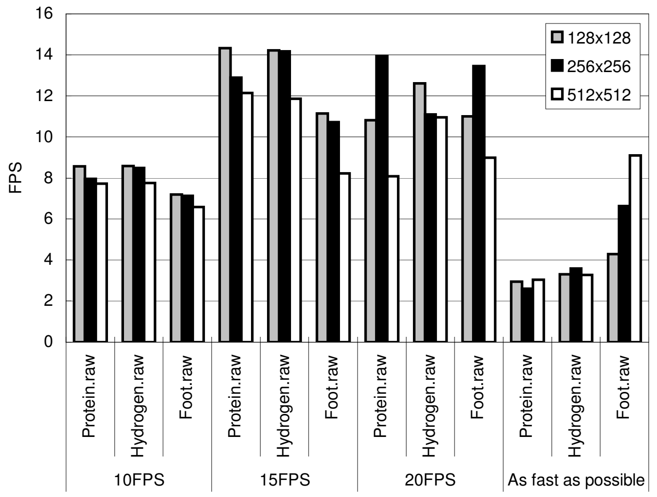

Fig. 6. Frame rate in push-out mode

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. DATA FLOW CONTROL MECHANISM IN PCE

In this section, we discuss the flow control mechanism for the push-out mode. The objective of this mechanism is to adaptively determine the optimal push-out rate for the rendering system to ensure maximum animation frame rate. The following figure shows the data flow model of the 7-node cluster, where n_{ij} is the number of messages generated by a rendering process *j*, and n_{oj} is the number of messages handled by a composition process *j*.

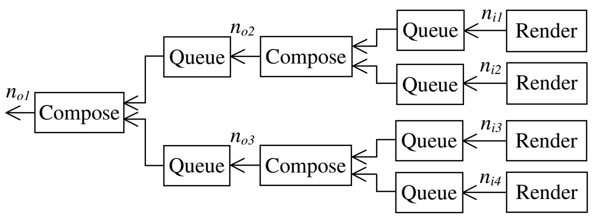

Fig. 7. The data flow model of the 7-node cluster

A composition process assembles two messages together each of which comes from a different queue, and creates one message. The whole system is composed of six small equivalent sub-systems as shown in Fig. 8, where n_i is the number of input message, and n_o is the number of output message.

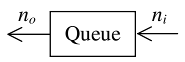

Fig. 8. The sub-system

A system achieves the maximum-speed response with a predictable operation if the system operates at a state close to stability boundary. A queue pair refers to two queues feeding the same composition process. If two queues in a queue pair are stable, i.e., always have some messages to feed a composition process and at the same time no queue has an overwhelming number of messages than another, the system can achieve the maximum message-sending rate. In other words, the system can achieve the maximum rate by maintaining a non-zero and small queue in a steady state, and draining queues when the sources do not have messages to send. Therefore, in order to achieve the maximum rate, we should maintain a small number of messages in every queue. This is the problem of making the sub-system be stable.

A queue does not change the number of messages, but it imposes delay to message flow. Thus a queue can be modeled as a delay part, so is a composition process.

Let *l=l(t)* to express the length of a queue, then we have:

$$\frac{dl}{dt} = \begin{cases} r_i - r_o & if \quad l > 0 \quad or \quad r_i > r_o \\ 0 & otherwise \end{cases}$$

Since queue length is greater than zero in a stable state, therefore, we have,

$$l(t) = \int_0^t (r_i(u) - r_o(u))du$$

whose corresponding Laplace transform is

$$L(s) = \frac{R_i(s) - R_o(s)}{s} \tag{1}$$

The message-sending rate is the derivative of the number of messages generated by a rendering process and output rate is the derivative of the number of messages processed by a composition process, i.e.,

$$r_o(t) = \frac{d}{dt} n_o(t), \quad r_i(t) = \frac{d}{dt} n_i(t)$$

Laplace transform for a derivative is

$$L\left[\frac{d}{dt} f(t)\right] = sF(s) - f(0\pm)$$

Since $r_o(0) = 0$ and $r_i(0)=0$, we have Laplace transforms for $r_o(t)$ and $r_i(t)$ as:

$$R_o(s) = L\left[\frac{d}{dt} n_o(t)\right] = sN_o(s)$$

$$R_i(s) = L\left[\frac{d}{dt} n_i(t)\right] = sN_i(s)$$

Therefore, the fluid-flow model for the open-loop sub-system can be modeled as shown in Fig. 9, where τ is the delay of the subsystem, R_i and R_o is the Laplase transform of the message-sending rate and message-output rate respectively.

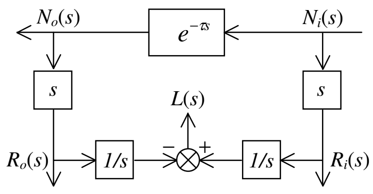

Fig. 9. Open-loop sub-system

From Fig. 9, we have the following equations:

$$R_o(s) = sN_o(s) = se^{-\tau s} N_i(s) = se^{-\tau s} \frac{R_i(s)}{s} = R_i(s)e^{-\tau s} \tag{2}$$

From equations (1) and (2), we know the open-loop transfer function of the system is:

$$\frac{L(s)}{R_i(s)} = \frac{1 - e^{-\tau s}}{s}$$

τ is small, so the system can be considered as a linear system within a small range of time. Thus the system can be analyzed by the stability criterion of a linear system, which says that a system is stable if all roots of its characteristic equation lie to the left of imaginary axis in the s-plane.

The characteristic equation of this system has only one root, $s = 0$, which means that the system is boundary stable. However, usually a boundary-stable-system is not stable in operation. To make this system stable, one option is to add a negative feedback to the system, as shown in Fig. 10, where λ is a feedback gain.

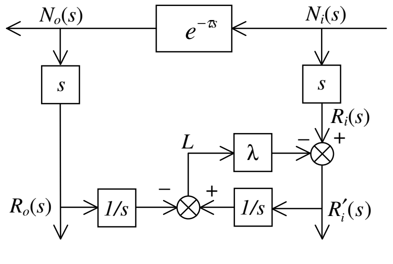

Fig. 10. Closed loop subsystem

From Fig. 10, we know,

$$L(s) = \frac{R_i'(s) - R_o(s)}{s}$$

$$R_i'(s) = R_i(s) - \lambda L(s)$$

$$R_o(s) = R_i e^{-\tau s}$$

Thus, the closed-loop transfer function is:

$$\frac{L(s)}{R_i(s)} = \frac{1 - e^{-\tau s}}{s + \lambda}$$

The system has a pole at $-\lambda$, therefore, the system is stable provided $\lambda > 0$.

We can apply this control mechanism to the 7-node cluster as follows. When the composition proxy receives the message, it returns the feedback λL to the child node where L is the current queue length. Then, the child node controls the message-sending rate to $r_i - \lambda L$. If the child node is the rendering server, it can add some time interval in the display loop to control the sending rate. If the child node is the composition proxy, it adds some time interval in the composition routine to control the message-sending rate to realize the feedback control.

6. CONCLUSIONS

We designed the sort-last rendering cluster system with optical switch over the optical fiber network and implemented the system to evaluate its performance. We found it from the experiments that the performance of the sort-last parallel rendering system is mainly affected by the image blending time and synchronization time. Synchronization time increases when transmission time grows or the loads on rendering servers are not balanced. While the frame processing ability of the composition proxy is related to the resolution and the density of the active pixel, it is relatively independent on the number of polygons rendered on the rendering server. Thus we can expect an 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 burden to render on a single machine.

When rendering servers generate images as much as possible and the message-sending rate exceed the processing capability on the composition proxy, the frame rate on the client viewer gets worse. To realize the optimal data flow inside the cluster, we propose the flow control mechanism which calculates the optimal message-sending rate from the current queue length and feedback to the child nodes to set the message-sending rate. Evaluating the efficacy of the adaptive flow mechanism is the future work, as well as testing in a fully realized optical network.

7. ACKNOWLEDGEMENTS

The visualization and advanced networking research, collaborations, and outreach programs at the EVL at the University of Illinois at Chicago are made possible by major funding from the National Science Foundation (NSF), awards EIA-9802090, EIA-0115809, ANI-9980480, ANI-0229642, ANI-9730202, ANI-0123399, ANI-0129527 and EAR-0218918, as well as the NSF Information Technology Research (ITR) cooperative agreement (ANI-0225642) to the University of California San Diego (UCSD) for "The OptIPuter" and the NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement (ACI-9619019) to the National Computational Science Alliance.

8. REFERENCES

[1] Jason Leigh, et al., "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization", **3rd Workshop on Advanced Collaborative Environments** (in conjunction with the High Performance Distributed Computing Conference), Seattle, WA, 06/22/2003 – 06/22/2003

[2] Eric He, et al., "QUANTA: A Toolkit for High Performance Data Delivery over Photonic Networks", **Future Generation Computer Systems** 1005, 1-15 01/01/2003 – 01/01/2003

[3] Steven Molnar, Michael Cox, David Ellsworth, Henry Fuchs, "A Sorting Classification of Parallel Rendering", **IEEE Computer Graphics and Applications**, 14(4): 23-32 (1994)

[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 Multi-computers", **The Journal of Supercomputing**, 18(2): 201-220 (2001)

[5] Michael Meißner, Jian Huang, Dirk Bartz, Klaus Mueller, Roger Crawfis, "A Practical Evaluation of Popular Volume Rendering Algorithms", **Volviz 2000**: 81-90 (2000)

[6] http://www.gris.uni-tuebingen.de/~bartz/

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

[8] D.C. Dorf and R.H. Bishop, **Modern Control Systems**, Addison-Wesley, 1998.