Visualization Walls Using Commodity Clusters:
Rocks Viz Roll

Student: Aaron Robinson
Advisor: Dr. Bruce Segee
August 5, 2009

Abstract

High-resolution data visualization is used for viewing large data while maintaining finer detail. The need for high-resolution arises from the very large data sets generated by many scientific models. If combined with building clusters from commodity hardware, this tool can be available to a larger number of researchers. This paper discusses using the Rocks Linux cluster distribution for easily creating visualization walls from commodity hardware. In addition, it addresses our experience at the University of Maine with implementing the framework on both newer and older hardware.

Introduction

Computer clusters designed from commodity hardware are rapidly gaining a dominant role in high-performance scientific computation. In 2004, 58% of the Top500 fastest supercomputers in the world were commodity clusters [1] and it is likely that this number has increased since than. Commodity cluster are often desired because their costs can be significantly lower than computers with custom architecture while maintaining similar performance. This fact has allowed a number of organizations, universities and businesses which don't have a large budget for a custom supercomputer to take part in high-performance computing. As high-performance computation grows, the data sets generated grow larger. Larger data sets require higher resolution displays so that users can view the big picture without losing the fine details. High-resolution data visualization is gaining a larger role in computer science because commonplace high-performance computing is more frequently generating these large data sets. As computer scientists, we can fusion the commodity cluster with high-resolution data visualization to bring this tool to even more users. When embracing such systems, it's important for us to rethink two key aspects of computer science. First, these clusters require us to redesign the manner in which we

manage the system. If a new software package is needed, it is no longer efficient to manually install that package on ever node, even with a modestly sized cluster. This fact requires us to design a methodology for propagating software changes to the cluster. Rocks is an open-source Linux cluster distribution [2] that has attempted to simplify the process of building commodity clusters by addressing this issue. Rocks provides an automated installation and configuration for a number of cluster applications. The second problem with cluster computing is designing an efficient solution for a specific problem. Unlike the personal PC which is required to be a jack-of-all-trades, the cluster computer is meant to be the master of a specific task. To solve this problem, rocks provides rolls that are sets of applications related to solving a specific problem. The Viz roll, which is specifically designed for creating visualization walls, includes DMX (Distributed Multihead X), Chromium and SAGE (Scalable Adaptive Graphics Environment). When considering the future of high-performance computation, Rocks has taken a great step in the right direction by addressing these two fundamental issues with cluster computing to create a system for easily building and administrating commodity cluster visualization walls.

Overview

Since we must redesign our methodology for managing systems when working with clusters, software installation is an important place to start. When Rocks is installed on the front-end machine, an SQL database and a master package repository are created for streamlining software installation on new cluster nodes. When a machine connects to the front-end via CD or PXE boot, the front-end makes a first pass

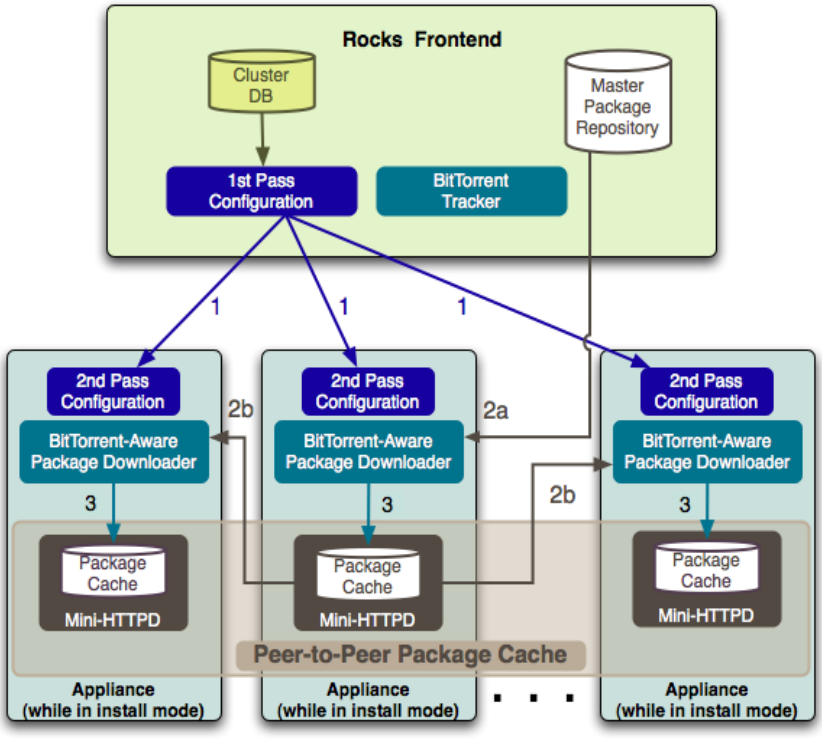

Figure 1: Rocks Avalanche Installer Process Flow (From [3])

configuration. After which the new node makes a second pass configuration which generates a Red Hat Kickstart File. Kickstart is an installation method that automates parts of installing a new OS, such as partitioning and boot loader configuration [4]. Once the new node has its Kickstart File, it begins downloading packages from the master package repository using a peer-to-peer file sharing package downloader. As more nodes connect to the front-end, they are configured using the same method as the first node. However, instead of getting packages from the front-end the new machines use peer-to-peer file sharing to download the cached packages from another node (See Figure 1). Once installation is done, all machines have the same configuration even if the cluster is heterogeneous. This has a distinct advantage over traditional imaging because this method only works for homogeneous clusters [3]. The cluster DB serves a second purpose after installation, it is used for application configurations and for propagating configuration changes to the cluster. Rocks Viz Roll provides the key components for efficiently creating visualization walls. Rocks provides a viz layout that is independent of the applications but is used for generating the configuration files for applications related to the display wall (See Appendix A). DMX, also refereed to as Xdmx, provides support for multiple displays from multiple machines. When combined with Xinerama, the user is provided with a single unified display. DMX works by creating an X proxy on the front-end machine that stitches together the X servers on the cluster nodes (See Appendix B) [5]. Chromium manipulates OpenGL command streams which allows rendering to be done on a graphics cluster [6]. The basic unit of Chromium is the Stream Processing Unit (SPU). When Chromium is used to render to a display wall using DMX, each back-end runs a *crserver* that connects to the front-end running the *mothership*. The *crserver* provides a *render* SPU used for rendering on that part of the display wall. After all the back-end machines have connected to the *mothership*, the *crappfaker* is started which manipulates the OpenGL. The *crappfaker* configures a *tilesort* SPU (See Figure 2). When OpenGL calls are made, they execute Chromium's libGL faker rather than the normal libGL. If a new window is created, the *tilesort* SPU communicates with DMX to get the window position and then the SPU directs the

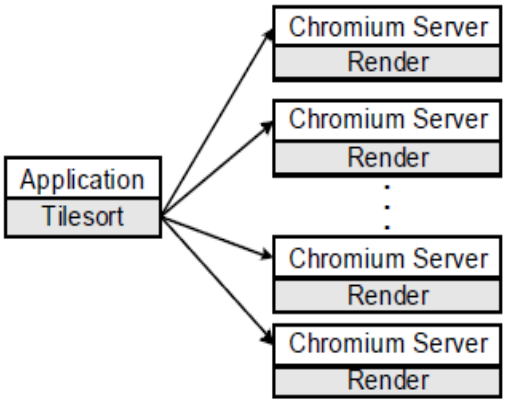

Figure 2: SPU configuration for a DMX display wall (From [8])

rendering to the back-end *render* SPUs [7]. SAGE is used to stream graphics to a display driven by a render cluster while providing a collaborative environment. The objective being to create an interactive environment for multiple users on high-resolution displays that allows collaboration on multiple distinct data sets [9]. SAGE is composed of the Free Space Manager, SAGE Application Interface Library (SAIL), SAGE Receiver and User Interface. The key component is the Free Space Manager which handles all messaging related to SAGE. It interacts with the UI by sending SAGE status messages and receiving user commands. SAGE Receivers and SAILs communicate status messages to the Free Space Manager. When a SAGE application is executed, a SAIL is created for each rendering node. Each display node has a SAGE Receiver. Pixels are streamed from SAILs to the SAGE Receiver where they are displayed (See Figure 3)[10].

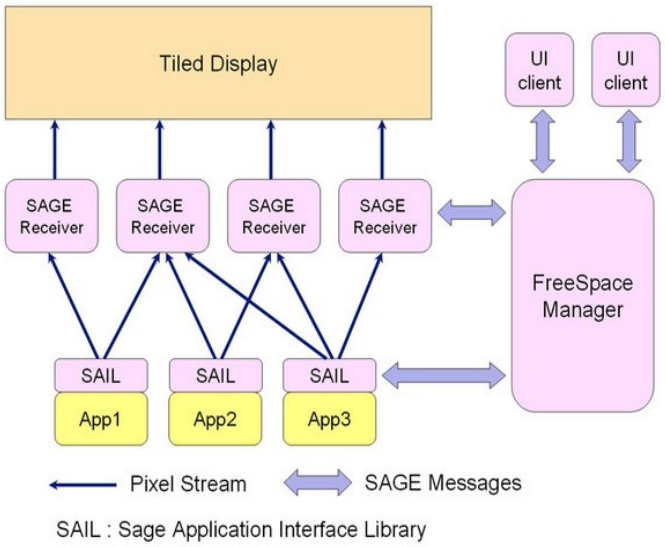

Figure 3: SAGE Framework (From [10])

Results

We originally explored using Ubuntu 9.04 with DMX and Chromium. We got DMX to run by using an archived version but Chromium would only run locally. This was unacceptable because the main purpose for us to use Chromium was the distributed rendering. After some more research we ran across Rocks and started with version 4.2.1. This version was limited because Rocks had barely developed the Viz cluster and so we moved to version 5.0. With version 5.0 we were able to get DMX and Chromium both to run. We started with a 1x2 with 2 back-end nodes and quickly scaled it up to a 3x3 with 3 back-end nodes (See Figure 4). These machines were dual Pentium 3 1 GHz with 1GB PC133 SDRAM, 100 Mbps Ethernet and Geforce4 MX 4000s 128MB. The front-end was different in that it had dual Gigabit Ethernet cards and one graphics card instead of three. The setup consistently ran glxgears at 60 fps with Chromium both in and outside of DMX. At this point there were a few issues; DMX created screen artifacts, many OpenGL applications would not render-fully if the window was

Figure 4: Rocks Viz Cluster - 3x3 display wall with 3 back-end nodes

split between nodes, xorg.conf required manual configuration outside of the Rocks provided tools, and SAGE had issues running properly. We were not able to fix the DMX artifacts, the OpenGL rendering issue, nor SAGE. Re-configuring xorg.conf was originally done by hand but after exploring the postscripts provided by Rocks in combination with Kickstart, we created shell scripts to rewrite the xorg.conf files during the installation process (See Appendix C & D). In addition, we used this script to install the legacy Nvidia driver required for the 3x3 display wall. This addition required us to add the Nvidia run script to /var/411/Files.mk under FILES_NOCOMMENT and rebuild the 411 service. The need for the postscript emerged from an issue that arose. When machines are shutdown improperly, the system assumes something has corrupted the node. When a power outage occurs, the cluster re-installs itself and destroys any manual configuration on the back-end machines. However, given these issues we were still able to install Rocks on the wall at the innovation center. At the innovation center we create a 4x4 wall with 2 back-end nodes (See Figure 5). The front-end machine in this configuration was a dual Xeon Quad-Core 2.66 GHz with 8 GB DDR2 PC2 5300, dual Gigabit Ethernet cards, and a GeForce GTX 260 896MB GDDR3. The back-end machines were Core 2 Quad 2.4 GHz with 2 GB DDR2, Gigabit Ethernet, and nVidia 8600GT Dual-DVI cards 256MB GDDR3. These machines continued to run glxgears at 60 fps. However, we ran into a vmalloc issue and the OpenGL rendering issue escalated. Originally only two of the four graphics cards were working. This was because there was insufficient video memory being allocated. To address this issue, we had to append "uppermem=512 vmalloc=512M" to the menu.lst. To do this in Rocks we used the

Figure 5: Rocks Viz Cluster - 4x4 display wall with 2 back-end nodes

command "rocks set host bootflags <node> flags=<flag>" where <node> is the name of the back-end machine and <flag> is "uppermem=512M vmalloc=512M". The escalation of the OpenGL issue was that the rendering with chromium in DMX occurred both in and outside DMX. We were not able to solve this problem.

Conclusions

Rocks Viz Roll provides an efficient framework for building easily scalable visualization walls from commodity clusters. Their software supports newer and older hardware with little to no modification. However, Rocks no longer supports DMX. It was removed from their Viz Roll in version 5.2 because their team was unable to get a working configuration with the release. Another limitation is Rocks has only a single Chromium configuration which is hard-coded into their commands. To allow for multiple configurations, the Rocks code would need to be rewritten. We believe this fact lead to the OpenGL rendering issue which occurred when a window was split between back-end nodes causing the GL to not render-fully. Beyond these issues, Rocks provides a tool allowing clusters to be created with relatively minimal system administrating. When combined with the Viz Roll, visualization walls can be created quite simply and provide incredible performance for the quality of hardware.

Future Work

Much like cluster computing, a Chromium configuration file is designed to work for a specific problem. Rocks limits the potential uses of Chromium by hard-coding a single configuration into their commands. Rewriting the Rocks Chromium command would allow the cluster to fully exploit some of the capabilities offered in Chromium. In addition, it would be beneficial to fully explore the capabilities of SAGE which we were unable to explore at this time.

Acknowledgment

We would like to thank the National Science Foundation (Grant 0754951) and the Department of Defense for funding the Supercomputing REU program at the University of Maine. In addition, we would like to thank Bruce Segee and Yifeng Zhu from University of Maine, Roger Shore and Aaron Titus from High Point University and Sure Canter and Merle Kudla from Guilford Technical Community College. These professors and instructors have been

instrumental in the fruition of higher education. Graduate students Jason Withee, Nathan Bourgoin, and Samuel Winchenbach for their support.

References

[1] G. Bruno, M. J. Katz, F. D. Sacerdoti, and P. M. Papadopoulos, "Rolls: Modifying a Standard System Installer to Support User-Customizable Cluster front-end Appliances," presented at IEEE International Conference on Cluster Computing, San Diego, USA, 2004. Available: http://www.rocksclusters.org/rocks-doc/papers/ieee-cluster-2004/paper.pdf. [Accessed: Jul. 20, 2009].

[2] "Rocks Clusters: About," [Online]. Available: http://www.rocksclusters.org/wordpress/?page_id=57. [Accessed: Jul. 20, 2009].

[3] "The Rocks Avalanche Installer," Jun. 19, 2008. [Online]. Available: http://www.rocksclusters.org/rocks-doc/papers/two-pager/paper.pdf. [Accessed: Jul. 29, 2009].

[4] "How Kickstart Works," Mar. 1, 2004. [Online]. Available: http://kickstart-tools.sourceforge.net/howkickstartworks.html. [Accessed: Jul. 30, 2009].

[5] "Distributed Multihead X Project," Jun. 13, 2004. [Online]. Available: http://dmx.sourceforge.net/. [Accessed: Jul. 30, 2009]

[6] "Chromium Homepage," Aug. 15, 2005. [Online]. Available: http://chromium.sourceforge.net/. [Accessed: Jul. 31, 2009]

[7] "Using Chromium with DMX," Sep. 1, 2006. [Online]. Available: http://chromium.sourceforge.net/doc/dmx.html. [Accessed: Jul. 31, 2009]

[8] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J. T. Klosowski, "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters," Apr. 10, 2002. [Online]. Available: http://graphics.stanford.edu/papers/cr/cr_lowquality.pdf.

[9] "Scalable Adaptive Graphics Environment," [Online]. Available: http://www.evl.uic.edu/cavern/sage/index.php. [Accessed: Jul. 31, 2009]

[10] "SAGE :: DESCRIPTION," [Online]. Available: http://www.evl.uic.edu/cavern/sage/description.php. [Accessed: Jul. 31, 2009]

Appendix A:

```
<wall>
      <defaults card="1" hres="1280" vres="3072" hborder="100" vborder="80"/>
      <col>
            <display host="tile-0-0"/>
      </col>
      <col>
            <display host="tile-0-1"/>
      </col>
      <col>
            <display host="tile-0-2"/>
      </col>
```

```
</wall>
```

Appendix B:

```
virtual show_bezels 3840x3072 {
      option +xinerama -fontpath unix/:7100 -norender;
      display tile-0-2:0 1280x3072 @0x0;
      display tile-0-0:0 1280x3072 @1280x0;
      display tile-0-1:0 1280x3072 @2560x0;
}
virtual hide_bezels 4240x3072 {
      option +xinerama -fontpath unix/:7100 -norender;
      display tile-0-2:0 1280x3072 @0x0;
      display tile-0-0:0 1280x3072 @1480x0;
      display tile-0-1:0 1280x3072 @2960x0;
}
```

Appendix C:

```
<?xml version="1.0" standalone="no"?>

<kickstart>

<post>
<![CDATA[
cp /runonce/reconfigx /etc/init.d/reconfigx
chmod 755 /etc/init.d/reconfigx
chkconfig --add reconfigx
sed s/id:5:initdefault:/id:3:initdefault:/ < /etc/inittab > /var/tmp/inittab
mv /var/tmp/inittab /etc/inittab -f
]]>
</post>

</kickstart>
```

Appendix D:

```
#chkconfig: 3 99 1
#description: runonce script to install nvidia and configure x.org
/runonce/nvidia-driver --no-network -s > /dev/null
HOME=/root
X -configure
mv /root/xorg.conf.new /root/xorg.conf
sed s/RightOf/Above/ < /root/xorg.conf > /var/tmp/xorg.conf
mv /var/tmp/xorg.conf /root/xorg.conf -f
sed s/"nv"/"nvidia"/ < /root/xorg.conf > /var/tmp/xorg.conf
mv /var/tmp/xorg.conf /root/xorg.conf -f
echo "Section \"ServerFlags\"" >> /root/xorg.conf
echo "Option \"Xinerama\" \"true\"" >> /root/xorg.conf
echo "EndSection" >> /root/xorg.conf
mv /root/xorg.conf /etc/X11/xorg.conf -f
chkconfig --del reconfigx
sed s/id:3:initdefault:/id:5:initdefault:/ < /etc/inittab > /var/tmp/inittab
mv /var/tmp/inittab /etc/inittab -f
rm /etc/init.d/reconfigx -f
init 5
```