LAUREA

Integrating open source distributed rendering solutions in public and closed networking environments

Seppälä, Heikki & Suomalainen, Niko

2010 Leppävaara

Laurea University of Applied Sciences
Laurea Leppävaara

Integrating open source distributed rendering solutions in public and closed networking environments

Heikki Seppälä
Niko Suomalainen
Information Technology Programme
Thesis 02/2010

Laurea-ammattikorkeakoulu **Tiivistelmä**
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma
Yritysten tietoverkot

Heikki Seppälä & Niko Suomalainen

Avoimen lähdekoodin jaetun renderöinnin ratkaisut julkisiin ja suljettuihin ympäristöihin

Vuosi 2010 Sivumäärä 64

Moderni tutkimustiede on yhä enemmän riippuvainen tietokoneista ja niiden tuottamasta laskentatehosta. Tutkimusprojektit kasvavat jatkuvasti, mikä aiheuttaa tarpeen suuremmalle tietokoneteholle ja lisää kustannuksia. Ratkaisuksi tähän ongelmaan tiedemiehet ovat kehittäneet hajautetun laskennan järjestelmiä, joiden tarkoituksena on tarjota vaihtoehto kalliille supertietokoneille. Näiden järjestelmien toiminta perustuu yhteisön lahjoittamaan tietokonetehoon.

Open Rendering Environment on Laurea-ammattikorkeakoulun aloittama projekti, jonka tärkein tuotos on yhteisöllinen renderöintipalvelu Renderfarm.fi. Palvelu hyödyntää hajautettua laskentaa nopeuttamaan 3D-animaatioiden renderöintiä. Tämä tarjoaa uusia mahdollisuuksia mallintajille ja animaatioelokuvien tekijöille joilta tavallisesti kuluu paljon aikaa ja tietokoneresursseja töidensä valmiiksi saattamiseksi. Renderfarm.fi-palvelu perustuu BOINC-pohjaiseen BURP- projektiin (Big and Ugly Rendering Project), joka on kehitetty pilkkomaan renderöintityö vapaaehtoisten tietokoneille.

Tämän työn teoriaosa käsittelee hajautetun laskennan järjestelmiä sekä niiden rakennetta. Keskeisenä aiheena on BOINC-teknologia (Berkeley Open Infrastructure for Network Computing), joka hyödyntää yhteisön tarjoamaa ylimääräistä tietokonetehoa. Toinen tärkeä osa-alue on keskitetty laskenta, jossa kaikki tietokoneet ovat kontrolloidussa, suljetussa ympäristössä. Vertaamme näiden teknologioiden eroja tutkimalla jo toteutettuja projekteja. Esimerkkinä käytämme hajautettua laskentaa hyödyntävää XtremLab-projektia sekä klusterilaskentaan perustuvaa Amazonin EC2-tietokonepilveä.

Tämän työn empiirisenä osuutena tutkimme hajautettua laskentaa hyödyntäviä teknologioita. Tarkoituksena oli tarjota Renderfarm.fi-palvelulle erillinen, paikallisessa verkossa toimiva tietokoneklusteri, jonka laskentateho olisi aina saatavilla.

Vertasimme neljää hajautetun laskennan teknologiaa, jotka olivat Yadra, DrQueue, Loki Render ja Farmerjoe. Saimme rakennettua kolme toimivaa tietokoneklusteria, joiden suorituskyvyssä ei ollut huomattavia eroja. DrQueue oli ominaisuuksiltaan monipuolisin ja yksi sen vahvuuksista oli tuki useille 3D-ohjelmistoille. Emme saaneet DrQueue:a toimimaan täysin mutta tulimme siihen päätelmään, että Renderfarm.fi hyötyisi siitä kaikista eniten tulevaisuudessa.

Asiasanat: Hajautettu laskenta, BOINC, renderöinti, Yadra, DrQueue, Loki Render, Farmerjoe

Laurea University of Applied Sciences **Abstract**
Laurea Leppävaara
Information Technology Programme
Information Systems and Data Communication

Heikki Seppälä & Niko Suomalainen

Integrating open source distributed rendering solutions in public and closed networking environments

Year 2010 Pages 64

Modern scientific research relies heavily on computers and scavenging computing cycles. Projects have become larger and resource requirements have grown respectively, increasing the total costs. Scientists have addressed this problem by developing distributed computing systems that offer an alternative to expensive supercomputers. Instead, computer resources are donated by volunteers who are interested in these research projects.

Open Rendering Environment is a project initiated by Laurea University of Applied Sciences. The project's most important output is the publicly distributed rendering service Renderfarm.fi. It was developed to take advantage of distributed computing for the purposes of creating 3D-graphics. It is a platform that offers computing power for graphic artists and animators who are dependent on these computing resources. The technology used on Renderfarm.fi is based on BOINC and BOINC project BURP (Big and Ugly Rendering Project). BURP makes it possible for volunteers to participate in the rendering work by donating their idle computing power.

The theoretical part of this thesis describes distributed computing and its structure. Especially we concentrate on volunteer computing based technology called BOINC (Berkeley Open Infrastructure for Network Computing). Other major technology we examine is grid computing that is used on centralized computing networks. Furthermore, we compare two real desktop grids; BOINC project XtremLab and Amazon's EC2 grid.

The empirical section of this thesis is concentrated on testing various grid computing platforms. Our goal was to provide Renderfarm.fi with computing power that is always available. This is only possible by creating a locally working computing grid where all the computers are dedicated to the system. We studied different ways of executing a render farm and then tested the performance differences between these technologies.

We examined four different render farm technologies; Yadra, DrQueue, Loki Render and Farmerjoe. We built three working render farms which proved to have insignificant performance differences. DrQueue was the most versatile with its feature to support many 3D-modelling packages. We did not manage to build a running environment with DrQueue, but we came to a conclusion that it would be the most beneficial for Renderfarm.fi in the future.

Keywords: Distributed computing, BOINC, volunteer computing, grid computing, rendering, Yadra, DrQueue, Loki Render, Farmerjoe

Contents

1 Introduction

Science is moving forward and opening new ways to implement technical inventions in everyday life. Projects like SETI@home invite people from around the world to participate in the search for extra terrestrial life forms. For the volunteer, this happens simply by installing the BOINC client (Berkeley Open Infrastructure for Network Computing) to one's computer. The BOINC client can be configured in different ways. For example, every time the computer is idle it is giving computing power to the digital signal processing of radio telescope data from the Arecibo radio observatory (Anderson 2004, 2). In 2000, SETI@home put up computing power about 15 TeraFLOPS which was greater than the fastest supercomputer at the time. These numbers are going up every year, because of growing amount of volunteers.

SETI@home is only one of the many BOINC projects that have seen the daylight. In fact, the amount of projects has been increasing steadily over time, which tells about the increased interest in volunteer computing. Projects like Folding@home and Climateprediction.net are also scientific projects that concentrate on currently important issues like medical discoveries and predicting the climate changes. (Anderson 2004, 2-3.) These are without a doubt very important subjects but the technology is also applied to projects that are focused on social and cultural purposes. Renderfarm.fi offers this kind of service for non-technical people to participate in the creation of 3D-graphics.

The ORE (Open Rendering Environment) project was launched in 2008. Its main purpose is to provide users with a web service that takes advantage of volunteer computing in rendering 3D-animations and images. Currently the technology is based on BURP (The Big and Ugly Rendering Project), which is a BOINC project that was built to do distributed rendering. This thesis examines the current situation of the web service's technology. It concentrates on finding the best solution for distributed rendering in a LAN environment to provide extra value to the service.

We will look into the BOINC framework, a middleware system which is based on volunteer computing where all the participants can share their computing power and storage space. We will also get familiar with BURP, a BOINC project that has been created to take advantage of BOINC in order to render 3D-animations and images. One objective is to clarify the differences between volunteer computing (VC) and grid computing. We will show and explain various grid-based computing platforms such as Yadra, Farmerjoe, DrQueue and Loki Render. We then test these technologies in a private network.

To get the basic understanding of distributed computing, we need to understand what are BOINC and the BOINC project BURP. That is why it is important that we cover BOINC thoroughly. We explain what is BURP and examine if it can be used as a platform for our work. We will reveal the structure of BOINC, including the client program and the server complex. Available documentation for BURP is limited but we will make our best effort to grasp the subject. We look into existing distributed rendering services that are using different technologies and compare them.

2 Background

This chapter describes the initial situation of the ORE project, need for development and its objectives in general. It introduces the key issues that form the framework for this thesis. This chapter explains briefly the reasons why we decided to write this thesis.

2.1 Initial situation

Renderfarm.fi is the most important output of Open Rendering Environment. It is a website where users can upload their animations to increase the rendering speed. Currently the render work done in the service is based on volunteers whose computers are giving power to the system, thus making the rendering faster. Vast amount of the people registered to Renderfarm.fi are users who use Blender for leisure purposes. The service has currently about 650 users and this number is increasing steadily. Although there are so many registered people it does not mean that they are all active. In fact, only twenty percent of them are regularly visiting the site and donating computer resources.

2.2 Need for development

At the moment the service only supports the open source Blender internal rendering engine. As it stands, it would be beneficial for Renderfarm.fi to offer support to other rendering engines. Another issue is that resources in volunteer based rendering cannot be guaranteed. These facts are hindrances that could make Renderfarm.fi appear unappealing to companies. For example, if a company cannot trust in meeting a deadline by using Renderfarm.fi, they will most likely revert back to non-volunteer based rendering.

Figure 1 demonstrates the service. It also shows a closed environment extension, representing our idea of how Renderfarm.fi could have permanent computing resources to counter the deficiencies of volunteer based rendering.

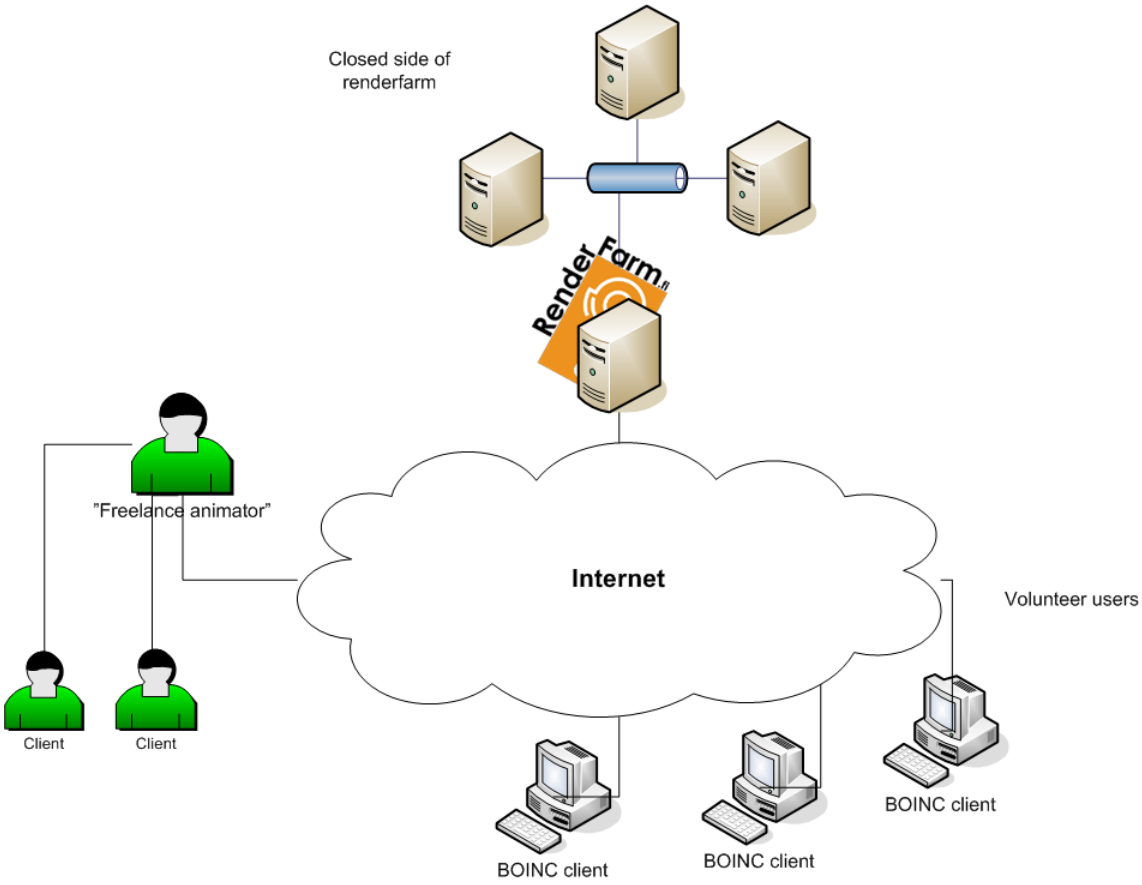


Figure 1. Design for the system

The BOINC clients in the figure 1 stand for the volunteers that are currently registered users. They are participating in rendering work through a BOINC client that is installed on their computers. The freelance animator is a professional 3D-animator who has works to be rendered for various companies. Currently an animator can only use the volunteer resources for rendering the work. In the proposal, the closed side of Renderfarm.fi would possess guaranteed rendering power at all times, which the animator could choose in order to get rendering jobs done faster and more securely.

Ease of use and a potential to attract substantial amount of people makes the BOINC project Renderfarm.fi an interesting subject to study. An interesting fact is that it is a one of a kind volunteer computing service, only meant for rendering 3D-graphics. Furthermore, all the software it uses is free, making Renderfarm.fi one of the first services that are fully open source. BOINC, BURP, Drupal and Blender are all free software and they form the backbone of the Renderfarm.fi service.

2.3 Confining the work

The purpose of this thesis is to model a locally working render farm to give resources for rendering 3D-animations. To achieve this objective we need to understand the technical side of distributed computing and find the most effective way to execute it. We will briefly analyze if it is feasible to create an extension to Renderfarm.fi and give our proposals on how this could be done.

We are also interested in distributed computing in general, because more and more large-scale calculation projects are assigned to such environments. May it be volunteer computing or grid computing, the idea of distributed computing is great because it provides a possibility for normal Internet users to participate and create things that otherwise would be out of reach.

2.4 Research method

In their research essay, Hevner, March, Park and Ram (2004, 5-6) discuss how an environment defines a problem space in which the interest for research resides. For Information Systems research, the environment is composed of people, (business) organizations and their existing or planned technologies. Business needs are assessed and evaluated within the context of organizational strategies, structure, culture and existing business processes.

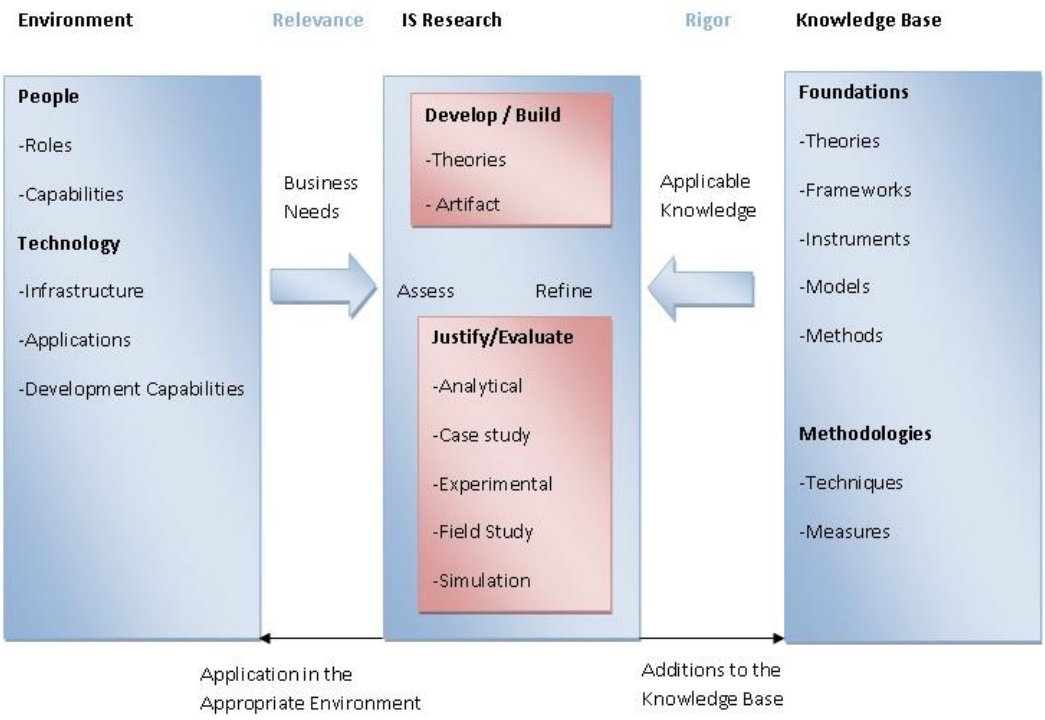


Figure 2. Information Systems Framework (Hevner et al, 2004)

Hevner et al. mention that in order to answer the business needs, IS research is to be conducted in two phases; through the building and evaluation of artefacts designed to meet the identified business needs. In our thesis the business needs are identified as an improvement to the current system in order to make it more appealing for the public. The building phase consists of implementing technological solutions and assessing their qualities. The evaluation process is ready, when the designed implementation is proven to add value to Renderfarm.fi.

2.5 Challenges

At the moment there are various open source technologies available for building distributed rendering in the LAN. Finding the best solution surely will be the hardest part of our work. To accomplish the objective we have to seek for the best solution for both the end-user and the administrator of the render farm. Since the graphical user interface (GUI) of the closed rendering environment will be integrated in the Renderfarm.fi it's not necessary for us to cover the GUI of the application in our thesis. For this work it's more important to find the technology which supports at least one external render engine (external render engines covered more closely in chapter 4).

2.6 Requirements for Implementation environment

To achieve the objective of our thesis we must consider different kind of implementation environments where it is possible to run distributed rendering technologies. Environment must pass at least these criteria to be fully considerable option:

- There must be enough working capacity for large work units and multiple jobs in the implementation environment.

- The network must be secure and robust.

- Already build environment (considering computers for master and slave mechanisms, clear network infrastructure etc.) Most important is to use left over capacity and avoid buying new computers or building a completely new environment.

2.7 Requirements for technology

Software must pass at least the following requirements:

- The slave accounts must be easy and fast to install. This is crucial when there are considerable amount of computers in the system.

- The software must be compatible with at least Microsoft Windows XP, Vista and Linux operating systems.

- There must be option to control the rendering process. The software must be sophisticated enough to provide at least controls for starting and pausing the rendering.

- To fit in our needs the software must be open source.

2.8 Key concepts

There are several key concepts to be opened in order to get successful results. Next chapters describe briefly the main concepts.

Blender

Blender is a free and open source 3D modelling and animation suite. It is a computer graphics program that allows its users to produce high quality still images and animations using three-dimension geometry. Blender has an internal render engine which is biased.

BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) is a platform for public-resource computing, granting the volunteers easy access to projects which require massive calculation work. It was originally developed by U.C.Berkeley Spaces Sciences Laboratory by the group that developed SETI@home. One of its goals is to lower the barriers of entry and encourage people to create and participate in as many projects as possible. BOINC relies on autonomous project handling which means that every project stands on its own and has its own server.

BURP

BURP is a BOINC project which primary goal is to serve public who require computing power for rendering 3D-animations and images. It is a system that makes possible for rendering jobs created by Blender to be split into smaller pieces. These parts are then rendered on the client computers and send to the server which rebuilds them again.

Condor

The goal of Condor is to support High Throughput Computing (HTC), basically with the same methods as BOINC, solving the technical and sociological challenges that exist in Public-Resource Computing. The idea of HTC environment is to deliver large amounts of computing power over long periods of time instead of getting the maximum power for a short time.

Grid computing (Cloud computing)

Grid computing, in short, means supercomputers and clusters of PC's used to do large-scale calculation work. Usually grid computing happens in Universities, research labs and companies. They exist in a closed environment and are managed usually by IT-professionals. Computers are connected full-time with high-bandwidth network connections. Because all the work happens in a closed environment it is highly secure. A grid is often expensive to maintain and hardware costs are high.

LuxRender

LuxRender is an unbiased physically based render engine. It is a free and powerful tool for creating realistic images from 3D-models. LuxRender relies on exporter scripts that export a scene from Blender.

Open source

Open source means programs where source code is freely distributed and anyone may modify the code and create their own software from it, using the same or an equivalent license.

Public-resource computing (PRC)

Public-resource computing means worldwide computing networks where all the volunteers have a chance to volunteer their computing power to scientific projects. Recently the development of PRC projects has also been moving towards social media.

Renderfarm.fi

Renderfarm.fi is a web service that provides Blender users the ability to upload their 3D animations to the site, therefore speeding up their rendering process by using public-resource computing. Renderfarm.fi is using BOINC and BURP technology to enable the PRC.

Render farm

Render farm is a cluster of computers connected in order to calculate 3D computer graphics.

Rendering

Rendering is the action which happens when an image is generated from a 3D-model created with software like Blender. Rendering applies to different features like geometry, lighting, shading, texture and viewpoint information. It calculates these effects in image file to produce the final image.

Yadra (Yet Another Distributed Rendering Application)

Yadra is an open source tool which enables the Blender files to be rendered on configured network of computers. Yadra has nothing to do with BOINC; it is a standalone program that can be used safely in closed environment to do all the rendering work.

YafaRay (Yet Another Free Ray tracer)

YafaRay is an open source ray-tracing render engine.

3 Renderfarm.fi

Renderfarm.fi is a web service that provides Blender users an ability to upload their 3D-animations to the service and therefore fasten their rendering process by using public resource computing. Renderfarm.fi has been developed since 2008, by a team leaded by Julius Tuomisto from Laurea University of Applied Sciences. Website works as a platform for uploading 3D-projects, viewing animation galleries, communicating with other people who share the same interest with 3D modelling and/or participating in volunteer computing. In other words, Renderfarm.fi has a lot of functions typical to social media sites. Renderfarm.fi is a quite different operator in public computing field as it is one of the first culture related BOINC projects. (Tuomisto 2009a.)

Users who share their computing power get credits from Renderfarm.fi, depending on how much they provide calculating time and power. For some users in Renderfarm.fi, collecting as many points as possible is their only aim. In the future, uploading works to Renderfarm.fi requires that the users have to have credits which can be traded for rendering time. Users can also be a part of a bigger group of volunteers, in the same way as in other BOINC projects and therefore collect more points to be used for future rendering. (Tuomisto 2009d.)

Renderfarm.fi is a valuable service for example for small 3D-studios, architects and non-professional or semi-professional 3D-artists. All the 3D-graphics uploaded on Renderfarm.fi are licensed under creative commons, meaning that the author of the project is free to use the output for any purpose he or she wants. Licensing makes the output also available for other users of Renderfarm.fi.

Licensing options in Renderfarm.fi:

"Public Domain"

- *The community has the same rights as the author and may sell, modify or license the output in any way they wish.*

"CC By"

- *The community may modify and sell the output under any license.*

"CC By SA"

- *The community may modify and sell the output under a similar license.*

"CC By Nd"

- *The community may sell the output but not modify it.*

"CC By Nc"

- *The community may modify the output and/or release it under any other non-commercial license.*

"CC By Nc Sa"

- *The community may modify the output under the same non-commercial license.*

"CC By Nc Nd"

- *The community may share the output but not modify it or sell it"*
  (Tuomisto 2009b.)

Authors, who want to use their output for commercial use, might want to keep their rendering projects secret. That is where the locally working render farm comes in. Renderfarm.fi should have an option where user can choose the computing grid for projects which are meant to be kept secret. There is also another significant benefit in a closed environment; administrators of the Renderfarm.fi can always be sure that there is the same amount of computing power working on a project.

3.1 Related work

Renderfarm.fi is not alone in the competition of getting the attention of Blender users. There are lots of rendering services available from all around the globe. Majority of the services are commercial, such as www.renderfriend.com from America and www.rebusfarm.com from

Germany. As far as we know there are at least two rendering services for Blender users, excluding Renderfarm.fi which does not charge their users. These are called vSwarm and Corefarm. At this point both of these services are just launched and they are in the alpha test phase. In the next chapters we are going to focus only on the cost-free rendering services.

3.1.1 VSwarm

VSwarm rendering service is based on volunteer computing and it uses the same basic idea of distributed rendering as Renderfarm.fi. Both services are using client program which handles the communication between a volunteer computer and the server. It has the same basic idea, but the distribution of the work is executed in a completely different way. When the Renderfarm.fi is using BOINC and BURP (BOINC explained more closely in chapter 6, BURP in section 6.4) for distributing work units to volunteers, VSwarm uses virtual machines to complete the same job. The client on VSwarm does not give any information about the frames rendered in volunteer computers; this enables that vSwarm animations are transparent to the users. This also means that volunteers can only hope that their computer is used for rendering frames, and not for any malicious behaviour. (How it works 2009.) At this point vSwarm service is on Alpha-phase and the rendering works only on Microsoft Windows XP and Vista operating systems. Submitting a job for vSwarm is done through the web control center of vSwarm. (System requirements 2009.)

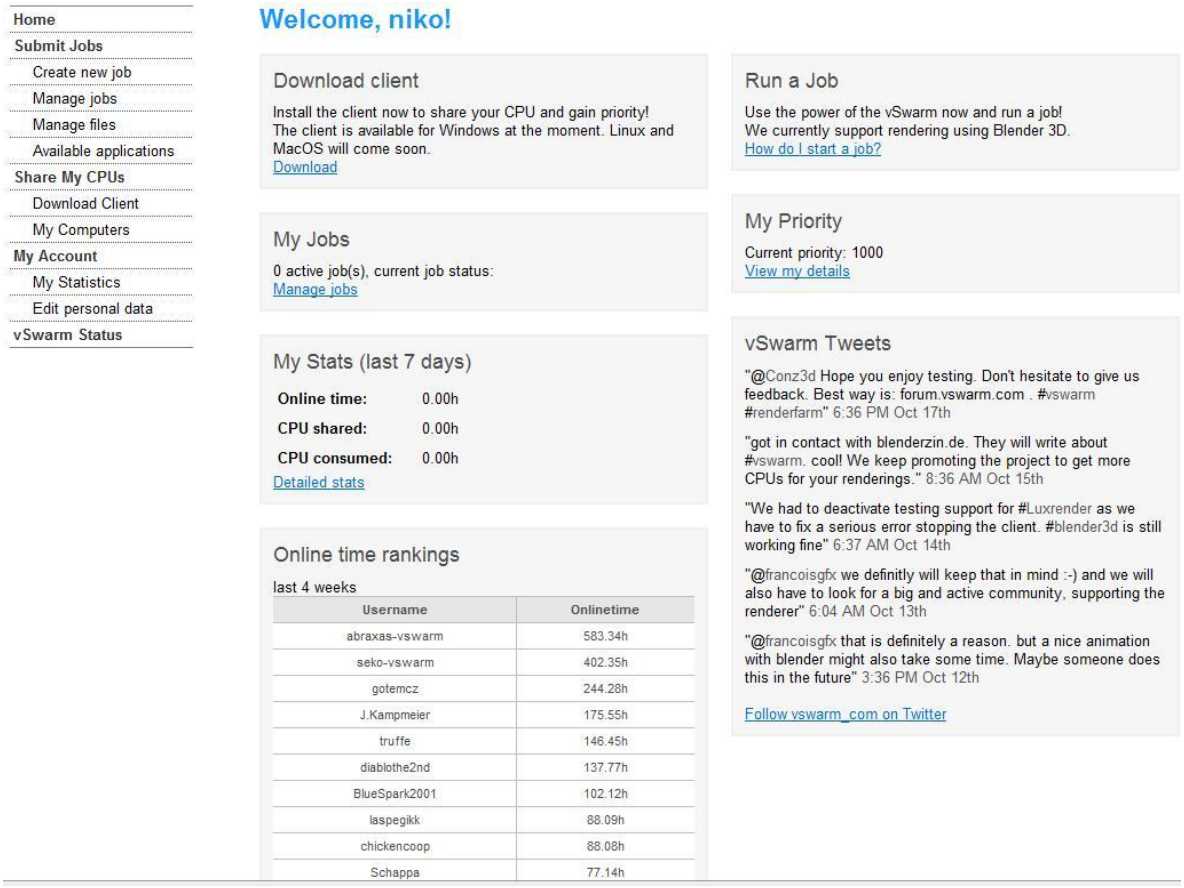


Figure 3. vSwarm's web interface

3.1.2 Corefarm

Corefarm is concentrated on YafaRay based distributed rendering. It is a non-profit seeking service just like Renderfarm.fi. People can upload their 3D-animations to the website where the workload is divided into pieces among the volunteers. There is also a possibility to download the client and just participate to the rendering work by offering computing power. At the time this paper was written there were 34 active clients. It is not much compared to Renderfarm.fi but it still fastens the work done significantly. (Le Ferrand 2009a.)

Corefarm is based on Google's native client. Native client grid is a desktop grid engine written in jocaml, which is an experimental functional programming language. There is not much information about this programming language but it is advertised to be very efficient. Naclgrid does not require rich client installation unlike BOINC. It has safe mechanisms preventing malicious usage so that the participants can not jeopardise the system. Project page also mentions that naclgrid enables multi tasking and building very flexible grids quite easily.

Obviously, one would need knowledge about the tools and the programming language in order to build such a desktop grid. (Le Ferrand 2009b.)

3.2 Benchmarking

Present moment vSwarm is the only considerable competitor of Renderfarm.fi in the field of cost-free internet rendering services. The biggest advantage of vSwarm is that the whole computing process can be kept hidden from the public. There is obvious possibility to benefit from that by using vSwarm also for commercial purposes. The problem of doing so is that volunteers do not know precisely what is going on in their computers. That is the main reason why vSwarm is considering how to reward their users for donating computing power. Another asset of vSwarm is that it already supports Luxrender (Luxrender explained more closely in chapter 4).

Compared to vSwarm and Corefarm, Renderfarm.fi benefits from BOINC long development cycle and large user community. Currently registered BOINC projects around the world benefit nearly five million installations of the BOINC client. (BOINC stats, 2010)

For Renderfarm.fi, it is not advantageous to hide the work-units or results which are calculated in volunteer computers. There are two reasons for that. First, the BOINC technology is designed to be visible for the volunteers and second, the communality aspect of Renderfarm.fi is based on the works made in volunteer computers.

4 Open source

All the technologies we use in this thesis are open source so we want to open the concept briefly. Open source means programs where source code is available for everybody, and anyone may modify the software or create their own software from it, using the same license. Open source programs are always free to use. Free, means freedom to alter the code, not necessarily that it is monetarily free. The core features of open source definitions goes as follows:

”*- Source code must be distributed with the software or otherwise made available for no more than the cost of distribution.*

- Anyone may redistribute the software for free, without royalties or licensing fees to the author.

- Anyone may modify the software or derive other software from it, and then distribute the modified software under the same terms.” (Weber 2004, 5.)

Open source is already a big part of the mainstream ICT cluster. There are thousands of open source based projects varying from small utilities to popular systems like OpenOffice, MySQL, Linux and Apache. Apache is the dominating web server with over 65 percent cut from the whole server market. Almost 40 percent of large companies in America use Linux in some form; Linux operating system has over thirty percent and Linux servers 14 percent of the whole market. Disney, Pixar and DreamWorks use Linux machines to render their special effects in big budget movies like Lord of The Rings and Titanic. (Weber 2004, 5-6.)

Everything speaks on behalf of open source, but why for example Linux is still a minor actor in operating systems on the public cluster? Steve Weber discusses this on his book called "The success of open source". He claims that the use of Linux and open source programs overall will increase in the next few years. Weber justifies this argument by saying that PC desktops, like Windows, are going to be much less important in the near future. This is caused by the general orientation towards web-services and the "dot-net" architecture which is also recognized at Microsoft. The author describes PC desktop as a steering wheel and the Internet as the engine which is built increasingly on open source software. Steering is important, but useless if the engine is not running. (Weber 2004, 7.)

The tools we are going to use in this project are all open source. Not only because we wanted so, but also because of the circumstances. The framework on current structure is based on open source, including Drupal (Website), BURP (BOINC project) and Blender (3D software). There are no good reasons why we should mess up the working framework by bringing commercial software to the mix. Besides, it can be assumed that the project would not gain any crucial value from commercial software at the moment, because it is not meant to produce any financial profit. Furthermore, our work is biased towards the technical configuration rather than thinking the monetary benefits. Although, by modelling the external environment for Renderfarm.fi, we enable a thinkable commercial use of the platform.

5 Blender

Blender is an open source suite of tools for creating 3D content. It was originally created by a company called Not a Number. After hard financial times, the founder of the Not a Number, Ton Roosendaal decided to put up the non-commercial Blender foundation and publish the source code of Blender in 2002. Since then it has gain a lot of support from users. Blender foundation has brought new features and updates frequently for Blender. According to a community survey made by CGenie in 2009 Blender has 5% market share in the field of 3D design suites. (In survey there were 2022 responses from the users of CGenie). (History 2009; CG community survey 2009.)

Blender is used for many different purposes and by various user groups. The most obvious user group would be 3d animators, but for example some architects use Blender on their work to create visualization to their models. Blender contains tools for example, modelling, animating, rigging, rendering and it also enables game creation. (Features 2009.)

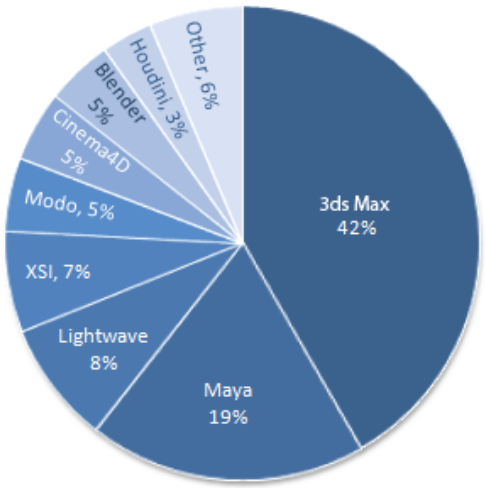


Figure 4. Community survey made by CGenie. (CGenie survey 2009.)

Blender comes for all the most used operating systems, including Microsoft Windows, OSX, Linux, FreeBSD, Irix and Sun. It can be downloaded for free from www.download.blender.org. Size of the downloadable file is under 3 MB and extracted file size is approximately 27MB in version 2.49. The small size is one of the benefits compared to commercial 3D suites. Although Blender is free, it's created for media professionals and artists. Blender is known for its hard learning curve, but when user gets familiar with Blender's graphical user interface, working with Blender can become really fast. All thanks to its wide range of keyboard shortcuts. (Get Blender 2009.)

Blender contains its own built-in renderer, but it is also possible to render 3D graphics that is made in blender using other renderers like Indico, Kerkythea, YafaRay, and so on. We will cover external renderers more closely in chapter 4. Blender supports model exporting and importing from other 3D suites, although there might be some limitations which parts of a model can be imported or exported successfully. (Import & Export. 2009; External renderers 2009.)

5.1 External rendering

One of the main reasons to use external renderers is to get more natural output for the 3D-images. To get realistic animations and still pictures, there are several different kinds of renderers which can be used. These renderers mainly use ray tracing technique (built-in renderer

in Blender uses ray tracing). Ray tracing uses simple algorithms to make image more realistic. The program basically simulates how the flow of light goes through pixels in an image plane. There are also other rendering techniques. For example, scan line rendering which is more commonly used in game development or for other purposes that require real-time rendering. There is no need to cover any real-time rendering in our thesis, so we are going to concentrate only on renderers which use ray tracing technology. (Shirley & Morley 2003, 8.)

As we told in earlier chapter, there are lots of different external renderers available for Blender in both fields; open source and commercial. These programs are often quite light and most of them can be downloaded from the internet. We are going to cover only the open source based programs.

External renderers give more options to adjust the lights to get different kinds of moods to the output. These programs enable for example night time lighting and creating illustrations of incandescent lamp lighting. Downside for external renderers is that they take longer time to finish the frame when using more detailed lighting. However most of the 3D-studios are willing to sacrifice large amount of time to get more polished outcome. (LuxRender Features 2009.)

Renderfarm.fi does not support any external renderers yet, but according to Mr. Tuomisto (Project manager of ORE) there is obvious need to integrate external renderer to the service at some point. YafaRay renderer is already considered to be the first external renderer to work with BURP projects, and some effort for integration has already been done.

There are few external renderers which are considered to be most suitable for Renderfarm.fi. We are going to study both YafaRay and LuxRenderer more closely in next sections. Both renderers are already used in other distributed rendering services. For us, the most important information about external renderers would be the fact how much longer it would take to render frames with YafaRay or LuxRender, compared to Blender's own renderer. By measuring that we can calculate how much bigger the work units would be in BOINC and how it will affect the service.

5.1.1 LuxRender

LuxRender is a free and powerful tool to create realistic images from 3D-models. LuxRender relies on exporter scripts that export a scene from the Blender. Blender supports all the features of Luxrender. It is based on PBRT renderer (Physically based rendering) developed by Matt Pharr and Greg Humpreys. PBRT was originally developed for academic use. After providing the source code under GPL, group of programmers led by Terrence Vergauwen modified the software and made it suitable for artistic use in 2007. Since then the number of features

and available exporters has increased constantly. (Goals 2009; Introduction to LuxRender 2009.)

There are many features in LuxRender which prop its state against the other open source renderers considered to be integrated to the Renderfarm.fi. Most of the features are related to the modelling and finishing the picture, for example motion blurs, which enable user to set cameras shutter speed to create more lifelike effects. The most interesting feature for us is the cooperative cross-platform network rendering, which makes it possible to use render farms with LuxRender. With that feature, it should be easier to attach LuxRender to Renderfarm.fi, and therefore make the Renderfarm.fi service more interesting for 3D artists. It's not necessary for this thesis to integrate LuxRender or any other external renderers to Renderfarm.fi, but it is quite certain that it will happen in the near future, so we need to figure out if it is possible and reasonable to use LuxRender in the closed side of Renderfarm.fi. (LuxRender Features 2009.)

5.1.2 YafaRay (Yet Another Free Ray tracer)

YafaRay is an open source render engine. YafaRay was formerly known as a YafRay renderer, but after a rewrite of the source code in 2006, it was renamed, making it clear that it is a new program. YafaRay is based on ray tracing like LuxRender but it differs as it lacks its own graphical user interface. YafaRay works as a render engine and uses Blender as a host application. This is done by using python-scripts. YafaRay is also widely used ray tracer just like LuxRender. It has great amount of different features, but most interesting aspect for us is that Yadra render farm is supposedly configured to work with YafaRay. (Estevez, Williamson & Matthews 2009.)

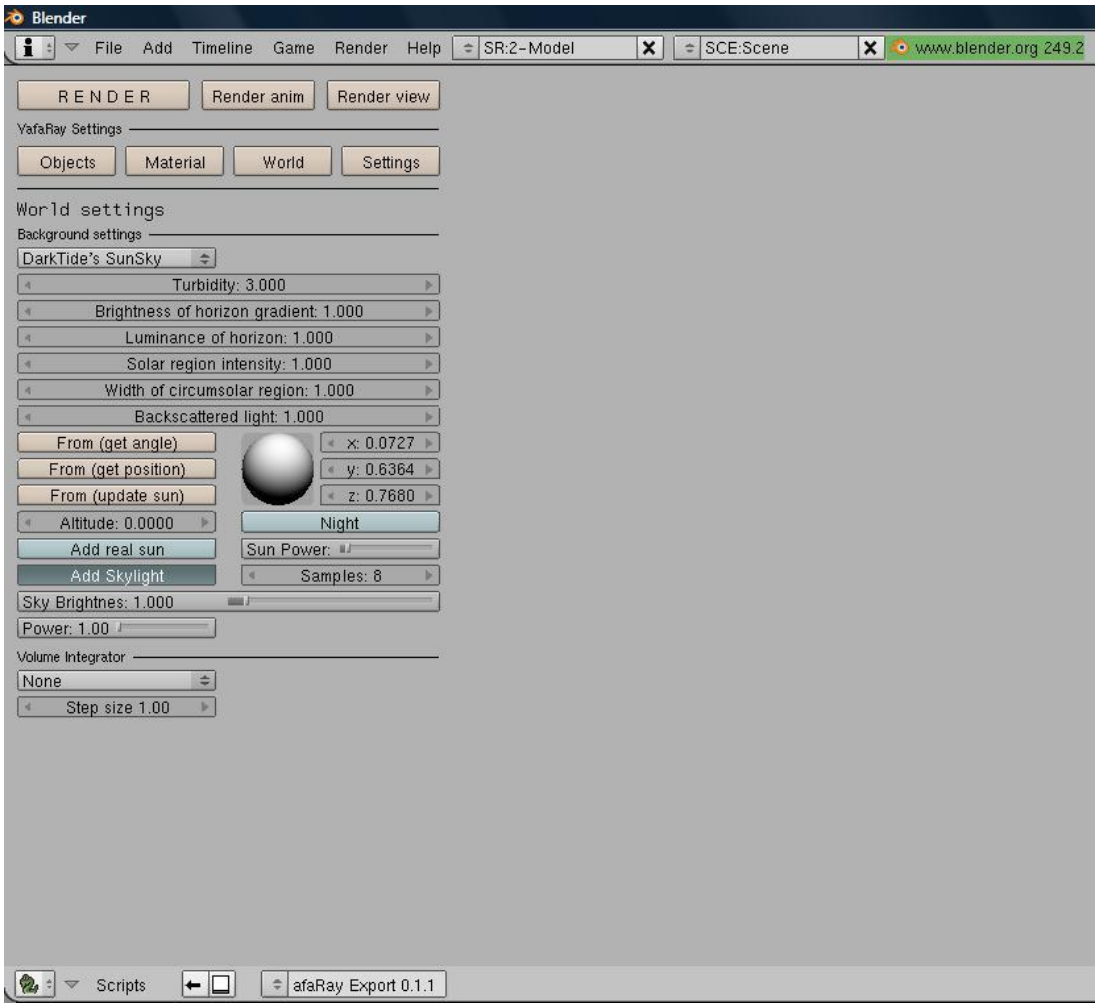


Figure 5. YafaRay python-coded settings interface in Blender

6 Volunteer computing

Volunteer computing, or public-resource computing, is a way of dividing large calculating projects into smaller bits and distributing these parts to the volunteers. BOINC is one platform designed to support VC. Computers calculate work units and then send processed jobs back to the server, where they are inspected and put back together as a one large output. The server then sends again new work units for the computers to be calculated. The given job can take from minutes to several months depending on the size of the project. Volunteer computing is build to be almost invisible. Usually there are different options from which the user can choose how the system functions; Volunteer can use his computer normally but when the computer becomes idle, calculating initiates. User can set the client to work in the background using one core from the computer without interrupting users own work. Just like computer usage, network traffic can be configured to flow as the user desires. (Carroll, Rahmlow, Psiaki, & Wojtaszczyk 2005, 3-6.)

One strength that volunteer computing possesses is the huge potential on computing power. If

a project has many volunteers, the computing power can challenge even the most advanced supercomputers. This is true in theory but the reliability issues of VC systems diminish its chances to compete on a practical level. This is mainly because the volunteered computers vary widely in terms of hardware, software, speed, availability and network connectivity. Supercomputers on the contrary, are always available offering resources to be scavenged, making them highly reliable. (Anderson & Reed 2009, 1.)

Volunteer computing is sometimes mistakenly interpreted as peer-to-peer computing where the users share and download files from each other. Programs like Napster, Gnutella and Freenet use peers (PC's) to operate, without the use of any centralized server. It is the major difference between peer-to-peer computing and volunteer computing. VC is meant to benefit the project, instead of the participants. Peer-to-peer computing is more about storing and retrieving data rather than computing. (Volunteer computing 2009.)

6.1 Security issues in volunteer computing

Volunteer computing suffer from security issues like many other web-based action. Malicious behaviour usually comes from certain individuals who do not care about the law or neti-quette. These people have also found their way in the volunteer computing field. We are addressing only BOINC related problems but the attacks are done also on many other plat-forms. BOINC-based projects are often targeted with different sort of attacks. The motives vary widely from credit falsification to account thefts. Many of the attacks are prevented by BOINC's built-in secure mechanisms, but still various vulnerabilities exist. (Security issues in volunteer computing 2009.)

Basic attacks are credit and result falsification. These are not critical attacks but they distort the real results for a project. Abusers get more credits by twisting the results of computation done or by manipulating the job to return false results. These attacks are a nuisance rather than a real threat to the system. One type of attack is the overrun of the data server. This happens when abusers repeatedly send huge files to the data server filling the storage and rendering them unusable. BOINC has an optional mechanism to prevent data server attacks, called upload certificates. Each upload have an associated maximum file size which stops too big files to be uploaded. The data server ensures that the data uploaded never exceeds this maximum size. Dangerous attacks occur when the abusers aim to make the system inoperable or to slow it down. One way of doing this is to hack into the server. Once in, the perpetrator can try to modify the database in order to plant and disguise a virus to be seen as a BOINC application. BOINC prevents this from happening by using code signing. This means that even if the hacker has access to the server he cannot cause the BOINC clients to accept the false code. (Security issues in volunteer computing 2009.)

Some perpetrators steal the participants email addresses or other account information. This is done in two ways; attacking the server or by listening the network traffic. Server attacks can be prevented by using normal security practices. Server should be behind firewall and all the unused network services to the server disabled. Access to the server should be done only with encrypted protocols like SSH. Ultimately a project should have developers who know what they are doing. BOINC projects are to be started by people who have the required expertise, and knowledge about volunteer computing in general.

Network attacks are harder to prevent and BOINC has no tools against them. A normal scenario is that an abuser is sniffing the network traffic to snatch user's account key. With the obtained key he can steal or manipulate the account's information. (Security issues in volunteer computing 2009.)

7 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) is a platform for volunteer computing granting volunteers' easy access to projects which require huge calculation work. It is developed by U.C.Berkeley Spaces Sciences Laboratory by the group that developed SETI@home. One of its objectives is to lower the barriers of entry and encourage people to create and participate in as many projects as possible. BOINC relies on autonomous project handling which means that every project stands on its own and is equipped with own server system. (Anderson 2004, 2.)

Volunteers download BOINC client and install it on their computers. They can attach the client to different projects available and define how it runs on their computer. After the client has been assigned for a project, it periodically sends scheduler requests to the project's task server. This request includes information about the host and its current, queued and finished jobs. Same message also includes a new job request. The reply message from the server may contain a set of new jobs, thus reducing the amount of multiple job requests from the client. The server sends job instances to hosts and eventually when they finish the jobs, they send the results back to the server. It then classifies the results as valid or invalid according to their agreement with other instances. (Estrada, Taufer & Anderson 2009, 3.)

BOINC provides flexible and scalable mechanism for data distribution which enables its use in high variety of applications. Its intelligent scheduling algorithms are designed to match the project requirements with available resources. These features allow the use of existing applications in common languages (C, C++) to be run as BOINC applications with little or no changes made to the code. (Anderson 2004, 2.)

Creators of BOINC have also found a solution how to attract people to participate in projects. BOINC has built-in incentive system in the form of credits. A system which grants volunteers credits according to the amount of their shared computing power. These resources are CPU power, network and disk space. Counting resource credits is very resistant to cheating, preventing false crediting. Visualization is also possible in BOINC project applications. BOINC makes it easy to add graphics to applications which can provide nice looking screensavers. (Anderson 2004, 2.)

7.1 Technology behind BOINC

BOINC consist of a client program which can be attached to all of the projects, and project specified server side complex. The client program works as a bridge between the workunit and the server, asking for new jobs if the workunit is idle. A BOINC server has at least two different servers, one web server and one for database tasks. It is centred on a relational database, whose tables answer to the abstractions of the BOINC's computing model, such as platforms, applications, application versions, jobs, and job instances. A platform is the execution environment which includes the operating system such as (Windows) and processor type (x86). The application is a program that gives computer instructions that provide the user with tools to finish a task. Application versions are executable programs, each provided with an application, a platform, version number and one or more files. Jobs are the computation to be done, each associated with an application and a set of input files. Job instances are the execution of jobs done on a particular host. Job instances are associated with an application version, depending on the requirements of the job and a set of input files. (Anderson & Reed 2009, 1.)

Server consist of five daemon processes; the transitioner, the validator, the assimilator, the file deleter and the feeder. All daemons handle their own area of responsibilities which keeps the system functioning at all times. We will handle the server and the client program features in more detail on next sections. (Søttrup & Pedersen 2005, 10.)

7.1.1 BOINC task server

BOINC has a task server which uses various programs that share a common MySQL database. The programs communicate through the database by scanning the tables in order to find jobs that match with the criteria, generated by other programs. These programs are also called as daemons. We are going to go through all the daemons in the task server, and explain their responsibilities.

The work generator creates new jobs and the input files. For example, in SETI@home there is

a program called "splitter" which processes and forms the work units from gathered data. This is feature for SETI@home project but other science applications may run a program that outputs data based on changes caused by a pattern. Work units are then send to volunteers who process them and give back the results. The work generator falls to sleep if the number of unsent instances is too high according to the threshold. (Work generator daemon 2006.)

The scheduler is a very important core daemon in BOINC. Its purpose is to handle requests from client programs. The scheduler goes through all the information received from the client, containing details about the host. The reply to the client includes a list of instances and their jobs that match. The request handling involves a lot of database operations, such as reading and updating records for user account, team, host and jobs. It runs as a fast CGI program from an Apache server, and it is capable of running job instances simultaneously. (BOINC scheduler 2006; Anderson, Korpela & Walton 2005, 3.)

The feeder makes it faster for the scheduler to access database. It is the maintainer of a shared memory segment which contains static database tables and a cache for unsent instance or job pairs. The scheduler goes through this segment in order to find instances that can be sent to a particular client. (Anderson, et al. 2005, 3.)

The transitioner goes through the jobs which states have been changed. For example, if a job is finished, it triggers a validation. It can generate new instances, flag jobs as validated, assimilated or erroneous. Many transitioners work at the same time in order to keep up with work unit/result flow. (Anderson, et al. 2005, 3.)

The validator is a back-end program which does validation and grants credit. Every BOINC-project must have one validator program to function. It compares the instances of a job and selects a recognized instance which represents the correct output. It decides the credit granted to users and hosts who give the right output and then updates those database records. (Anderson, et al. 2005, 3.)

7.1.2 BOINC client

When a user wants to start donating his/her computing power for a project, the first thing to do is download and install BOINC client which can be found from BOINC's download page (http://boinc.berkeley.edu/download.php). After installation user goes to a project's website and registers to a service, after that he gets an email from project, which contains unique project id. Project id and project's URL is used to connect his BOINC client to a project. This process is called attaching to a project. (Søttrup & Pedersen 2005, 7.)

The client program handles all the BOINC applications running in a user's computer. Applications are connected with a runtime system on a local computer where BOINC runs applications at a zero priority. Volunteer computer can then decide should it continue using the capacity or quit. Client can handle multiple projects at the same time, and it allows the user to decide how much time the computer should use for the project. User can set minimum and maximum amount of work that client should have on computer. If the amount of work goes under the minimum, client asks for more work, and downloads work units until work amount is again at maximum. At the same time client also reports finished results and checks for new application updates. (Søttrup & Pedersen 2005, 8.)

All communication between BOINC scheduler server and client is started by client program. Client communicates with a project task server using HTTP. The job request is in xml form that contains the information of volunteer computers, such as information about the hardware; number of processors, RAM size, computers availability, completed jobs and request for more jobs. The reply message contains the list of available jobs presented also in xml form. To prevent pauses in job calculation for those users who are not always connected to the Internet, BOINC client downloads more work to keep their computers busy until the next connection to the server. (Anderson & McLeod 2007, 1.)

7.1.3 Local scheduling

There are two local scheduling policies in BOINC. The first is called CPU scheduling that decides which project to run and which pending job to keep in memory for later use. The second is called a Work fetch, which says when to ask for more work, which projects, and how much work to ask. Big part of the performance of BOINC projects depends on the scheduling policies. The data that BOINC is working with is also often changing and uncertain. Most of the time there is no contact between the clients and the scheduler during the computation, so the scheduler has no idea of the state of the computation progress and the only time when scheduling decisions can be made is when the client contacts the central scheduler. For scheduling policies there are many issues it has to deal with, for example users may change user preferences, projects may be off-line or have no work available at the time. (Anderson & McLeod 2007, 2.)

According to David Anderson, scheduling policies have three different goals to achieve:

- ”Maximize the average rate at which the host is granted credit. To this end, the policies try to maximize CPU utilization and to avoid missed deadlines.”
- ”Enforce resource shares over the long term.”

- ”Maximize variety: if a host is attached to several projects with similar resource shares, the scheduler should avoid long periods when the host works for a single project.” (Anderson & McLeod 2007, 3.)

7.1.4 Web interface for administrator and volunteers

There is an administrative web interface in every BOINC project that lets administrators show following information:

- Browse the database
- Show users
- Administer users
- Create and edit applications
- Send emails to users
- Cancel workunits
- View stats about results, and failures
- Browse charts
- Browse log files (Administrative web interface 2007.)

BOINC also has a web interface for the volunteers. It contains all the BOINC data concerning the client. In Renderfarm.fi, the BOINC data is integrated with the rest of the website. Figure 6 demonstrates the volunteer interface.

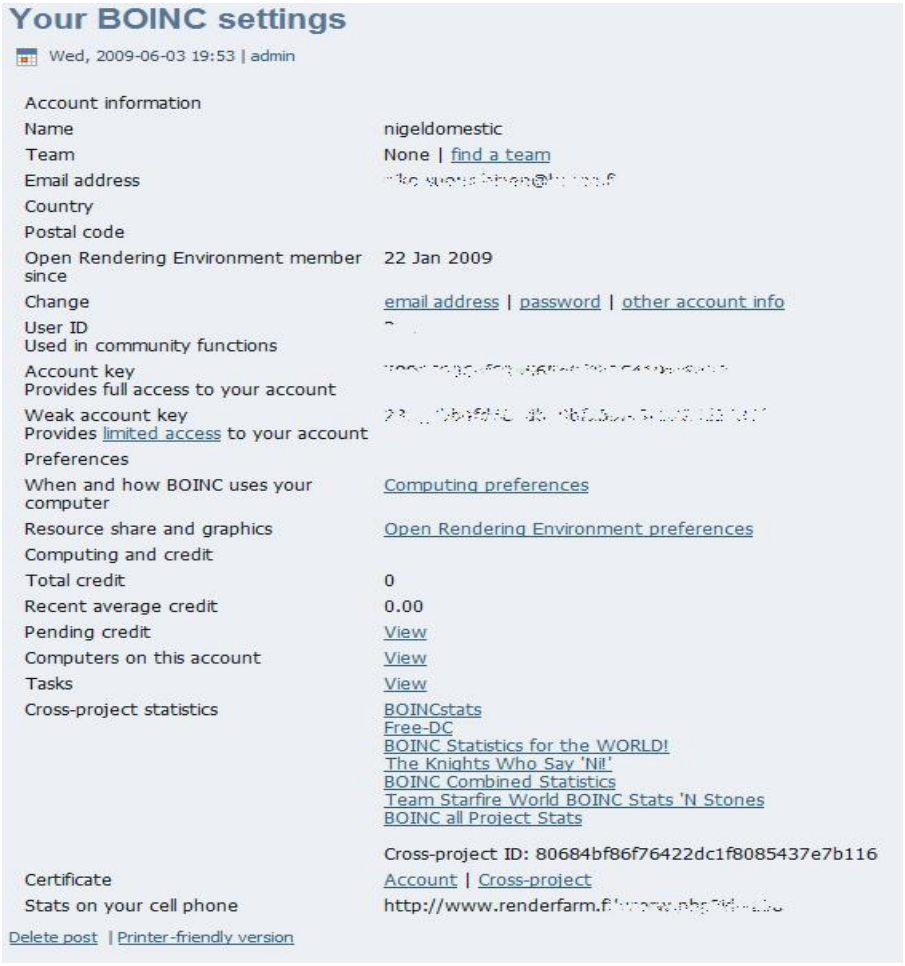


Figure 6. BOINC client view in Renderfarm.fi (All the private data is hidden)

7.2 Jobs

A project's database can have millions of jobs saved in its tables and the server can handle hundreds of scheduler requests per second. For every request, a scan is made from the tables in order to find the best job for the client. If the job cache is large, like in big projects, it is more likely to contain jobs that fit to a given client. (Anderson & Reed 2009, 3-4.)

A BOINC project job selection starts from scanning the job cache at a random point and making jobs suitability checks that do not require database access. For example, making sure that the host has enough RAM and disk space, and checking if it can complete the job by its deadline. If the job gets through these checks, it is locked and then assured from the database that it has no other job instances issued to a host belonging to the same volunteer. System repeats this until adequate jobs are selected to satisfy the host's work request. (Anderson & Reed 2009, 4.)

7.2.1 Job size matching

The job size can be set randomly for many applications. Setting the right size for jobs within a project can be difficult. If job size is too big, slow hosts won't be able to complete the tasks within their deadline. For example, in BURP and Renderfarm.fi an average work unit takes three hours to complete. On the other hand, if the job size is too small, load on the server will be extreme. BOINC's idea is to give certain time intervals T between scheduler requests. It sends each host a job that will take T time to complete. But this method requires that the project must create suitable distribution of job sizes. The scheduler must also pay attention to the job sizes. The job size issue is handled by a Census program which computes the mean and standard deviation of host throughputs. (Anderson & Reed 2009, 5.)

The Census program is created to list all the hosts in the database giving them different HR (homogeneous redundancy) classes. It grants each HR class with its own recent average credit. After it has computed the average host throughputs, the Feeder access this information and saves it to the shared memory segment. Shared memory can hold in its cache roughly 1000 jobs. The feeder also periodically computes the average and standard deviation of the job sizes within shared memory. Basically the scheduler tries to find hosts which throughput differ standard deviation from the average by X amount. Then it attempts to send jobs that differ by X amount of the standard deviations from the job-size average. (Anderson & Reed 2009, 4-6.)

There are situations when scheduling decisions are based on information that BOINC does not know about. For example, whether the client has the needed software package installed. In this case BOINC project allows "probe jobs" to be sent. They are send exactly once for each host and the results are stored in the host record in the database. Results are typically XML documents and they can be referenced in a project-specific scoring function. Next section handles how scoring function works. (Anderson & Reed 2009, 6.)

7.2.2 Job scheduling based on scoring function

On the sections earlier we explained the different criteria for job assignment corresponding to host and job diversity. These criteria generate a need for one job assignment policy. This is done by scoring function S(J, H). Anderson and Reed explain in their paper how scoring function works. *"The job assignment policy is essentially to send H the cached jobs J for which S(J, H) is greatest. The default scoring function is a linear combination of the following terms:*

- The expected FLOPS of the fastest application version available for H (try to send jobs for which fast applications are available)

- 1 if the volunteer selected J's application, else 0 (try to give volunteers jobs for the applications they selected).

- 1 if J is already committed to the homogeneous redundancy class to which H belongs, else 0 (try to finish committed jobs rather than create new ones).

- (A-B) ^{2}, where A is the size of the job and B is the speed of the host, expressed in units of standard deviation.

- The disk and RAM requirements of J (avoid sending jobs that need few resources to hosts that have lots of resources).

- 1 if we've decided to send H a non-replicated job, and J has no other replicas, else 0.

Projects can adjust the weights of these terms, or they can replace the scoring function entirely." (Anderson & Reed 2009, 6.)

7.2.3 Volunteer application selection

A project may have multiple applications and jobs are assigned to them regardless of the application. However, BOINC has several mechanisms which limit or control job assign, depending on the application requirements. The feeder enforces jobs to be dispatched on given ratios between applications. This means that if ten jobs are dispatched, six of them should belong to application 1 and four to application 2. (Anderson & Reed 2009, 5.)

Projects allow the volunteers to select applications. They will be sent only jobs concerning those particular applications. Volunteers can also configure the jobs to be sent from other applications if none are available for their selected application. (Anderson & Reed 2009, 5.)

7.3 Problem situation

Volunteer computing is based on volunteers, each equipped with different kind of hardware and software. This diversity places many demands on BOINC. Foremost is the job selection problem which a BOINC server has to make. It has to go through millions of jobs in the database in order to find the "best" one for the client. After this, the job needs to fit to a complex set of criteria that is a feature in BOINC. Furthermore, the server gets hundreds of these job requests per second. There are cases where the server has received so many simultaneous requests causing it to shut down in a disastrous overload. BOINC has a number of mechanisms to prevent this from happening. There is a client/server communicational back off system in

the case of failure. The back off system applies also to computational errors. If an application fails for whatever reason, the BOINC client will not try repeatedly contact the server; instead, it delays the requests based on the number of failures. (Anderson 2004, 4.)

We have studied BOINC structure in depth because it is one of the first and most popular distributed computing platforms. It gave us a lot of important information about volunteer computing and we strongly believe that it has been great help in our efforts. BOINC project BURP is being used in Renderfarm.fi which is the reason we wanted to learn more about it. The original plan was to use it in our work as a framework. As we gained more knowledge about distributed computing in overall, we became aware of the fact that BOINC based environment is not suitable for a locally working render farm. We will explain in next chapter more in detail why BOINC is not the best choice for our work.

7.4 BURP (The Big and Ugly Rendering Project)

BURP is beta-testing stage project using BOINC as framework to do large-scale distributed computing for rendering 3D-animations. It is powered by participants who share their spare computation power. BURP went online in June 2004 and it has gone through different development phases to reach the stage that it is now. The tests have shown promising results. Not only that distributed rendering for 3D-animations is possible, but also that it has potential to be a rival for big commercial rendering farms. It enables professional, computer-generated rendering for people who do not have the money to upload their work on expensive computing grids. Although BURP is technically quite complicated it is made transparent and easy to use for the artists and participants. Currently BURP supports only 3D-modelling tool Blender because the project is based on open source software. (Link of the week – BURP 2008.)

So far BURB has been a one man effort, created by Danish national Janus Kristensen. Since BURP is open source it was our original plan to implement it as our framework. We have found out that it is very hard to create a project similar to BURP because of the lack of documentation. Also, the feasibility of a BOINC-based project for our grid computing environment has raised a question mark. This is because BURP is meant to be used in a volunteer computing project, where all the computers, users, administrators and environment are unknown. Unlike our project, that would use defined cluster of computers in an administrated environment. To address these problems we contacted Janus Kristensen and asked him for assistance. He replied quite clearly that for grid computing environment such as ours, BURP is definitely not the best choice. He pointed out that we should consider other tools to use as the framework. His suggestion was to look into software called DrQueue which has been widely used to do rendering in a controlled environment.

7.5 Problem description with BURP implementation

- Documentation for BURP is inadequate. BURB's home site offers discussion about BURP on its forums, but finding the crucial information is challenging.

- Comprehending the technicalities that lie within BURP installation.

- Integration between closed environment grid computing and public Renderfarm.fi websites.

- Finding the right research methods to get trustworthy results to prove or disapprove the feasibility of such integration.

Based on the conversation we had with Janus Kristensen, it appears that we should concentrate our work towards grid-based computing technology rather than BURP. To prove this statement we made a comparison between volunteer computing and cloud computing (Chapter 9). Cloud computing as a term has quite similar meaning than of grid computing, with an exception that clouds are often referred as commercial computer clusters such as Amazon's EC2. By now, we know that volunteer computing does not serve our purposes but it is important to understand how it differs from cloud computing in the means of computer power and reliability.

8 Grid computing

Grid computing is one way of doing distributed computing. Basically a grid works as a large virtual supercomputer with many computers attached to it. The basic idea is the same as in volunteer computing; using spare computing power to do calculations. But the methods and workload sharing are different. Grids are also transparent for the users, so they do not see where the computing power is coming from and what is happening between the networks. Unlike BOINC which operates in the public field, grid computing happens inside organizational boundaries. Grid computing is often referred with the term Virtual Organization (VO). VO is a collection of individuals or organizations that share the same goal. The users within the VO share their computing power, data and applications to reach the common goal. Established, currently working grids are mostly created by scientists who are connecting research institutions and their machines. They are not commonly used by the public, unlike Internet. (Søttrup & Pedersen 2005, 43.)

The definitions of what is a grid vary widely. Ian Foster made a three point checklist from which one can identify a grid.

"*- Grid is a system that coordinates resources that are not subject to centralized control.*

- It uses standard, open, general-purpose protocols and interfaces

- Grid delivers nontrivial qualities of service" (Foster 2002, 2-3.)

If we contrast these checkpoints to BOINC we can see that BOINC is clearly not a grid. It does not use general-purpose protocols nor deliver nontrivial qualities of service. Definitions of a computing grid are now described but it still lacks standardization which could make it to be seen as a global inter-grid, similar to the Internet. If the protocols that grids use are standardized, it could open up the possibility for a one big computing grid. On the next section we give an example of a grid computing platform called Condor. (Søttrup & Pedersen 2005, 43.)

8.1 Condor

Condor is other major project that was created to support distributed computing. It's based on grid computing which we described before. We are going to explain Condor quite briefly compared to BOINC, because we do not have the resources to cover it so thoroughly. It is one of the biggest distributed computing systems so it is worth mentioning.

Condor was developed in 1984 at the University of Wisconsin by Miron Levny in cooperation with Dewitt, Finkel, and Solomon who are the designers of powerful Crystal Multicomputer. This combination added with Remote Unix software, designed by Michael Litzkow, resulted in creation of Condor. The goal of Condor is to support High Throughput Computing (HTC), basically with the same methods as BOINC, solving the technical and sociological challenges that exist in Public-Resource Computing. The idea of HTC environment is to deliver large amounts of computing power over long periods of time instead of getting the maximum power for a short time. Condor can be seen as an ancestor to BOINC because they share some common roots in the University of Wisconsin. (Søttrup & Pedersen 2005, 14; High Throughput Computing 2006.)

The backbone of Condor is called a pool. It consists of central resource manager, one or more submit machines and execute machines. The central resource manager is responsible for pairing up the resources with jobs. It gains the knowledge about these through classified advertisements (ClassAds). Providers advertise what they have to offer in the form of a service and also give information of what kind of restrictions apply to it. Same time users advertise their needs and wishes and sent the ClassAds to service called matchmaker which then matches the jobs with services. (Søttrup & Pedersen 2005, 15.)

Providers are the users who own the executing machines, and the people submitting the jobs are the users. Submit machines are basically only for submitting the jobs to the pool. After submitting, jobs are run by the execute machines which can be dedicated computers or normal computers configured to do processing when idle. (Søttrup & Pedersen 2005, 17.)

8.2 Condor and BOINC

Condor differs from BOINC quite much but the major difference is that while Condor is meant to run jobs inside the organizational boundaries, BOINC works outside the borders of organization. This leads to the fact that Condor can trust on its resources whereas BOINC has to deal with unreliable resources. BOINC is still better choice for volunteer computing because of its reliable results, user protection and its capability to overcome firewalls. Although Condor has many benefits speaking on its behalf, it is mainly meant for closed environment grid computing.

There are now attempts to integrate BOINC and Condor. The project is developed to answer the periodical problem that Condor faces from time to time. If there are no jobs to be done in the system, all the resources it uses go to waste. This idle state is the result of Condor's incapability to redirect its resources to another project. Unlike BOINC which can be assigned to multiple projects at the same time, so that there is always work to be done. The developers of Condor are making effort to make use of BOINC's never-ending supply of jobs. Condor is modified in a way that it has knowledge of BOINC client. This way the administrators of a Condor project can modify the machines to hand over the idle resources to BOINC. The priority of a Condor project is still this; if a Condor job is assigned to the system, it terminates the resource usage for BOINC and starts crunching numbers for Condor. (Wright 2006, 1.)

9 Traits and trade-offs between grid and volunteer computing platforms

One of the original topics of this thesis was to analyze if it is profitable to extend the BOINC project Renderfarm.fi. We did not make any cost estimations because they were not within our development goals. Instead, we tried to get an overview of the project's possibilities from a resources point of view. Does Renderfarm.fi have an actual need for extra computing power? Should we use VC or a grid computing system? In the next chapters, we study those questions based on research carried out in other projects. Specifically, we will reference the scientific paper "Cost-Benefit Analysis of Cloud Computing versus Desktop Grids" (Kondo, Javadi, Malecot, Cappello & Anderson 2009) from the University of Berkley. They examined the monetary cost-benefits and performance tradeoffs between two platforms; cloud and volunteer computing (desktop grids). Although, cloud computing is referred as a commercial platform it is still a grid computing platform, which makes it relevant.

We will use the comparison made between Amazon's cloud computing based platform EC2 and volunteer computing based platform XtremLab, to find out the main differences between these two. Making realistic estimations is hard because of the many variables involved in measuring performance and cost efficiency between platforms. This is why we have to take advantage of already collected information on server measurements and financial expenses, provided by real volunteer computing and cloud computing projects.

9.1 Amazon EC2

We chose Amazon Elastic Compute Cloud as our frame of reference for commercial cloud computing. It is a web service that is designed to make web-scale computing easier for developers. Developers can resize their applications as they wish, making EC2 a very flexible platform. It uses a web service interface which allows capacity configurations to be made. This means control over computing resources and Amazon's computing environment. Amazon gives storage and computing resources on a pay-per-use basis. It also charges for data transfers in and out of the storage. The charging rates as of the time this paper was written were:

- $0.10 per GB for all the data transfer in the storage system.

- $0.17 per GB for all the data transfer out from the storage system.

- $0.10 per CPU-hour for the use of its compute resources.

- $0.15 per GB-month of data stored. (Amazon Elastic Compute Cloud 2009.)

On their own, these numbers do not provide any perspective if EC2 is financially reasonable. We need to put the costs in contrast with a VC project in order to get the actual results. Also the size of the project, in the means of CPU and storage requirements, affect where it could be used most efficiently.

It is proven on Amazon's EC2 that "for a data-intensive application with small computational granularity the storage costs are insignificant as compared to CPU costs" (Deelman, Singh, Livny , Berriman & Good 2008, 11). The hourly CPU usage costs should be kept in mind when planning to deploy project on EC2. With VC systems one does not have to worry about CPU usage costs but it is not certain that the computing resources are available at all times. EC2 on the other hand has always resources waiting to be scavenged. This difference needs to be put in proportion with the scenario at hands.

9.2 Performance equivalence: Cloud and volunteer computing systems

Finding the differences we have to compare the performance between Cloud and VC systems. We require knowledge on how many hosts does a VC project require in order to provide equivalent computing power to one dedicated EC2 instance. This is important because it helps us to outline which one excels cost-beneficially. (Finding out this ratio is so long process that we don't have the time to do it.) Fortunately, Kondo et al. (2009, 3), made this cloud equivalence ratio. They find out that one small EC2 instance is equal to 2.83 volunteer hosts.

Kondo et al. got the result by first calculating the average FLOPS that each host contributes. Using the statistics from SETI@home they found out that the mean is 0.539 GigaFLOPS per host. Then they ran the Whetstone benchmark by means of the BOINC client on an EC2 small instance, and got the result of 1.528 GigaFLOPS on a single core processor. Calculated from this the ratio is approximately 2.83 volunteers/EC2 small instance.

We read the statistics from Renderfarm.fi in order to find the average rate of registered hosts per day. Beta version was launched in early July 2009 and at the end of September there were 612 hosts. In an ideal scenario the throughput could be about 2.3 TeraFLOPS, if each host output is on average 3.773 GigaFLOPS. We divided the amount of hosts by the days passed and got the mean rate of registration to be about 7 hosts per day. We divide the average registration rate (7) with the cloud equivalence (2.83), resulting in 2.5 needed cloud instances per day. We find that with this registration speed it would take almost 40 days on Renderfam.fi to reach equivalent cloud computing power of 100 nodes (150 GigaFLOPS). Furthermore, to achieve 1000 cloud nodes equivalence it would take over a year on a VC platform with this registration rate. Although 1000 cloud nodes would be very expensive on the long run. These results are indicative because they are based on the example computer that Kondo et al. (2009, 3.) used. Figure 7 shows how much time must past on a new VC platform before certain performance on cloud nodes can be achieved. It is calculated on the average registration rate per day on Renderfarm.fi.

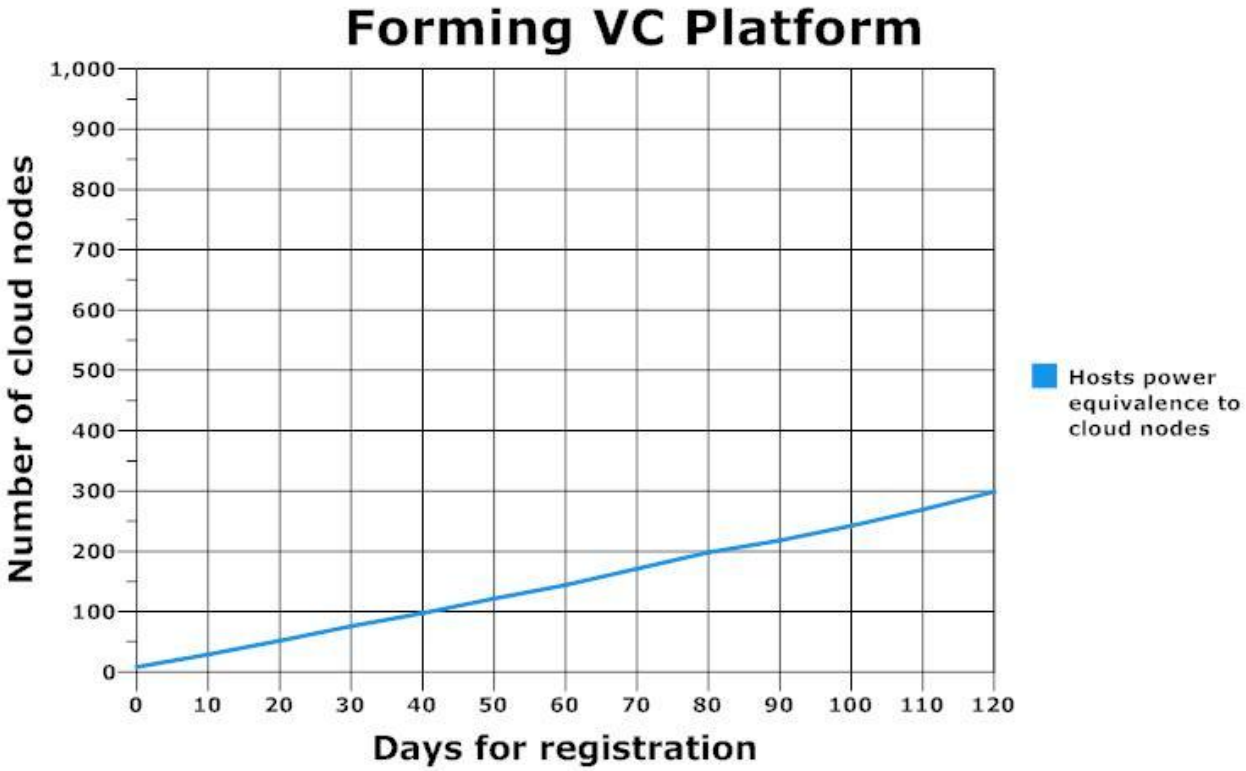


Figure 7. VC hosts vs. cloud nodes performance. (Kondo et al. 2009, 3)

9.3 Volunteer computing project costs

A VC project's expenses consist of different factors than of a cloud computing project. On a VC project, the main costs come from staff salaries and hardware. Employees are needed for programming and installing the server software, maintaining the hardware and recruiting new volunteers. Start-up costs for hardware are considerable depending on the size of the project. Kondo et al. (2009, 5), inspected the costs of two VC projects, SETI@home and XtremLab. Both projects were created in North-America, so the costs are calculated in USD. XtremLab was a small project that was cancelled in 2007 and the project's financial data was gathered afterwards. We concentrate on XtremLab because it was a small-scale project, thus making it more suitable point of comparison.

XtremLab was a BOINC project created to measure the available computing resources for large-scale VC projects. The measurement results were analysed in order to improve the design and implementation of current systems, such as BOINC. XtremLab had roughly about 3000 active hosts. The majority of the costs on XtremLab came from administrator/programmer salaries. Start-up costs on a server with RAID storage server were about 3000USD/month. Total costs were about 5000USD/month. Network usage would have increased the total monthly costs but it was covered by a University. Figure 8 illustrates the total costs on cloud (EC2 100n, EC2 1000n), and on a VC system (XtremLab). EC2 100n and 1000n means how many

nodes are in use on the Amazon cloud. XtremLab had approximately 3000 volunteers. (Kondo et al. 2009, 6)

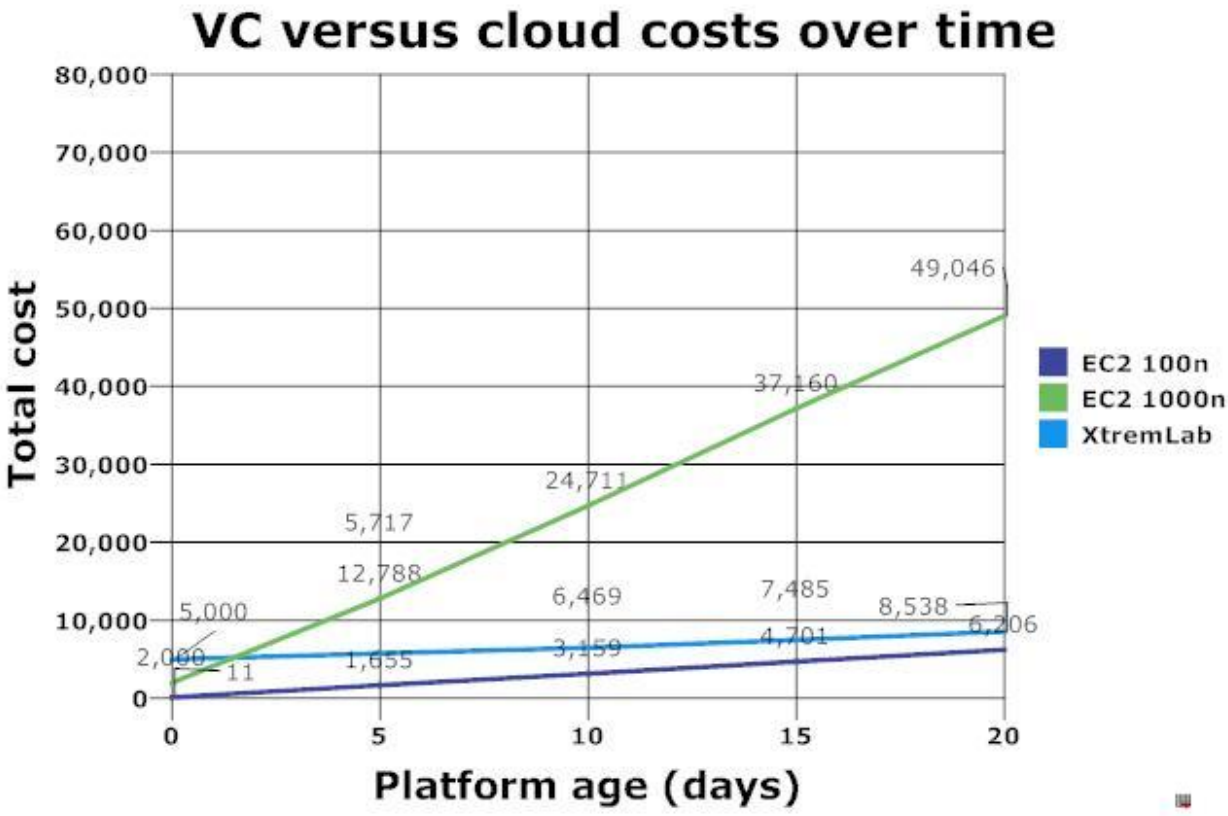


Figure 8. Total cost of cloud versus VC over time (Kondo etc. 2009, 6)

Kondo etc. also investigated the possibility to deploy a VC server on EC2 to reduce the start-up costs. They found out that it would lower the costs between 40-95% depending on the resource usage. Kondo etc. made a conclusion that hosting a server on a cloud is cheaper than on one's own if bandwidth needs do not exceed 100Mbit and storage needs are under 10 TB. So this kind of hybrid would work if the project is small enough, thus keeping the network traffic relatively low.

We can see from the figure 8, that when Amazon cloud is using 100 nodes for output power, it remains always the cheapest choice. If the computing power needs to increase, EC2 rapidly grows more expensive, as we can see from the 1000 nodes version. EC2 100n has pretty much the same expenses than XtremLab so which one is better depends on the resources required. If a volunteer computing project becomes bigger than XtremLab it is not wise to have it hosted on a cloud because the costs will increase exponentially. We cannot reflect these results directly on our scenario but it is clear that it would not be wise to use VC on our local computing network.

Extra computing resources on Renderfarm.fi are not necessary at the moment. But there are signs that this situation might change in the future. When people start to be more aware of this service and its possibilities, the need for more computing power comes to the picture. Keeping this in mind it is important that Renderfarm.fi would have the capacity to provide sufficient resources at all times. Hopefully the locally working render farm that we are creating could be used if this scenario comes current. Better even, if it could provide some monetary benefit for the service.

10 Distributed rendering in a closed environment

In the next sections we will go through different technologies for doing distributed rendering in a private network. These platforms are more simplified for the developer than many volunteer computing based platforms such as BOINC. This is because the network is build with computers in a controlled environment, making it secure. The computing itself is based on master and slave system. We will briefly explain the concept of each platform and show them in action. We also compare the performance differences with a test animation. Installation instructions for each platform are in the attachment section at the end of this work.

10.1 Yadra

Yadra (Yet Another Distributed Rendering Application) is an open source tool which enables Blender files to be rendered on configured network of computers. The biggest reason why it caught our attention was that the creator Oliver Schulze mentioned on his Website that Yadra could take advantage of the YafaRay renderer (look section 4.2.2). This is important because at the moment BURP does not support the use of external renderers. Regardless of many attempts we were not able to make Yadra work with YafaRay which was very disappointing. We think it is because of a different version of python script that Blender uses that makes it incompatible with Yafaray. (Schulze 2008.)

Yadra is a standalone program that can be used safely in a closed environment to do all the rendering work. It excels in a scenario where a lot of computers are available to be used as slaves. For example, a University could provide many computers for creating a large computing network. We should not ignore it as one possible tool to be adapted in Renderfarm.fi.

Yadra is very easy to setup and it is platform independent, meaning it can be used on almost every operating system. This is because it is programmed with java and no compiling is necessary for the different platforms. The system is based on master and slaves that communicate through a secure connection. Upon installation, one defines a password for the master and the same password is applied for every slave to connect to the master. When the connection

is established the master can start sending jobs to the slaves which then process them and send back the completed parts. Yadra is very simple to build and manage which makes it very efficient for rendering Blender animations. (Schulze 2008.)

Yadra comes with Web interface where one can keep track on job progress. Web interface allows jobs to be paused, stopped and deleted. It is also possible to see all the slaves, the job queue and the jobs the slaves are currently working on. A nice feature is the preview option of the work-in process in the selected output forms, such as .JPEG and .PNG. Finally, when the slaves have finished the jobs, a user can download the final result. Below is a screenshot of jobs in process from Yadra's Web interface. (Schulze 2008.)

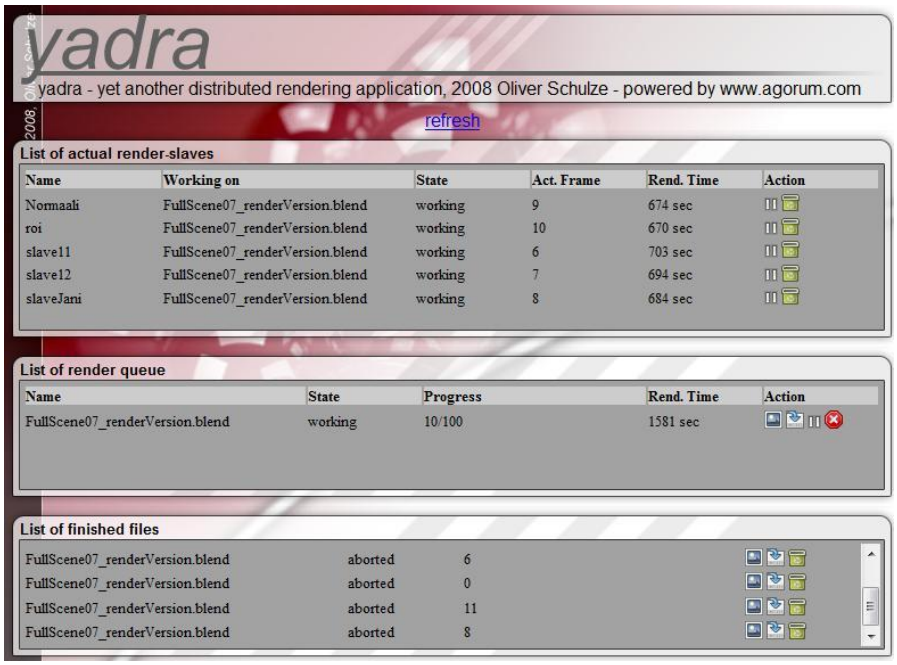


Figure 9. Yadra's Web interface

10.2 Loki Render

Loki Render is a management system for rendering Blender animations. It enables distributed network rendering in Local Area Networks (LAN) in the same way as Yadra renderer. Loki Render is also a platform independent so it can be used in Windows, Mac OS X and Linux operating systems. Furthermore, it can handle multiple operating systems on one project. Loki Render is written in C#.net language. It is simple to install, but requires installation of the Mono framework (cross-platform .net development tool) and Blender. Loki Render does not work correctly if the same version of Blender is not installed in every computer. (http://loki-render.berlios.de/index.php)

Loki Render is also based in master and grunt clients (in Yadra renderer grunts are called

slaves). Yadra requires a master computer, although a slave can exist in the same computer. A Loki Render user can decide which computer works as a master every time when he or she opens the application. This feature enables users from different computers to put jobs in queue.

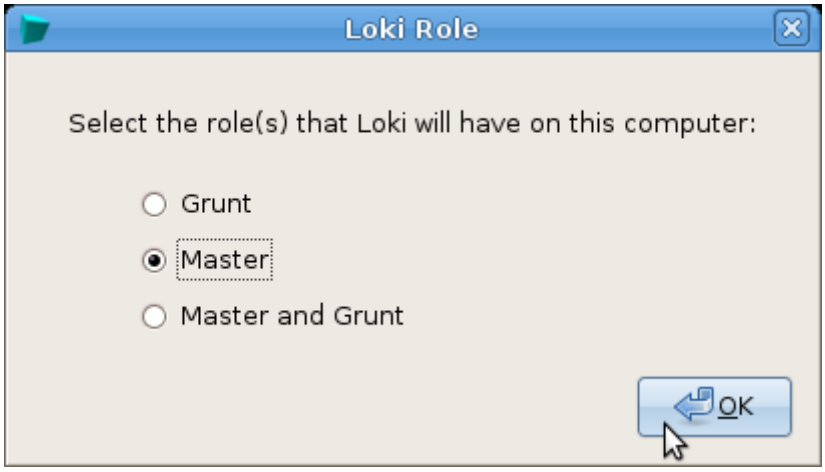


Figure 10. Loki logon view

Loki render is quite interesting project for us, since it is open source and completely free of charge. It is easy to set up and it is platform independent. Nevertheless it does not work with any external renderers which are crucial for our project. The graphical user interface in Loki is more primitive compared to Yadra and it also lacks features like preview of images during the rendering. We will cover more specific evaluation of Loki and Yadra in the next chapter.

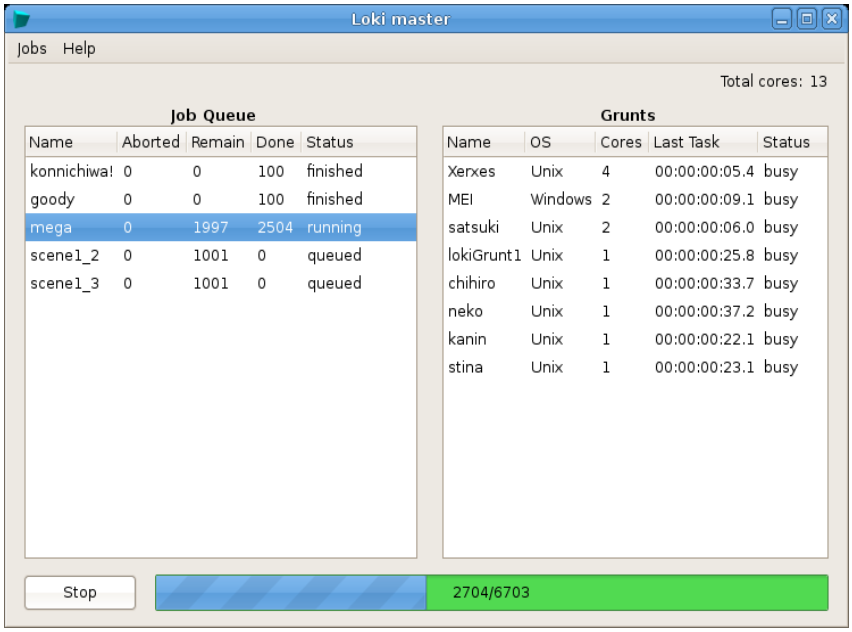


Figure 11. Loki Master view

10.3 Farmerjoe

Farmerjoe is also an open source tool for doing distributed rendering in a controlled environment. It was originally created because the developer had so many problems with the installation of DrQueue, which led to the decision of building his own platform. Farmerjoe is written in Perl and compiled into .exe so users do not have to install Perl on their computers. Like Loki Render and Yadra, Farmerjoe uses masters and slaves. The only exception is that user does not have to install the slaves on every computer he uses. Farmerjoe is one of the first distributed rendering systems for Blender that use frame based rendering and bucket rendering. Bucket rendering means that one image can be rendered in parts. Farmerjoe supports the mainstream operating systems such as Windows, Linux and OS X and it can be configured to work cross-platform. For creating a working environment, one needs Blender, Farmerjoe and Imagemagick. (Farmerjoe – The render Farmer 2006.)

Figure 12. Farmerjoe's working process

10.4 DrQueue

DrQueue is an open source distributed rendering manager. It is a widely used tool in the field of 3D graphics. It has a good reputation in the movie industry. For example, some of the scenes in the full length feature film Pirates of the Caribbean were rendered using DrQueue. Also the well-known Blender Foundation movie Elephant's dream was fully rendered using DrQueue manager. DrQueue supports many 3D-animation software including commercial ones. Blender, Maya, 3DStudioMax and StudioTools / ImageStudio are all currently supported. Similar to Farmerjoe, DrQueue can be installed on various operating systems such as Windows, Linux and OS X. It can also be modified to work as a cross-platform solution. For example, the master computer can be operated on OS X and the slaves can do all the calculation work on Windows. (DrQueue background 2009.)

DrQueue's support for multiple rendering engines makes it an appealing counterpart to BURP. Before testing any of the grid computing platforms, DrQueue would be the choice to be used in Renderfarm.fi, based on its capacity to reach bigger user group.

Installing DrQueue can be demanding because of the scripts that enable its use in various platforms. Originally DrQueue was meant to be used in a UNIX environment, Linux and OS X being the main operating systems. This fact placed difficulties on our attempt to build it on a Windows system. Instructions on how to install DrQueue on Windows XP PRO exist, but since XP is not the latest version of Windows, the instructions are outdated. After being unsuccessful at installing the program in Windows, we tried it on Linux and Mac OS X. Even after installing all the different programs required by DrQueue on a UNIX platform, we were unable to get it fully working. Since we are not experts in UNIX based operating systems, we decided to go back to trying a Windows based installation.

After many days we still could not make DrQueue work. On Windows XP we got the master computer working but it was unable to get connection with Vista slaves. We then put Windows Vista OS running as the master computer and tried to connect it with slaves also running on Vista. This time problems related with Microsoft.NET framework appeared. A new version of .NET could not work with DrQueue's IPC services. Every time we tried to get the service running, it immediately shut down. After searching through many internet forums, we came to the conclusion that we were not alone in our inability to get DrQueue running successfully.

DrQueue requires a lot of effort from the builder when compared with other render farm technologies. With Windows operating system it should be relatively easy to install, but all the external software required and the version incompatibility issues makes it very hard to

understand. Nevertheless, we think that somebody with UNIX skills can build DrQueue, which then would be the best solution for Renderfarm.fi.

11 A comparison of distributed rendering managers in a LAN environment

The render farm managers presented in the previous chapter all share the same basic idea of dividing render jobs into smaller pieces of data and processing them on a grid of computers. How small these work pieces can be made depends on the manager used. In the managers we tested the size of the work piece at smallest varies from one frame to a quarter of a frame. Smaller work pieces enable more calculation power for one frame. This makes it possible to render large frames faster, assuming that there is an optimal amount of slaves in a grid. All these technologies have one master computer which handles at least the work distribution and scheduling.

Usually there is also a web interface that enables previewing completed work and gives the ability to control the rendering process. In each manager the behaviour of slaves is quite similar and it seems that the biggest differences between these are found from the master accounts, and the way they execute render jobs and manage render engines.

There are a large amount of installation guides for building a rendering environment using the managers we presented in the previous chapter. All the managers except Drqueue were relatively simple to install and configure to our set of computers. To implement a render farm environment, one must know how to use Blender and have at least basic knowledge of Local Area Networks (LAN).

11.1 Test environment

For implementing the test environment we used five (5) similar PC's with following statistics:

- Operating system: Microsoft Windows Vista, 32 Bit.
- Memory: 4GB of memory
- CPU: Dual core 2.33 GHZ.

We planned to implement a cross-platform installation for each manager using the Windows Vista, Mac OSX and Linux Ubuntu operating systems, but since Yadra and Farmerjoe do not support cross-platform rendering we decided to abandon the Mac and Linux installations.

All the test computers were in the same local area network. For setting up Loki Render and Farmerjoe a shared folder in the Windows installation is required. We had to map the required network drives every time the test computers were restarted, because of a lack of a

central DNS (Domain Name Service) server, resulting in the need to assign IP addresses manually. There were also firewalls in the test computers so we had to make exceptions on their rules to get the connection working between the test computers.

11.2 Test scene

For test rendering, we used a turntable animation made by Jani Lintunen. The video was originally made for Alternative party 2009 as a part of Renderfarm.fi promotion. The outcome of the animation lasts 80 seconds, and it contains 2000 frames with the speed of 25 FPS (Frames per Second). For our tests we decided to shrink the length of the video for only 20 seconds, this means that we rendered only the first 100 frames. We wanted to choose an animation which does not contain baked textures, fluids or anything that could bring problems when testing different rendering technologies. It is noticed that there has been problems on Renderfarm.fi with scenes which are not frame independent (A frame depends on the computation made in the other frame). The test scene still contains some complex features, like camera movement and lots of particle based items. The size of the .Blend file was 138 MB. (Tuomisto 2009c.)

The promotional video of Renderfarm.fi was chosen to be the test scene because it had already been used in a real life situation. The quality and the length of the animation met the requirements of, for example, a television ad. The example animation is so computationally intensive that it could clearly benefit from being rendered on a distributed rendering environment. Rendering of the example scene took 5125 minutes (85 hours) to complete when using only one of our test computers and the Blender internal render engine (the specs of the test computers mentioned in previous chapter). In our opinion the test scene is a good example of the types of videos that could be rendered on the closed side of the Renderfarm.fi.

Figure 13. Single frame from the test scene

11.3 Test results

Building the grid computing environments was a relatively simple task, excluding DrQueue, which we could not get running successfully. The time used for installation on figure 15, is the time it would take if everything goes according to plan. For example, Loki Render is very fast to setup as long as one knows that the master computer can only have one network adapter running. It took a while for us to realise that, because it is not mentioned anywhere.

Yadra and Farmerjoe were installed without any major problems and we got the test environment up and running in no time. Size of the packages is small, making it a fast download even with a low bandwidth connection. All the installations were made on Windows Vista, which proved to be very functional, regardless of our initial doubts. Except DrQueue that refused to work with the .Net 3.5 framework that comes with Vista as a default. A common factor between all the platforms is that Blender is required as part of the installation process. This is obvious for most of the people who are used to work with rendering technologies, but for non-experienced persons this should be mentioned.

	Yadra	Loki Render	FarmerJoe	DrQueue
Size of the installation file	*7,20 MB*	*0,13 MB*	*7,11 MB*	*2,9 MB*
Other applications needed to work	*Blender*	*Blender*	*Blender*	*Blender*
Time used for installation	*less than 1h*	*10 min*	*30 min*	*several days*
Platforms supported	*Windows, Linux*	*Windows, Linux, Mac OSX, Solaris*	*Windows, Linux, Mac OSX*	*Windows, Linux, Mac OSX, Irix, FreeBSD*

Figure 14. Comparison of the installation between distributed render techniques

The time used for rendering the test scene on each grid was very similar, approximately four and a half hours. There were no big differences in the fastest and the slowest frame either. Loki Render and Yadra were the only ones that offered results for individual frames. Time used on each frame vary only some seconds. Yadra's web interface did not give any information about the time used for rendering one frame. It only gave the overall time for the whole scene.

Quality of the output was similar as expected because of all render pipelines used the Blender internal render. We compared the .jpeg output images on the same screen and came to the conclusion that there were no visible differences in the results. No errors were found after inspecting all the frames, so the technologies seemed to perform perfectly.

Loki Render and Farmerjoe support bucket rendering and both are also capable of handling physics simulation data. These facts make these two the best out of the three tested ones. DrQueue would have been our choice if we had succeeded to make it work. It has such a huge potential, that it would be the obvious platform to be integrated with Renderfarm.fi. This is work to be done in the future.

	Yadra	Loki render	FarmerJoe	DrQueue
Time used for rendering 100 frames	*4h 41min*	*4h 30 min*	*4h 30 min*	
Fastest frame	*Undefined*	*12 min 54 sec*	*12 min 49 sec*	
Slowest frame	*Undefined*	*14 min 35 sec*	*14 min 53 sec*	
Quality of the render, any mistakes?	*No errors*	*No errors*	*No errors*	
Supported output-formats	*TGA, IRIS, HAMX, FTYPE, JPEG, IRIZ, RAWTGA, PNG, BMP*	*PNG*	*jpeg, PNG, BMP, Targa*	
Supported render engines	*Blenders own, YafRay**	*Blenders own*	*Blenders own*	
Support for tile rendering (bucket rendering)	*No*	*Yes*	*Yes*	
Physics simulation data supported (soft-body, particles, cloth, etc..)		*All supported, except fluids*	*All supported*	

Figure 15. Performance comparison of the distributed render techniques

12 Conclusion

In this thesis we studied different ways to use spare computing power for rendering 3D-images. Our work covers multiple techniques for creating a render farm in a local area network, but also techniques for publicly distributed rendering. To fully understand opportunities of commercial rendering services, we explored existing volunteer based rendering services and benchmarked their assets. We studied BOINC more closely because it is the backbone of the current Renderfarm.fi service. The original hypothesis was that it would be a good option for the closed side of the Renderfarm.fi. After comparing cloud computing and VC, studying the structure and the technology behind BOINC and discussing with Janus Kristensen (creator of BURP) we were certain that there are more efficient ways to achieve our objective. One

of the technical reasons which caused us to leave the combination of BOINC and BURP out of our tests, was the unnecessary complexity of the job delivery. After the decision to abandon BOINC and BURP we focused on more lightweight distributed rendering techniques.

The bottom line in our work was to use only open source based solutions. This objective was surprisingly easy to achieve, since there are great variety of open source rendering software available, both render engines and distributed rendering techniques.

We created a micro-sized implementation environment for our rendering tests. The environment was placed in Laurea's network laboratory, where we were able to test the performance of these techniques in an environment which can be considered to be equivalent to a real life office building or school. The only platform that we had problems with was DrQueue. We simply could not get DrQueue working which was unfortunate because it had the most promising technical specifications. All the technologies we tested successfully were nearly equal comparing on performance. There are still some minor differences in their features. Farmerjoe is the fastest to initialize, Loki Render has most supported operating systems and Yadra has the best control over rendering process, thanks to its web based graphical user interface. Excluding DrQueue, none of the previously mentioned techniques had an external render engine support at the time. Its technical superiority and support for multiple render engines, including commercial ones, are the reasons why we think DrQueue should be integrated to Renderfarm.fi in the future.

Two major questions remain with Renderfarm.fi; how to attract more Blender users to assign jobs to the system and more importantly, from a commercial perspective, would anyone be willing to pay for these extra resources? BURP is still lacking some features which could make it more appealing to companies and the public. For example, many effects on Blender cannot be used with BURP. This is because BURP does not support any external renderers, such as Luxrender or YafaRay. This is big deficiency in the structure of BURP, because many Blender users need extra effects in their animations to get the best result. Making these improvements to BURP was not the focus of this thesis, but we think they are questions that require answers in order for Renderfarm.fi to achieve success.

Suggested improvements:

- Connecting Renderfarm.fi with a closed render environment
- Successful installation and integration of DrQueue with Renderfarm.fi
- Testing cross-operating system support in distributed rendering environments
- Creating a graphical user interface for the closed rendering environment in Renderfarm.fi

References

Anderson, D. 2004. BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing. November 8, 2004, Pittsburgh, USA.

Anderson, D., Korpela, E., & Walton, R. 2005. High-Performance Task Distribution for Volunteer Computing. First IEEE International Conference on e-Science and Grid Technologies. 5-8 December 2005, Melbourne

Anderson, D. & McLeod, J. 2007. Local Scheduling for Volunteer Computing. Workshop on Large-Scale, Volatile Desktop Grids (PCGrid 2007) held in conjunction with the IEEE International Parallel & Distributed Processing Symposium, Long Beach.

Anderson, D. & Reed, K. 2009. Celebrating Diversity in Volunteer Computing. Hawaii International Conference on System Sciences (HICSS), January 5-8, 2009.

Deelman, E., Singh, G., Livny, M., Berriman, B. & Good, J. 2008. The Cost of Doing Science on the Cloud: The Montage Example. USC Information Sciences Institute, Marina del Rey, CA. University of Wisconsin Madison, Madison, WI. Infrared Processing and Analysis Center & Michelson Science Center, California Institute of Technology, Pasadena, CA. Infrared Processing and Analysis Center, California Institute of Technology, Pasadena, CA.

Estrada, T., Taufer, M. & Anderson, D. 2009. Prediction and Analysis of BOINC Projects: An Empirical Study with EmBOINC. To appear, Journal of Grid Computing.

Hevner, A.R., March, S.T., Park, J., Ram, S. (2004) Design Science in Information Systems Research, MIS Quarterly (28:1), pp 75-105.

Kondo, D., Javadi, B., Malecot, P., Cappello, F. & Anderson, D. 2009. Cost-Benefit Analysis of Cloud Computing versus Desktop Grids. 18th International Heterogeneity in Computing Workshop, May 25 2009, Rome.

Shirley, P. & Morley, K. 2003. Realistic ray tracing. Salt Lake City. A K Peters, Ltd.

Søttrup, C. & Pedersen J. 2005. Developing Distributed Computing Solutions Combining Grid Computing and Public Computing. M.Sc. Thesis, Department of computer science, University of Copenhagen, 1st March, 2005.

Wright, D. 2006. BOINC and Condor – Scavenging the Scavenger. wright@cs.wisc.edu, UW-Madison Condor Project (www.condorproject.org)

www-references

Administrative web interface. 2007. University of Berkeley. Last viewed 8.12.2009 http://boinc.berkeley.edu/trac/wiki/HtmlOps

Amazon Elastic Compute Cloud. 2009. Amazon Web Services. Last Viewed on 6.9.2009. http://aws.amazon.com/ec2/

BOINC scheduler. 2006. University of Berkeley. Last viewed on 23.8.2009. http://www.boinc-wiki.info/Scheduler

BOINC stats. 2010. BOINC stats. Last viewed on 10.2.2010. http://fi.boincstats.com/stats/boinc_host_stats.php?pr=bo&st=0

CG community survey. 2009. CGenie Last viewed on 7.12.2009.
http://cgenie.com/component/content/article/47-articles/104-cg-community-survey-upgrades-09.html

CGenie survey. 2009. CGenie. Last viewed on 7.12.2009.
http://cgenie.com/component/content/article/47-articles/104-cg-community-survey-upgrades-09.html

DrQueue Background. 2009. DrQueue website. Last viewed on 22.11.2009.
http://www.drqueue.org/cwebsite/about_drqueue.php

Estevez, A., Williamson, C. & Matthews, J. 2009. What is YafaRay. Last viewed on 2.12.2009.
http://www.yafaray.org/documentation/userguide/introduction#history

External renderers. 2009. Blender foundation. Last viewed on 19.11.2009.
http://www.blender.org/download/get-blender/external-renderers/

Farmerjoe – The render Farmer. 2006. lobo.nz@gmail.com. Last viewed on 15.11.2009.
http://farmerjoe.info/farmerjoe/README.html

Features. 2009. Blender foundation. Last viewed on 7.12.2009.
http://www.blender.org/features-gallery/features/

Get Blender. 2009. Blender foundation. Last viewed on 6.12.2009.
http://www.blender.org/download/get-blender/

Goals. 2009. LuxRender. Last viewed on 25.11.2009. http://www.luxrender.net/v/goals

High Throughput Computing. 2006. University of Wisconsin. Last viewed on 15.10.2009.
http://www.cs.wisc.edu/condor/htc.html

History. 2009. Blender foundation. Last viewed on 7.12.2009.
http://www.blender.org/blenderorg/blender-foundation/history/

How it works. 2009. vSwarm. Last viewed on 14.10.2009. http://www.vswarm.com/how-it-works.html

Import & Export. 2009. Blender foundation. Last viewed on 19.11.2009.
http://www.blender.org/download/python-scripts/import-export/

Introduction to LuxRender. 2009. LuxRender. Last viewed on 26.11.2009.
http://www.luxrender.net/wiki/index.php?title=Introduction_to_LuxRender

Kristensen, J. 2007. CATS. Last viewed on 14.11.2009.
http://burp.renderfarming.net/forum_thread.php?id=740

Le Ferrand, W. 2009a. Corefarm.alpha. Last viewed on 18.10.2009.
http://www.corefarm.org/

Le Ferrand, W. 2009b. naclgrid 0.0.1. Last viewed on 13.11.2009.
http://groups.google.com/group/native-client-discuss/browse_thread/thread/3693b4926f954a9a?pli=1

Link of the week – BURP. 2008. International Science Grid This Week. Last viewed on 10.12.2009. http://www.isgtw.org/?pid=1000986

Luxrender Features. 2009. LuxRender. Last viewed 16.9.2009.
http://www.luxrender.net/v/features

Schulze, O. 2008. Yadra – News. Blender Publications. Last viewed on 16.8.2009. http://blendoli.googlepages.com/yadra

Security issues in volunteer computing. 2009. University of Berkeley. Last viewed on 28.8.2009. http://boinc.berkeley.edu/trac/wiki/SecurityIssues

System requirements. 2009. vSwarm. Last viewed on 14.10.2009. http://support.vswarm.com/wiki/ClientSystemRequirements

Tuomisto, J. 2009a. About. Last viewed on 12.10.2009. http://www.renderfarm.fi/page/about)

Tuomisto, J. 2009b. Licensing of rendered output. Last viewed on 12.10.2009. http://www.renderfarm.fi/page/licensing

Tuomisto, J. 2009c. Limitations of publicly distributed rendering. Last viewed on 12.10.2009. http://www.renderfarm.fi/page/limitations-publicly-distributed-rendering

Tuomisto, J. 2009d. Terms of use. Last viewed on 12.10.2009. http://www.renderfarm.fi/page/terms-of-use

Volunteer computing. 2009. University of Berkeley. Last viewed on 10.9.2009. http://boinc.berkeley.edu/trac/wiki/VolunteerComputing

Work generator daemon. 2006. University of Berkeley. Last viewed on 22.8.2009. http://www.boinc-wiki.info/Work_Generator_Daemon

Table of Figures

Appendix

Attachment 1 How to install Yadra master

Yadra master and slave installation files can be downloaded from http://sourceforge.net/projects/yadra/. There is a bundle packages for Windows and Linux, but also platform-independent package can be used. We used Windows Vista Home Basic as a platform for our test so we decided to download Windows bundle package as it already contains Java-Virtual-Machine (JVM). The same installation package can be used both for master and slave.

How to install master:

When installing the Yadra master account the first thing to do is extract the Yadra zip file in for example root of C: drive. (C:\\yadra).
Setting up the master for Windows happens via command prompt. All the commands must be done under the Yadra folder, in our case (C:\\yadra\yadra).

Step 1: User must run the configuration file using command: "config.bat"

After running config.bat user must go through few steps to configure Yadra master to fit his/hers environment.

Step 2: Installation wants to know if we are setting up a master account: "Is this a Master [y/n]?"
Hit "y" and then "enter".

Step 3: Master computer has to have a unique name. User can type the name, or just hit enter when name is set to be Master.

Step 4: User must set a folder where all the blend files and rendered images goes. In our case: C:\\yadra\work.

Step 5: Yadra needs to set up a password for connection between master and slaves. In our case password is: testi12.

Step 6: User must set a network interface address to listen on. Setup offers address 0.0.0.0 when Yadra listens on all the network adapters. According to the Yadra documentation this should work most of the times, if there is no multiple network adapters. In our case 0.0.0.0 address didn't work, so we put master computers own ip-address: 10.8.6.2.

Step 7: There must be one network port opened for Yadra where slaves can connect to the master. Configuration offers

port 2206. User must make sure that port is not already in use and its not blocked by the firewall.

Step 8: User can decide whether or not to run web server for showing states of jobs and for example showing rendered images.

Step 9: Webserver needs one open network port. Setup offers port 8080. User must make sure that port is not in use. The port 8080 is quite often used in different applications, for example Skype and Xampp use this port if not configured otherwise. To connect the webserver open internet browser and type address in this form: http://masteradress.network_port. In our case address for the webserver is http://10.8.6.2:8080

After configuring the network port, the Yadra shows following settings.

Your settings:
This is a master

PassPhrase: testi12
MasterAddress: 10.8.6.2
MasterPort: 2206
WorkPath: c:\yadra\work
WebServerPort: 8080

Step 10: Confirm your settings, if correct.

Step 11: Setup is ready, now user must run master application. Type your master "applications_name.bat" (see step 3) in our case, its master.bat

You should see something like this:
yadra - yet another distributed render application Copyright 2008 Oliver Schulze
Loading config: "C:\yadra\yadra\Master.config"

2008-01-08 15:09:12.729::INFO: Logging to STDERR via org.mortbay.log.StdErrLog
2008-01-08 15:09:13.104::INFO: jetty-6.1.7
2008-01-08 15:09:13.464::INFO: Started SocketConnector@0.0.0.0:8080
Master is ready for connections on port: 2206

Attachment 2 How to install Yadra slave

Yadra master and slave installation files can be downloaded from http://sourceforge.net/projects/yadra/. There is a bundle packages for Windows and Linux, but also platform-independent package can be used. We used Windows Vista Home Basic as a platform for our test so we decided to download Windows bundle package as it already contains Java-Virtual-Machine (JVM). The same installation package can be used both for master and slave. Slave installation is quite similar to master installation, with only few differences. It is necessary to install master client before slave client.

How to install slave:

When installing the Yadra slave account the first thing to do is extract the Yadra zip file in for example root of C: drive. (C:\\yadra).
Setting up the slave for Windows happens via command prompt. All the commands must be done under the Yadra folder, in our case (C:\\yadra\yadra).

Step 1: User must run the configuration file using command: "config.bat"

After running config.bat user must go through few steps to configure Yadra slave to fit his/hers environment.

Step 2: Installation wants to know are we setting up a master account: "Is this a Master [y/n]?" Hit "n" and then "enter".

Step 3: Slave computer has to have unique name. We decided to name our slaves with unique number marking. Slave1, slave2, etc.

Step 4: User must set a folder where all the temporary blend files and rendered images go. In our case it is: C:\\yadra\work.

Step 5: Yadra needs to know password for connection between master and slave. In our case password is: testi12. (See attachment 1)

Step 6: User must type the same network port as master's network port. In our case port number is: 2206. Make sure that port is not already in use and it's not blocked by a firewall.

Step 7: setup asks path for Blender.exe. After typing it, setup checks path.

Step 8: User can decide how many CPU's or cores he or she wants to use for rendering images. Valid values are from 1 to 8. We used two threads.

Now setup shows your settings.

This is a slave
PassPhrase: testi12
MasterAddress: IP of Master
MasterPort: 2206
WorkPath: c:\yadra\yadra\work
Identification: Slave1
Blender-Executable: c:\program files\blender foundation\blender\blender.exe
Render-Threads: 2

Step 10: Confirm your settings, if correct.

Step 11: Setup is ready, now user must run slave application. Type your slave "applications_name.bat" (see step 3) in our case, it's: slave1.bat

Attachment 3 How to use Yadra

To render images using Yadra, there must be both master and slave accounts installed. At least master client must be running. Slave accounts can be started later. All the .Blend-files must be located in work folder of Yadra (in our case: C:\yadra\work\work).

Step 1: User must open a new command prompt window.

Step 2: User must go to the installation directory of Yadra. In our case it is C:\yadra\yadra.

Step 3: Start at least one slave, and master

Step 4: When using master, type: sendToQueue_NAMEOFSLAVE.bat c:\PATH_TO_BLEND_FILE\NAME_OF_BLEND.blend 1 150 JPEG. In our case command would be in following form: sendToQueue_SLAVE1.bat c:\yadra\work\work\testiframe.blend 1 100 PNG. Our work folder is located in C:yadra\work\work, the .Blend file is called as testiframe.blend and our animation starts from frame 1 and ends in frame 100.

Attachment 4 How to install Farmerjoe

To install Farmerjoe, user must download the installation file from: http://farmerjoe.info/index.pl?p=1. In this document we are only going to cover installation for Windows platform, installation manual for Linux and OS-X can be found from: http://farmerjoe.info/farmerjoe/README.html. When using the Farmerjoe there is no need to install slaves.

Step 1: Create a folder for Farmerjoe, in our case: C:\Farmerjoe.

Step 2: Share the Farmerjoe folder and give users right to view and edit files in that folder.

Step 3: Edit the farmerjoe.conf file. Farmerjoe configuration needs specific information about master computers ip-address, root directory, and the location of the Blender. This is how our configuration file looks like. In configure file there is also some settings for Linux and OSX, but there is no need to configure them if using only Windows.

# Master Server Configuration
port = 2006
master = 10.8.6.5 (masters ip-address)

jobs = jobs (location of jobs)
logs = logs

windows_root = r: (this is the name of our shared folder)
windows_blender = r:\bin\windows\Blender\blender.exe (this is the location of the Blender.exe)
windows_composite = composite

# Application server Configuration
appserver_port = 2007 (this is the port for web service)

Now Farmerjoe is ready for use.

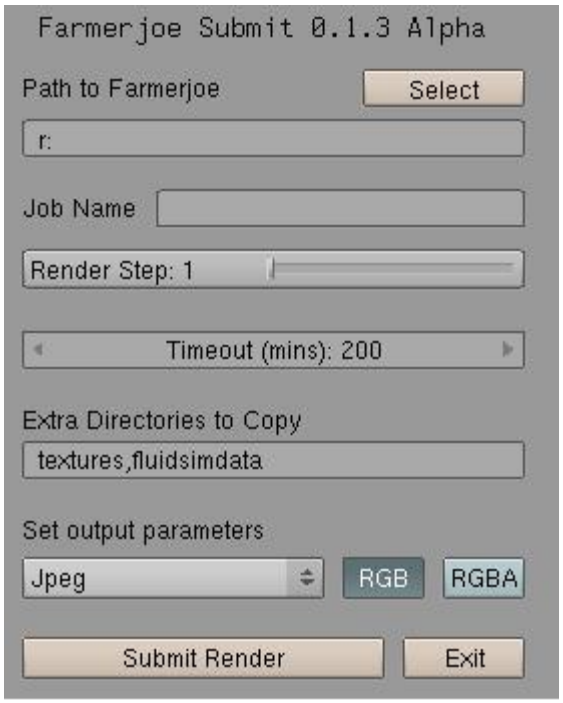
Attachment 5 How to use Farmerjoe

Queueing jobs to Farmerjoe happens trough Blender.exe.

Step 1: User must open text edit window in Blender.

Step 2: User must open and run "farmerjoe_submit.py" python script. (press alt+p to run the script)

Step 3: Blender opens Farmerjoe's submit window. To do the basic render user must choose the shared folder where Farmerjoe is installed, name the job, choose how many render steps there is, and choose the output format.



Step 4: Press "Submit Render" button.

Attachment 6 How to install Loki Render

How to install Loki render:

Since the release of Loki render 0.6 series there is no separate installation needed. One must download and unzip lokiRender_060.zip from: http://loki-render.berlios.de/index.php/download. Blender must be installed before the installation of the Loki render. Loki render sends broadcast messages to Grunts so there is no need to share any network drives like in other systems we tested. We also noticed that to enable Master computer to connect Grunts, user must disable all network adapters except the one Loki is using. This can be done from Control panel -> System -> Device manager.

In order to get the Loki render working user must go through few steps.

Step 1: After downloading and unzipping the downloaded file, user must start lokiRender_062.exe (Newest version of Loki at this point)

Step 2: Loki Render asks to set the path to the Blender.exe

Step 3: Loki Render asks which role user is going to use. Master, Grunt (Slave), or both at the same time. (Master can't do rendering, this is why user must choose both if it is desired to do rendering in Master computer.)

Now Loki Render is ready for rendering. To set up the Grunts user must follow the same procedure, but in step 3 choose the Grunt.

Attachment 7 How to use Loki Render

To put a job in a queue in Loki user must run Loki as a Master.

Step 1: User must click Jobs from the task bar
Step 2: Then click "New"
Step 3: Name the animation
Step 4: Set the path to the Blend.file
Step 5: Set the output directory (in our case the output directory was in c:\loki\renders)
Step 6: Set the first and last frame (in our case first frame was 1 and last frame 100)
Step 7: Enable tile rendering if rendering only one image
Step 8: Click "Save"
Step 9: Now user can start rendering by clicking "Start" button from the bottom of the page