Optimization of Job Efficiency in a Heterogeneous and Distributed Beowulf Cluster through Node and Job Analysis

Robert Foertsch, Matti Kariluoma, Brian M. Slator
Computer Science and Operations Research
North Dakota State University
Fargo, ND 58108
rjfoertsch@gmail.com, matti.kariluoma@gmail.com, slator@cs.ndsu.edu

Abstract

The World Wide Web Instructional Committee (WWWIC) at North Dakota State University has been using Autodesk Maya to develop a high-definition, stereoscopic, animated short film for educational purposes. The film will run twelve minutes and contain over 36,000 frames. Each frame is a high definition PNG image of varying complexity, rendered from a model containing hundreds of textures and 750,000 polygons. To further complicate the situation, this film is in production; segments under review will need to be rendered multiple times.

To tackle this problem a 33 node Beowulf cluster was constructed, using older, retired hardware. To this was added a student lab of 21 multiple core Linux workstations, donating idle CPU cycles to render frames. In addition, our university's Center for High Performance Computing was employed, adding a portion of the 96 multiple core nodes. The challenge became developing a strategy to most effectively coordinate these heterogeneous nodes towards an organized rendering of the animation.

1 Introduction

Distributed systems can be used to solve complex problems, but they are in turn complex to build and to manage. When a distributed system is composed of disparate and heterogeneous computing units, it creates additional issues of control and optimization.

This paper is comprised of three related themes. First is the story of assembling and constructing an ad hoc Beowulf network from a handful of cast-off work-stations and its aggregation to an assemblage numbering well over 100 units. Second we describe a load balancing algorithm that seeks to fully employ all processors, the fast with the slow, in an effort to complete distributed computations as quickly as possible. Third we describe an intelligent strategy to further optimize routing and scheduling in those cases where a model of input complexity can be constructed through multiple trials.

2 Background

The World Wide Web Instructional Committee (WWWIC) at North Dakota State University (NDSU) is an ad hoc committee of faculty, staff, and students working to advance education through the use of Immersive Virtual Environments (IVE; Slator et al. 2006).

As part of this overarching goal, WWWIC is developing a twelve minute, high definition, stereoscopic short film depicting a Native American village called On-A-Slant. We have been using Maya 3D animation software for this project and, once completed, the models will consist of hundreds of textures and over 750,000 polygons. The final animation will consist of over 36,000 individual frames.

Rendering is the process by which the software takes the model and associated texture files and creates an image, usually representing a frame of the animation (Alias 2005). The resulting image will be based on the shading, lighting, and camera currently set in the animation. These attributes can be changed depending on the quality of the render job desired. For example, ray tracing, which is used to calculate the reflections in the river in the model, can be disabled, substantially decreasing the time required to render the animation. Thus, there are many parameters that can be changed depending on the quality of render, whether it is production quality or simply for internal use and commentary.

To create a full animation, one can imagine a camera moving through a model where a picture is taken every 30^{th} of a second. The individual frames are collected and encoded into a movie file. For a stereoscopic animation, a second camera is created representing the second eye, and the process is similar to create an animation.

There are two items to note. First, rendering these 36,000 frames in a reasonable amount of time is not feasible using one workstation. Even on the fastest machine available within our lab, a single production-quality frame can take up to fifteen minutes to render. Very few frames take a minute or less. For reference, we estimate that the first 5000 frames of the film, using the fastest machine of which we have access and a low quality render would take over twelve days. A high quality render takes four times as long.

Secondly, the animation is currently in development; sections of the movie will be rendered repeatedly as changes are made. To date, each segment of the movie has been rendered multiple times, in some cases, dozens of times. Summing these processing times over the last six months results in a total exceeding six years of 'render time'.

3 Deep Background

In the epic poem, when Beowulf leads his men to Denmark he meets King Hrothgar, who is described as "old and good." When Beowulf defeats Grendel, Hrothgar rewards Beowulf and his men with great treasures. Hrothgar loves Beowulf like a son, and in turn, Beowulf serves under the rule of Hrothgar during his greatest adventure (Wikipedia 2011).

Thus it is written, and so it became Hrothgar that first controlled the ad hoc Beowulf cluster, and later the CS department lab machines, and lastly the High Performance Center Beowulf cluster. See http://hrothgar.cs.ndsu.nodak.edu/ and especially http://hrothgar.cs.ndsu.nodak.edu/maya/ for more information about the current rendering project.

4 Implementation

To help manage the large scale of the rendering problem, WWWIC began by building a Beowulf Cluster. By utilizing cast-off machines being replaced by other departments on campus, the cluster has grown to 33 Pentium 4 workstations of varying processing power. These workstations are a no-cost means of expanding WWWIC rendering capabilities. In addition to the WWWIC Beowulf Cluster, the NDSU Computer Science department operates a small Linux student lab. This lab consists of 21 workstations, some with Core 2 Duo processors, some Intel Quad Core processors, and some with an Intel i7 processor. By utilizing the Unix 'nice' command, these machines can donate idle CPU cycles to rendering frames of the animation while still giving priority to student users working on their projects. In practice, student use of this lab is relatively sporadic and there are long stretches, especially in the early hours of the morning, when these machines are wholly available for rendering.

In addition to the ad hoc WWWIC Beowulf cluster and the department student lab, the NDSU Center for Computationally Assisted Science and Technology (CCAST, formerly

known as the Center for High Performance Computing) has a high performance Beowulf computing cluster. This 96 node cluster is a shared resource between other entities on campus; at most times only a portion of the nodes is available for WWWIC to render frames of the animation.

Taken together, WWWIC now has (at peak capacity) up to 150 nodes of varying hardware and architectures at its disposal to render frames of the animation. This "heterogeneous distributed Beowulf Network" substantially reduces the time required to render collections of animation frames. However, since this distributed system is composed of disparate and heterogeneous computing units, and the underlying cluster can change based on usage and availability of the nodes, it creates additional issues of control and optimization.

To optimize overall system performance, two key factors are taken into consideration. Using performance data gathered from each node, combined with information acquired from rendering frames once through the development process, we have designed and developed an intelligent frame dispatching system and load balancing algorithm. The aim is to assign frames to nodes for rendering such that total combined render time is completely minimized.

4.1 Job Distribution

How are frame rendering jobs distributed to the cluster in an optimized way? For the purposes of this paper, an optimized job distribution would result in every node finishing its allocation of frames at exactly the same time. In other words, a situation where one node has completed all of its frames while another node has five frames left to render would be non-optimal.

One simple solution would be to distribute the frames equally to all the nodes. However, the nodes in the system do not have equivalent capabilities. A single frame on an Intel Pentium 4 processor may require seven times longer to render than on an Intel i7. Our cluster is made up of heterogeneous nodes, thus the capability of each node must be taken into account when dividing the animation into parts to be rendered.

In addition to the fact that the nodes are not equivalent, individual frames are of much different complexity. Assuming the hardware is held constant, we have determined the most complex scenes take eight times longer to render than the simplest ones. This is intuitively explained in terms of modeling constraints. Some frames in a sequence may consist of a single polygon surface (the sky). This is the simplest of frames, and within the scope of this animation the quickest to render. Other frames may consist of complicated ray tracing calculations to create reflections on a river's surface, or an avatar animated and performing some action or task, or using advanced features, such as Maya Dynamics ® to simulate grass moving in the wind, or fire and smoke, or any combination of these which serve to increase the time required to render the frame in question.

We have collected statistical data for the difficulty of frames in our animation. Figure 1 shows the render time required for a particular batch of frames rendered on machines with identical hardware with ray tracing disabled. The changes in the time required to render are distinct and follow a pattern. Frames 0 to frame 150 have Maya Dynamics ® to simulate grass, and the time required to render immediate after frame 150 decreases as the grass moves out of frame. Table 1 shows three individual frames and describes the complications in each.

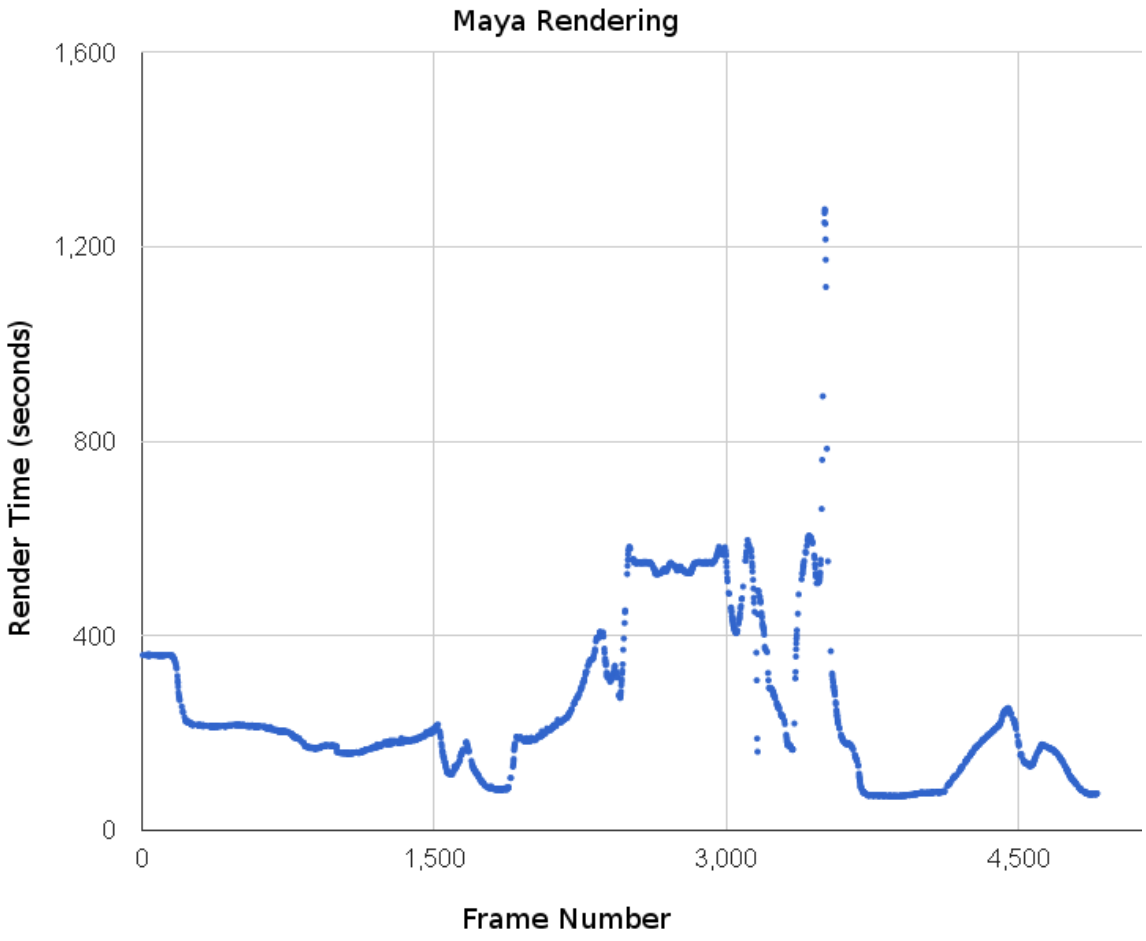

Figure 1: The changes in render time are clearly apparent depending on the section of the animation being rendered. This is the basis for determining frame complexity.

Frame	Description
	Frame 00001: The grass in the foreground uses Maya Dynamics ® to simulate wind. Also note the large area of the river; this entire area requires complex ray tracing calculations to create the reflections. This frame is of a 'medium' complexity.
	Frame 03450: This frame contains a large number of polygons for the trees and the garden, and also contains a long view of the water in the distance. This frame is of a 'high' complexity.
	Frame 04000: This frame does not contain any ray tracing reflections, nor are there many polygons in view. This frame is of a 'low' complexity; one of the simplest in the movie and the 'easiest' to render.

Table 1: A sampling of the wide and varying complexity of individual frames in the animation.

4.2 Optimization Issues

The initial job distribution system created to solve this problem resembled a queue. Hrothgar, the head node of the cluster contains a database maintaining the state of the nodes. Each node has a state defined in this database -- the two main states are *rendering* and *waiting*. Hrothgar also maintains a list of frames that need to be rendered. At first launch, the head node queries the database and finds the list of nodes that are in a state of *waiting* and dispatches each of these nodes a frame to render. On receiving a job, a node updates its state to *rendering*; once it finishes rendering, the node parses its log file and updates the information about the particular frame in the job database (this includes information about the number of seconds required to render, memory usage, page faults; essentially all of the information provided by the Maya log file is parsed). After it the node has completed parsing the log, it updates the job database to show that the frame is complete and ready to be copied to Hrothgar. It also updates its machine state to be *waiting* for another frame. Hrothgar constantly polls the database looking for *waiting* nodes and dispatches to them the next frame (sequentially ascending by frame number) until all of the frames have been dispatched.

In addition to the dispatching daemon that runs on Hrothgar, a retrieving daemon also executes. This daemon is responsible for looking for frames that are complete but not yet copied to Hrothgar in the jobs database. The retrieving daemon copies the image file and the Maya log file from the node and other associated cleanup work. Ultimately we can see the progress of the rendering job as frames are finished through an associated web page.

When Hrothgar had direct access to nodes through the network (this being the Beowulf nodes and the CS department Linux lab), Hrothgar could simply issue a command to the node and it would render. A complication arises when Hrothgar must issue a command to a node on a different private network, as is the case with NDSU's CHPC. Just like the Hrothgar Beowulf Cluster, the CHPC has a single head node that is publicly accessible, and nodes that exist solely on a private network; thus not directly accessible by Hrothgar. The solution is rather simple: Hrothgar issues a command to the CHPC head node identical to the commands sent to any other node. However, instead of rendering, the CHPC head node interprets the command and passes it on to one of the CHPC worker nodes. The worker node does the rendering, and once complete alerts the CHPC head node, which then alerts Hrothgar that the frame is complete, just like any other node. Therefore the CHPC head node acts as an intermediary bridging the two clusters together. This also implies that such a configuration would work between Hrothgar and any other cluster, given one publicly accessible member; ultimately meaning Hrothgar could easily expand to add additional resources should they become available, whether it would be individual nodes or whole clusters.

This queue system served its purpose well for many months; however, it possesses two key inefficiencies we wish to overcome. First, the system treats each node identically, without recognizing differences in processing speeds. For example, assume there are two frames left in the queue. The Hrothgar Beowulf nodes render around 15% of all the frames in a batch, therefore there is a 15% chance that these render jobs will be

dispatched to a 'slow' Beowulf node, even if a faster node is about to become available. Given a fast node renders in 15 minutes, and a slow node renders in 60 minutes, it would be advantageous for the system to wait for a fast node to finish its current job and then allocate it the frames instead of immediately allocating frame to the slower node.

The second flaw is based in the operation of the Maya rendering software. Like many such systems, when rendering, Maya allows for a starting frame, an ending frame, and a 'step' (a step of three would render every third frame). In the queue system, each job consists of one frame, so each render job would be one frame at a time. Thus, Maya needs to load into memory, load the model, render, and then clean up and end the process after every single frame rendered. Though metrics about this entire process cannot be generated, Maya does log the time required to load the necessary model. On average, this takes 30 seconds for every frame rendered in the current queue system. This would then be an underestimate of the cost of rendering one frame with each dispatched job.

For the purposes of rendering the full animation of 36,000 frames, these small process management steps add up to significant overhead. The Maya system recognizes the issue and provides a solution: instead of having a single frame be a render job, one can set a render job to be a sequence of frames. This effectively reduces the process management overhead issues, however it introduces a risk that a slower node might get the final job of a sequence of frames; trading one inefficiency for another.

The difference in render times is only a few days when dealing with the short animation of 10-12 minutes we describe here. However the problem grows in severity and importance in direct proportion to the length and complexity of the rendering job.

5 Experimental Methods

5.1 Hrothgar 2.0

An ideal rendering system would result in Hrothgar, the head node, distributing all of the frames such that each node would receive an appropriate range of frames to render and all of the nodes would finish at exactly the same time. The problem is that all the nodes have different capabilities, and all the frames have different complexities. To move forward, we make two assumptions.

5.2 Node Ability

The first assumption is that the machine rendering abilities are consistent per machine. Given that node "A" can render frame #1000 in 700 seconds once, node A will consistently render frame #1000 in 700 seconds (within a small and acceptable variation) If the time it takes Maya to render a single frame is inconsistent, it may be difficult to distribute the jobs evenly to all of the machines.

To examine this assumption, an experiment was constructed. The experimental question is: what is the difference in the time required for Maya to render a representative frame, say frame #1000, between the different groups of nodes?

To gather the necessary data, each node was instructed to repeatedly render frame #1000 with ray tracing enabled; a production quality render. A second experiment was performed with ray tracing disabled.

Overall, we find that the time it takes for Maya to render a frame on a node is consistent (usually within ten or twenty seconds). This of course is to be expected, since no parameters are changing. The node hardware is not changing, nor is the model or the frame to be rendered.

The test did yield two interesting results. First, we find that in terms of consistency, frames with ray tracing are less consistent than frames without ray tracing. Most of the frames rendered without ray tracing finished within a few seconds of the average, while frames with ray tracing enabled finished within ten to twenty seconds. However, when a frame can take anywhere from fifteen minutes to an hour to render depending on the hardware, twenty seconds should not affect our overall goal of calculating an optimal distribution of frames.

The second fact we found was that Maya performs inconsistently when memory is insufficient. Some of the Beowulf machines have less than 256 MB of memory, in these particular cases Maya rendering times were erratic. However, machines with 256 MB of memory or more performed consistently.

5.3 Frame Complexity

The second assumption is that given an identical frame, all of the nodes perform consistently relative to each other. For example, if node A can render frame X three times faster than node B, is it safe to assume that node A can render the much simpler frame Y three times faster than node B? And is it the case, in fact, that node A can render any frame three times faster than node B, given that the frame number is consistent between the two. If this is true, it means we can create a representative model of the complexity for each frame through the entire animation, and use that information to schedule rendering jobs.

This might be intuitive, however remember that Maya is a black box; we cannot make confident assertions about how it may perform. Variables such as available memory, processor speed, and available processors could affect the speed at which Maya will render a frame in ways that are not obvious or predictable.

The experimental question is this: does the relative amount of time required to render frames of different complexity remain constant if the underlying hardware is different. So, we took four groups of machines with different processor and memory attributes and

instructed them to render a representative sample of frames from the animation, with the intention of capturing the different aspects of rendering: Maya Dynamics, ray tracing reflections on the water, Maya animation, simple polygons, and so forth. These frames were specifically frames 5, 720, 1000, 2900, and 3750. All the machines rendered the frames repeatedly to generate a large data set. The experiment involved production quality frames.

The results are promising. In particular, on one of the CHPC nodes, frame five takes 1.587 times longer than frame 1000. When rendered on a node with a Core 2 Duo processor, frame five takes 1.61 times longer than frame 1000. This difference is quite small, and implies that the frame complexity is independent of the underlying hardware used to render it. Therefore, we should be able to make assumptions about how complex a frame is, and include that information in our calculations for an optimal render dispatch.

Given that the two previous experiments showed our two key assumptions are correct, we can create an intelligent job dispatching algorithm.

5.4 Additional Remarks

We will be rendering segments of this animation repeatedly as changes are made to the model. This poses the question: If the animation is being changed, will the information gathered about the time it takes to compute a frame become unreliable? Ultimately it depends on what the changes are. For example, a batch was rendered where the number of trees in the background was increased. Other batches are created where we move polygons within the model around to different places. Many of these changes, while visually creating a different animation, have very little effect on the time it takes to render the frames. Other changes, such as increasing the time the camera is focused on a body of water (because water requires ray tracing for the reflections, which is CPU intensive) will alter the time it takes to render substantially. This is important because the data on frame complexity is generated from previous renders. If a change occurs between batches that substantially changes the animation, our assumptions about frame complexity are no longer valid and we would need to generate a new set of data to base future batches from. However, since most changes don't substantially change the animation, it is not a concern.

6 Dispatching Algorithm

For this job dispatching algorithm, we use a rate equation to calculate the optimum distribution. The key insight is that we need to abstract away the concept of a frame, and instead dispatch frames to the node based on the combined 'frame complexity' (or render units, RU, as we call them through the rest of the paper) of the dispatched frames. Given that frame 1000 is one RU, if frame X takes three times as long to render, frame X would

be three RU. Thus, a RU is a ratio between frames. Based on the previous two experiments, we can make this abstraction.

We can add up the total render units to determine the total amount of work that needs to be completed. We also know how long each node takes to render one RU. These two numbers can tell us what portion of the total RU each node should render, and the time required to complete the rendering of the frames. The equation is below.

$$\frac{1}{node_1 RU} \times t + \frac{1}{node_2 RU} \times t + \cdots = RU\ Total$$

Since we know the number of seconds required for one RU from each node, and we know the total number of render units in a set of frames, we can solve this equation for *t*, the time required to render the whole batch. Once we know *t*, we can also then find the number of render units to allocate each node.

Note, however, that this creates an imperfect solution. Eventually, RUs need to be translated back into a set of consecutive frames that are allocated to each machine. Since frames cannot be broken down into portions of a frame, we cannot reach the resolution required for a perfect allocation. Each node could be allocated a number of frames that is less than their RU rating, however this would result in extra frames remaining at the end to be rendered. Or every node could do one frame above their RU rating, however this would result in the final node being allocated less frames to render, thus it would be substantially below its total RU allocation. And, in addition to all of this, trying to find an optimum positioning of each node within the set of consecutive frames is a packing problem, which we have determined to be np-complete. However, such inefficiency would result in some of the nodes doing one extra frame beyond their allocation of RU. If it is optimized in such a way that the best nodes were chosen for this inefficiency, we would be adding minutes to a render job that lasts days; this is acceptable.

7 Conclusion

The main point to take away from this research is that using distributed job dispatching based on node ability and task complexity, while working well for an active rendering environment, can also be utilized in any sort of distributed system where information is known about the complexity of each task, and there is a measurable cost for the number of jobs dispatched. It works well for Maya rendering because we can make assumptions about the complexity of each frame based on previous renders. Such a model also works well for Maya because there is a cost for having many jobs of a small size. Where the original queue model created 5000 jobs for a five thousand frame batch, the new model creates only one job per node; in this case decreasing inefficiency that occurs through process management. Assuming 150 available nodes, Maya would be spawned only 150 times. With the queue system, Maya would be spawned 5000 times.

8 Acknowledgements

The authors wish to thank Otto Borchert, Guy Hokanson, and Brad Vender for their helpful comments on this paper. The research described in this paper was supported in part by National Science Foundation Grant #IIP-0945807

References

Alias (2005), Getting Started with Maya 6.5..

Slator, Brian M., Richard Beckwith, Lisa Brandt, Harold Chaput, Jeffrey T. Clark, Lisa M. Daniels, Curt Hill, Phil McClean, John Opgrande, Bernhardt Saini-Eidukat, Donald P. Schwert, Bradley Vender, Alan R. White. (2006). Electric Worlds in the Classroom: Teaching and Learning with Role-Based Computer Games. New York: Teachers College Press. Columbia University. 192 pages.

Wikipedia (2011) Hroðgar, Wikipedia [Online]. Available: http://en.wikipedia.org/wiki/Hroðgar Accessed 15March11.