Peer to peer rendering and computation

4th year dissertation

BY
Akilan Selvacoumar (H00281538)

SUPERVISOR
RYAD SOOBHANY

Declaration

I, Akilan Selvacoumar confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas,equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references used is included.

Signed: Akilan Selvacoumar

Date:

Abstract

This project focuses on running heavy tasks which a regular computer can’t run easily such as high spec video games, rendering 3D animations , protein folding simulations. In this project the major focus won’t be on the financial incentive part. A peer to peer network will be created to run tasks decentrally, increasing bandwidth for running tasks . To ensure the tasks in peer to peer network does not corrupt the server 0S (Operating System) they will be executed in a virtual environment in the server.

With the introduction of overlay protocols like IPFS (Inter Planetary File System) for websites and Libp2p, it is easier to create custom peer to peer applications. Using the existing developments of peer to peer frameworks, a peer to peer application can be created to do any type of rendering or computation in a distributed environment.

The main aim of this project will be to create a peer to peer network using Libp2p and Kademlia DHT (Distributed Hash Table) and distribute the rendering of graphical tasks. The user acting as the client will have total flexibility on how to batch the tasks and the user acting as the server will have complete flexibility to set how much computation he/she wants to provide to the client.

Table of contents

Chapter 1: Introduction

1.1 Motivation

Many of the users rely on our PC / Laptop or servers that belong to a server farm to run heavy tasks and with the demand of high creativity requires higher computing power. Buying a powerful computer every few years to run a bunch of heavy tasks which are not runned as frequently to reap the benefits can be inefficient utilization of hardware. On the other end renting servers to run these heavy tasks can be really useful. Ethically speaking this is leading to monopolisation of computing power like what is happening in the web server area [44]. By using peer to peer principles it is possible to remove the monopolisation factor and increase the bandwidth between the client and server.

1.2 Aim

This project aims to create a peer to peer (p2p) network, where a user can use the p2p network to act as a client (i.e sending tasks) or the server (i.e executing the tasks). A prototype application will be developed, which comes bundled with a libp2p library and a custom configuration which can execute docker containers or virtual environments across selected nodes.

1.3 Objectives

- Background review on peer to peer network, virtual environments, decentralized rendering tools and tools to batch any sort of tasks.

- Creating p2p network
- Customise routing and tasks distribution algorithm
- Server to create a containerised environment
- The client node to run tasks on Server containerised node
- Implement prototype application

1.4 Organization overview

This paper will will be in the following format:

Chapter 2. Literature review: This chapter looks into the basics of peer to peer networks, Strategies which can be used to execute tasks in a decentralized manner. The related works section helps to understand the best strategies to use to build this project.

Chapter 3. Requirement analysis: This chapter looks into the functional and nonfunctional requirements of this project.

Chapter 4. Evaluation: This chapter looks into different strategies to evaluation if the project is working as intended and helps log the progress of the project.

Chapter 5. Project management: This chapter looks into the risk management, Gantt chart and legal issues of the project.

Chapter 6. Conclusion: This chapter gives a conclusion based on the current progress of the project.

Chapter 2: Literature Review

This chapter will cover the core concepts of peer to peer networks, containers, SSH servers , GPU virtualization tools , display servers and related work in this area.

2.1 Background

2.1.1 Client to Server model

Client to server model refers to a node or nodes serving service/services and another node or nodes using that service/services . The client and server can even reside on the same system. [2]

2.1.2 Peer to Peer Systems

Peer to peer network is a distributed application architecture that assigns tasks or workloads between peers. Peers make a portion of their resources available with needing to be controlled by a centralized server. A peer to peer network is designed around the notion that peers can act as a client or server. A simple example would be the client server model mostly popularly known as FTP (File Transfer Protocol) where the client initiates the transfer and the server satisfies these requests. [1]

2.1.3 Distributed Hash Table

Distributed Hash Table (i.e DHT) is used to coordinate and maintain metadata about peers in a network. Peer to peer networks are an overlay on top of the current topology hence the user can think of DHT as the routing table for the overlay. A really popular DHT is called the Kademlia

DHT. The reason for its popularity is due to its efficient lookup which is around 20 hops in around 10 million nodes , resistance to various attacks by preferring long lived nodes. [3]

2.1.4 Content Delivery Network (CDN)

CDN is a geographically distributed data center which works together to provide fast delivery of internet content. Benefits of CDN include improving time of downloading data, reducing bandwidth cost. [4] P2P networks give a whole new better use case of CDN as they can act as a CDN itself. For example, Netflix uses IPFS[3] to distribute their containers within their clusters. "Leveraging IPFS as a peer-to-peer CDN lets nodes inside Netflix's infrastructure collaborate and seed common pieces to neighboring nodes, helping make container distribution faster". [5]

2.1.5 Virtual Machines (VM)

VM is an emulation of a computer system. They provide settings to set the OS (Operating System) and custom hardware. This is to give the feeling that the user is experiencing a whole new system.

VM creates a sandbox environment that means that the application the user runs can use the allocated hardware and even if the application corrupts the VM OS (Operating System). The main operating system is not affected. A computer can deploy multiple VMs. The software that manages multiple deployments of VM is called a hypervisor. [6]

2.1.6 Container

Containers is a solution on how to get software to run from one computer environment to another environment. The docker file consists of the software and dependencies to run on different environments. [7]

The concept of container was introduced by Docker. Another way to consider containers is to think of it as an OS-level virtualization. The software that hosts the container is known as docker engine. [8]

2.1.7 Rendering

“Rendering or image synthesis is the process of generating a photorealistic or non-photorealistic image from a 2D or 3D model by means of a computer program.”[9] Rendering is mostly used on Video games , Architecture models , CGI (Computer-generated Imagery) and specific types of visualization. [9]

Real time Render

Mostly found in video games. The 3D images are calculated at high speeds. Ex: player playing a video game in real time. The objective is to create relative realism at a minimum or usually at 24 frames per second. The minimum to create the illusion for the human eye. [10]

Pre Render

This is used mostly where speed is not a concern but quality is. Pre-renders mostly use multiple CPU cores rather than the GPU. Examples of Pre render are architectural models and CGI (Computer-generated Imagery). [39]

2.1.8 Compute Unified Data Architecture (CUDA)

CUDA is a parallel computing platform or API. It was developed by NVIDIA to speed up applications processes by harnessing the power of the GPU.

CUDA is mostly only found on NVIDIA products such as GeForce , Quadro , ION and Tesla. CUDA has dramatically improved over the coming years. With version 9.2 by using multiple P100s GPUs you can get upto 50x performance over CPUs. [11]

2.1.9 remote Compute Unified Data Architecture (rCUDA)

rCUDA is a GPU virtualization solution. It is a middleware developed by the Universitat Politecnica de Valencia. rCUDA allows any computer to connect to a remote GPU. It gives the impression that the GPU is locally available to the OS (Operating System). GPU virtualization makes much better use of the Hardware in the computer. This makes the nodes with the GPUs become the servers and other nodes connected become clients. rCUDA enables concurrent use of CUDA enabled devices. This means with no modifications to the CUDA code the user can run it on rCUDA. [12]

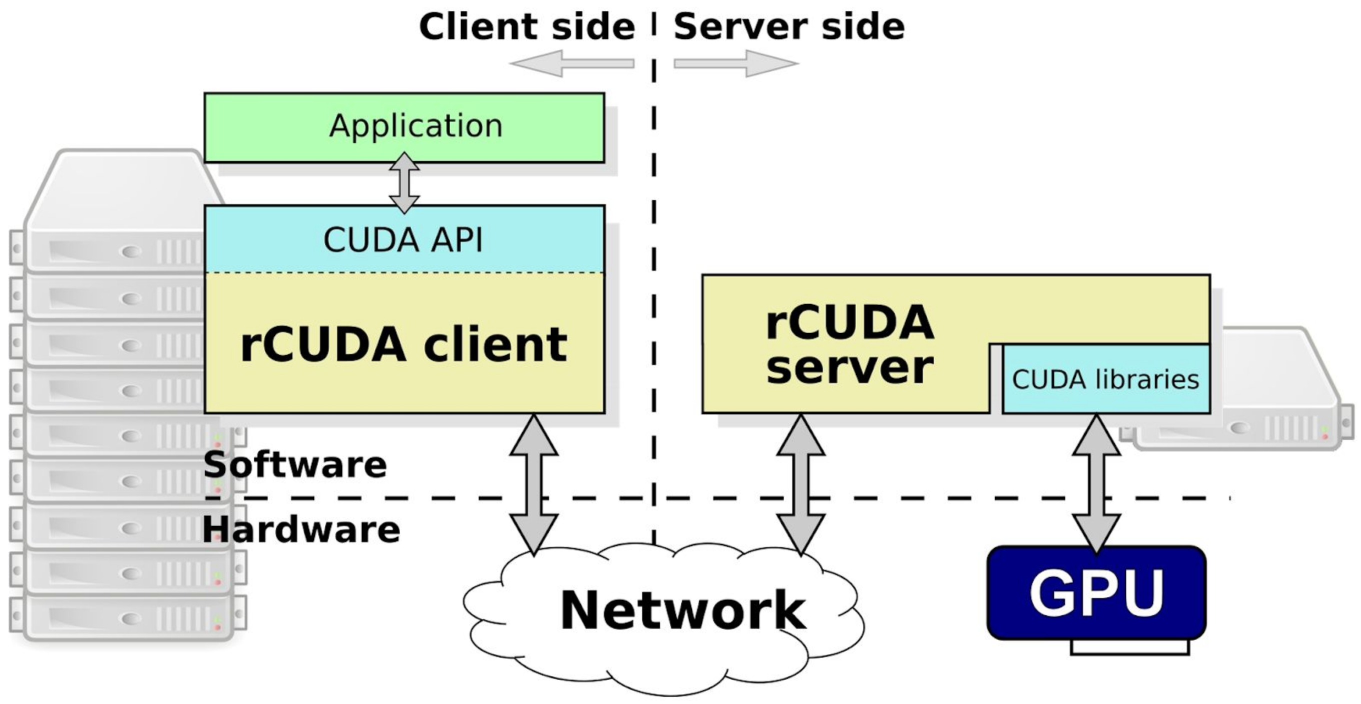


Fig 2.1 rCUDA architecture [13]

2.1.10 Open Graphics Library (OpenGL)

OpenGL is a cross platform and cross language API for rendering 2D and 3D graphics. This API is mostly used to interact with the GPU to achieve hardware accelerated rendering. OpenGL has bindings in many languages for example JavaScript binding WebGL and C binding WGL, GLX. [14]

2.1.11 Equalizer

Equalizer is a middleware for creating and deploying parallel and scalable OpenGL applications. Equalizer makes it possible to run OpenGL applications in any amount of GPU and CPU. This means the OpenGL application can either run on your laptop or Server Farms and code would be the same. [15]

2.1.12 Display Server

A display server is a program whose main task is to coordinate the output and input of clients from the OS (Operating System) , hardware. The display server talks to the client using a display server protocol.

The display server is a key component for the GUI (Graphical User Interface) and desktop displays. A few examples of display servers would be X11 , wayland , MIR. [16]

X window System (X11)

X11 provides a basic framework for drawing and moving different windows in a desktop environment. It can be found in most Unix like Systems. The user can consider it to act like a independant system for remote GUI (Graphical User Interface). [18]

Use cases for X11 [17]:

- Accessing remote machines graphically (Similar to remote desktop)
- Running heavy cases on remote machines and displaying result in local machine
- Running graphical software on many machines at once

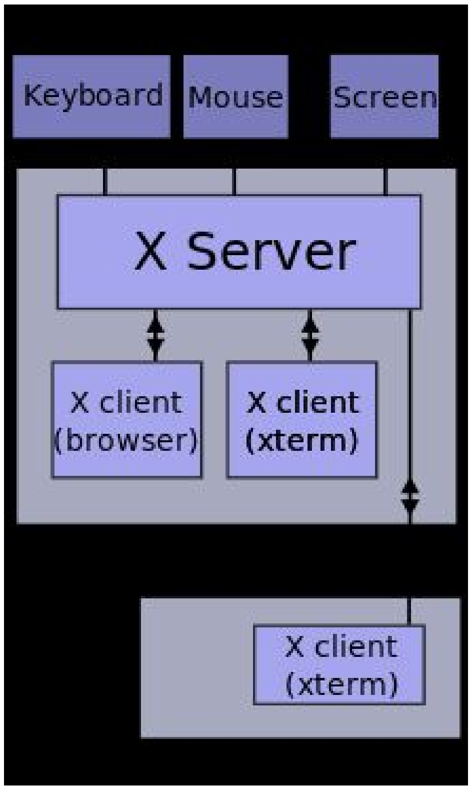


Fig 2.2 X Server Architecture [18]

2.1.13 Secure Shell (SSH)

SSH has been popular since its inception in 1995. It is a Cryptographic network protocol for running a secure shell over an unsecured network. We can login to a SSH server using either a password or a public key cryptography[19] [21].

A general use case for SSH would be for server administrators to login into remote servers and run specific tasks like deploy new applications or maintain them.

The SSH session begins by the client establishing connection with the server. Both the client and server send information using the SSH identification string (i.e protocol version , server version).

The client and server also exchange cryptographic algorithms and perform exchanging of keys for authentication. [20]

2.1.14 VirtualGL

VirtualGL is an open source toolkit used for remote and the possibility to run OpenGL on fully accelerated hardware. With VirtualGL OpenGL commands and 3D commands are redirected to GPU. Hence this helps to virtualize the GPU. The main point is that only rendered frames are only sent over the network. Like rCUDA VirtualGL allows multiple users to use the same GPU. If the network speed is over 100 megabits it gives you a workstation-like feel. Users can visualize huge amounts of data in real time without copying any data to the other machine. [22]

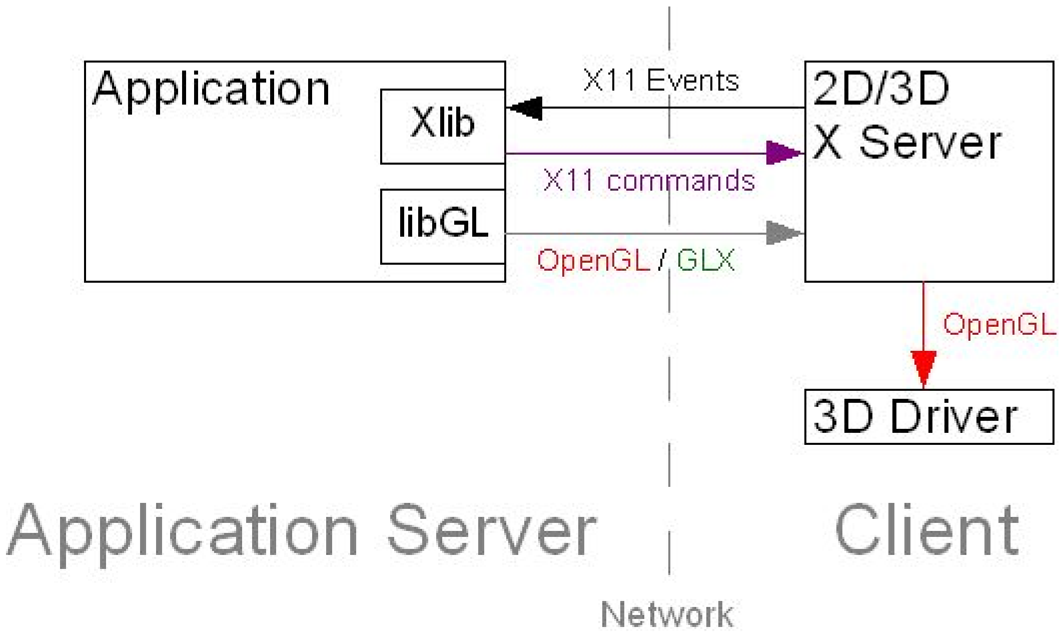


Fig 2.3 Indirect OpenGL Rendering using GLX [23]

2.2 Related Work

Doing computation in peer to peer networks or distributed is nothing new. There have been a lot of recent developments in this area.

2.2.1 Folding@Home

Folding@home is a distributing computing project used to help scientists simulate protein folding [24]. Folding@ Home uses a client and server architecture. The clients are installed by volunteers.

The clients request the server for tasks to process. Once a task is assigned the client downloads the necessary inputs files and sends the result back to the server. The folding@home architecture is really flexible and can be used to support other types of projects. To execute files, the folding@home main server sends signed binaries to execute on the client computer. The clients/volunteers have the capability to allocate how much computing power to use for the folding (i.e change to smaller work units due to bandwidth costs, set the number of cores in the CPU to ensure the laptop does not freeze).

The main role of the works server is to generate tasks for the volunteers/clients. Each server can run different projects administered by different researchers. The servers also act like schedulers and load balancers [25]. The servers monitor the task running in each of the clients.

To ensure that the assigned tasks can even run when the network is down. In recent versions folding@home have introduced something called collection servers. The main role of collection servers is to ensure that if the main server goes down the results can be uploaded to another set of servers. So that once the main servers are up the results are uploaded there from the collection servers. [27]

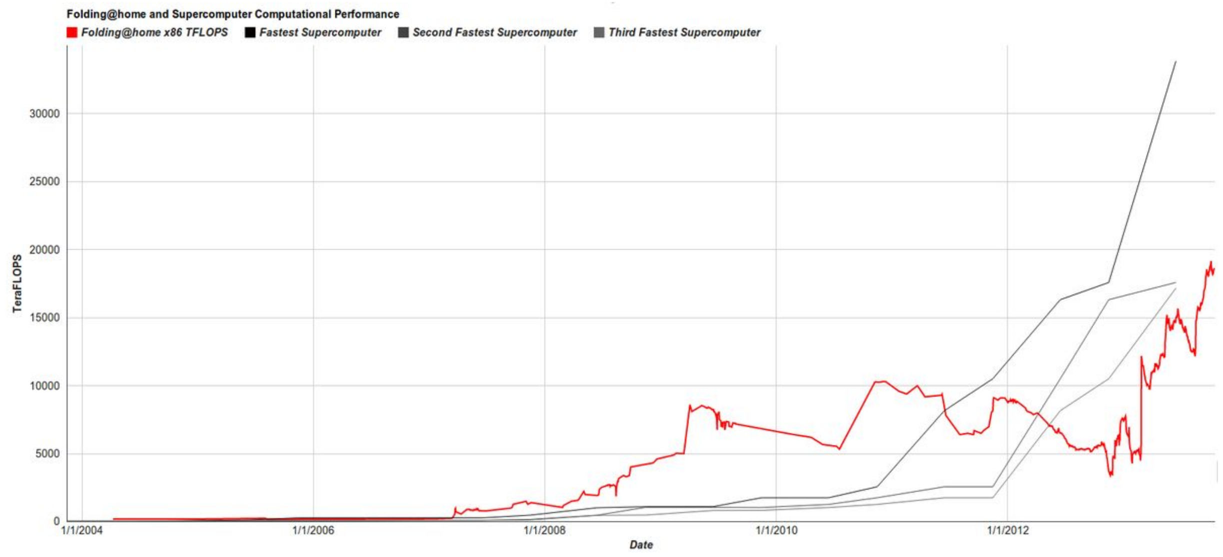


Fig 2.4 Folding@home vs Supercomputers [26]

2.2.2 Golem network

The golem network claims to be the first truly decentralized supercomputer. The main goal of golem is to create a global market of computing power. Golem connects nodes/computers in a decentralized network (i.e peer to peer network). The user can both request and lend resources. Golem has similar objectives to this project that is to provide an alternative to cloud rendering providers like AWS and Google Cloud. Golem network uses libp2p and IPFS to run it's peer to peer network. Golem uses Ethereum smart contracts [28] to run as it's financial incentive layer. [29]

Golem Timeline[29]:

1. Brass Golem: Proof of concept and focuses on Blender and LuxRender. Introduces IPFS to deliver content. Creates docker containers with golem provided images.
2. Clay Golem: Introduces Task API and application registry to make the network multi-purpose (currently in beta [30]). Supports virtual machines and docker containers.
3. Stone Golem: Introduces security and stability. Creates an advanced version of the Task API. Introduces certificates for community trust. Allows users to define tasks.

4. Iron Golem: Introduces whitelisting applications that can run outside the VM or containerised applications. Web interface to native Golem application. Golem is also planning to create a custom reputation system to monitor nodes in the network.

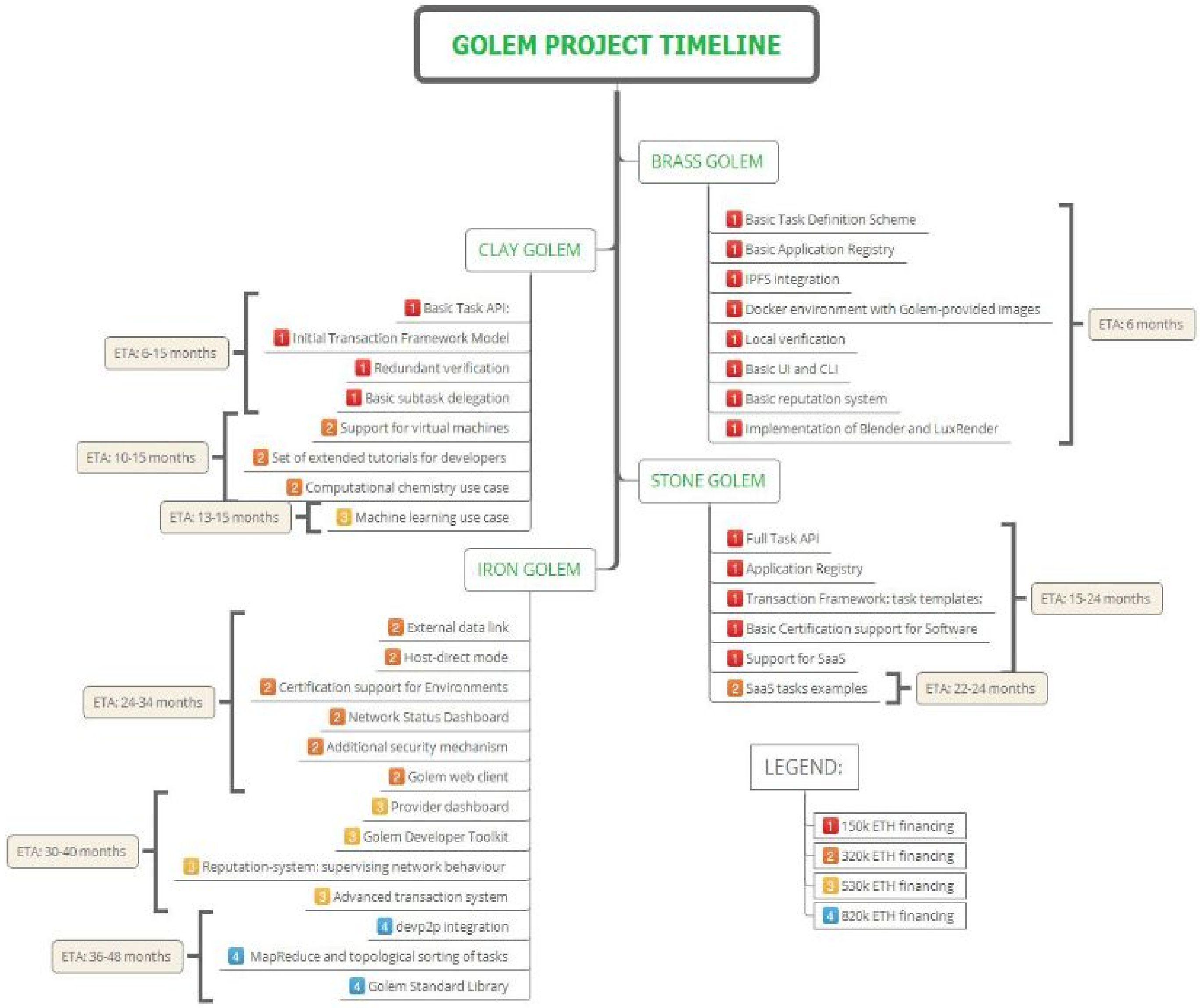

GOLEM PROJECT TIMELINE
BRASS GOLEM
1 Basic Task Definition Scheme
1 Basic Application Registry
1 IPFS integration
1 Docker environment with Golem-provided images
1 Local verification
1 Basic UI and CLI
1 Basic reputation system
1 Implementation of Blender and LuxRender
ETA: 6 months
CLAY GOLEM
1 Basic Task API:
1 Initial Transaction Framework Model
1 Redundant verification
1 Basic subtask delegation
ETA: 6-15 months
2 Support for virtual machines
2 Set of extended tutorials for developers
2 Computational chemistry use case
ETA: 10-15 months
3 Machine learning use case
ETA: 13-15 months
STONE GOLEM
1 Full Task API
1 Application Registry
1 Transaction Framework: task templates:
1 Basic Certification support for Software
1 Support for SaaS
ETA: 15-24 months
2 SaaS tasks examples
ETA: 22-24 months
IRON GOLEM
2 External data link
2 Host-direct mode
2 Certification support for Environments
2 Network Status Dashboard
2 Additional security mechanism
2 Golem web client
ETA: 24-34 months
3 Provider dashboard
3 Golem Developer Toolkit
3 Reputation-system: supervising network behaviour
3 Advanced transaction system
ETA: 30-40 months
4 devp2p integration
4 MapReduce and topological sorting of tasks
4 Golem Standard Library
ETA: 36-48 months
LEGEND:
1 150k ETH financing
2 320k ETH financing
3 530k ETH financing
4 820k ETH financing

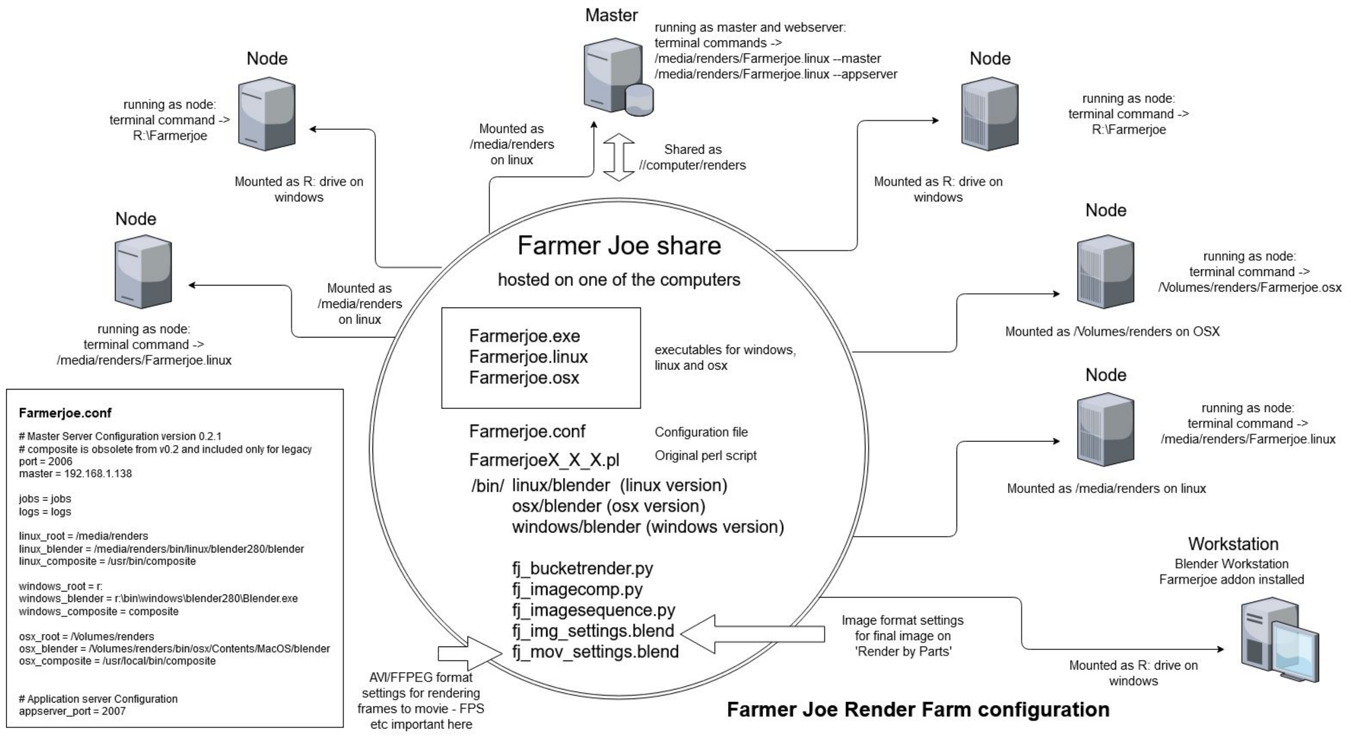
Fig 2.5 Timeline for Golem [29]

2.2.3 BitWrK

BitWrK main focus is to provide peer to peer rendering for Blender. BitWrK can be runned in any local network with other nodes to reduce bandwidth cost. BitWrK can be combined with nodes in the local network and with nodes outside the local network thus creating a huge swarm. BitWrK uses bitcoin as the mode of payment. This means users pay in bitcoin for using computing power. The whole implementation of the application is written in the Go programming language [32]. The client enables control of trade currently in progress. [33]

2.2.3 Farmer-Joe-Render

Farmer-Joe-Render is a render farm for Blender. Farmer-Joe-Render was a popular render farm used in Blender till version 2.5x. Farmer-Joe-render lost compatibility with higher versions of Blender because Blender changed their APIs. Blender version 2.8x got compatibility for Farmer-Joe-Render. Farmer-Joe-Render can be used to queue jobs on a single computer or multiple computers. Farmer-Joe-Render has compatibility on multiple Operating Systems. This means that the task batched to different nodes/computers can consist of different Operating Systems. Farmer-Joe-Render successfully works in Linux, Windows and macOS. Any computer that is able to run perl and the appropriate versions of Blender should be able to run Farmer-Joe-Render. [34]


Master
running as master and webserver:
terminal commands ->
/media/renders/Farmerjoe.linux --master
/media/renders/Farmerjoe.linux --appserver
Node
running as node:
terminal command ->
R:\Farmerjoe
Mounted as /media/renders on linux
Shared as //computer/renders
Mounted as R: drive on windows
Node
running as node:
terminal command ->
R:\Farmerjoe
Mounted as R: drive on windows
Node
running as node:
terminal command ->
/media/renders/Farmerjoe.linux
Mounted as /media/renders on linux
Node
running as node:
terminal command ->
/Volumes/renders/Farmerjoe.osx
Mounted as /Volumes/renders on OSX
Node
running as node:
terminal command ->
/media/renders/Farmerjoe.linux
Mounted as /media/renders on linux
Farmer Joe share
hosted on one of the computers
Farmerjoe.exe
Farmerjoe.linux
Farmerjoe.osx
executables for windows, linux and osx
Farmerjoe.conf
Configuration file
FarmerjoeX_X_X.pl
Original perl script
/bin/ linux/blender (linux version)
osx/blender (osx version)
windows/blender (windows version)
fj_bucketrender.py
fj_imagecomp.py
fj_imagesequence.py
fj_img_settings.blend
fj_mov_settings.blend
Image format settings for final image on 'Render by Parts'
AVI/FFPEG format settings for rendering frames to movie - FPS etc important here
Workstation
Blender Workstation
Farmerjoe addon installed
Mounted as R: drive on windows
Farmer Joe Render Farm configuration

Farmerjoe.conf

```
# Master Server Configuration version 0.2.1
# composite is obsolete from v0.2 and included only for legacy
port = 2006
master = 192.168.1.138

jobs = jobs
logs = logs

linux_root = /media/renders
linux_blender = /media/renders/bin/linux/blender280/blender
linux_composite = /usr/bin/composite

windows_root = r:
windows_blender = r:\bin\windows\blender280\Blender.exe
windows_composite = composite

osx_root = /Volumes/renders
osx_blender = /Volumes/renders/bin/osx/Contents/MacOS/blender
osx_composite = /usr/local/bin/composite


# Application server Configuration
appserver_port = 2007
```

Fig 2.6 Example setup of Farmer-Joe-Render [34]

2.2.5 DrQueue

DrQueue is an open source render farm system. Similar to Farmer-Joe-Render. DrQueue is used in the Visual effects ,science and finance industry. The objective of DrQueue is batch tasks and run as a task management tool for multiple nodes. DrQueue is compatible on Windows, macOS, Linux, Irix and FreeBSD. DrQueue can be used with any renderer that supports a CLI(Command Line Interface). DrQueue auto generated scripts for Blender, Maya, Lightwave, Cinema 4D, Maya, lightwave, Mental Ray, After Effects, Aqsis, 3Delight, Pixie, Snake , Terragen and Nuke. DrQueue makes it possible to talk to master and slave nodes with a custom API which any user can create based on the inbuilt functions. [35]

2.3 Critical analysis

Based on the literature review and related work done for peer to peer rendering, computation and task based render farm algorithms all have the same purpose, that is to make it easier to distribute tasks to other nodes.

Folding@home: Does the task of doing computation decentrally. The tasks are dispatched from a centralized server and only those who have permission can assign and batch tasks.

Golem network: Does the task of doing computation decentrally. The tasks can be dispatched from any node acting as the client. Tested and works well with blender. Testnet working with the use of Etherium as mode of transaction. Well maintained open source repository. Future plans involve allowing to run any containerized application on the network. Golem network is a peer to peer network.

BitWrK: Does the task of doing computation decentrally. The tasks can be dispatched from any node acting as the client. Works with blender only. Open Source application and stable release is available. Uses bitcoin as a mode of transaction. BitWrK is a peer to peer network.

Farmer-Joe-Render: Does the task of doing computation decentrally. The tasks can be dispatched from any node acting as the client. Made specifically for Blender. Open Source application and stable release is available. Not a peer to peer network. Use case is master batching tasks to slaves nodes in a local network or server farm.

DrQueue: Does the task of doing computation decentrally. The tasks can be dispatched from any node acting as the client. Can render anything that can be triggered from a CLI (Command line interface). Not a peer to peer network. Use case is master batching tasks to slaves nodes in a local network or server farm.

Based on the analysis of the following networks the closest network similar to this project is the Golem network, which lacks creating virtual environments for the specific task (i.e all the virtual environments have to be pre-set) and the tasks cannot be cutomly distributed to selected nodes by the client. The plan of this project is to implement the features that are not implemented in the Golem network and to develop a simpler algorithm to run custom tasks on the peer to peer network. DrQueue might be a good tool to be the task manager for this project.

Chapter 3: Requirement Analysis

To develop this network basic requirements need to be established. The following chapter will look at the functional and nonfunctional requirements as well as using the MoSCoW analyses to determine the importance of each requirement.

Requirements are classified into functional and nonfunctional:

Functional: Refers to required inputs and outputs of the application and network.

Nonfunctional: Refers to the behavior of the application and network.

The requirements are followed by the MoSCoW analysis

Must: These are the most important requirements of the application and network. Failure of these requirements would result in the failure of the application and network.

Should: Not as important as the above point. Failure to implement these requirements would have a relatively high impact on the application and network. Does not always result in the failure in the application and network.

Could: Not mandatory requirements. Implementation of these requirements could enhance the use case of the application and network.

Would: Not required requirements. Implementation to be only considered if sufficient time is available.

3.1 Functional Requirements

FR No.	Description of requirement	Priority
1	Must be able to read computing power available from neighbouring computers.	Must
2	Client must be able to SSH into selected servers in the p2p network	Must
3	Servers nodes must be able to create virtual environments	Should
4	Application should come with client and server mode	Must
5	Application should allow the server node to set how much computation power the server is willing to provide.	Must
6	Application informs client when server is doing client operations and server goes down.	Must
7	Client must be able to interact with the network using the Web applications and CLI(command line interface)	Should

Table 3.1 Functional Requirements

3.2 Non - Functional Requirements

N-FR No.	Description of requirement	Priority
1	Client nodes must be able to automatically batch the process to another server node when the current server node is down.	Could
2	Server nodes should be able to create a virtual environment in under 2 minutes.	Could
3	Pulling all dependencies to run tasks from clients nodes	Should

Table 3.2 Non - Functional Requirements

Chapter 4: Evaluation

This section will look at how each section of the project will be evaluated. The reason of choice to use different evaluations strategies is due to the reason that each component in the project plays a different role. Each of the following components in the project are linked to each other.

4.1 Installation

The application should be able to install by single script no matter the OS (Operating System) or computer specification. After installations is when the configurations will be set. The installation should come bundled with the client and server. To ensure all modules are installed properly a set of unit tests are to be runned to ensure each module is working as intended.

4.2 Client

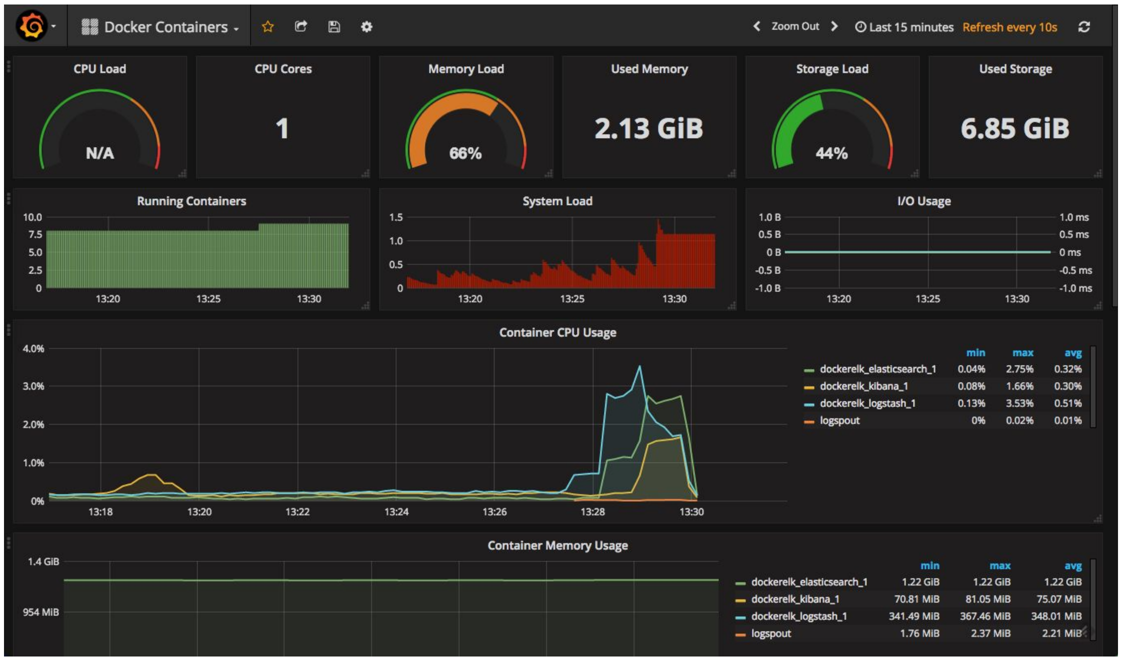
The Client side will evaluate if the user is able to view the servers which broadcast the computing power available for a task. For testing, a set number of clients and servers in the testnet (A test network) will be created. The time will be logged of how long it takes the client to detect the server, once the server node joins the network. The client would do a speed test to the server and check metrics like ping, download rates and upload rates. This gives a good idea to the client if it's a good idea to do tasks such as live renders or can the network take full advantage of the speeds the ISP (Internet Service Provider) provides.

4.3 Peer to Peer network

The Peer to peer network will be benchmarked using a framework called Testground[36]. Multiple docker images will be created which contain this project. These docker images will be randomly set to nodes in different locations and the multiple docker containers can also be present in the same computer based on the test configuration. Each of these docker containers can either act as Clients or servers. Once the docker containers are set up, test operations are triggered; these operations don't take much time to run. The main objective is to check the quality (i.e average time to pull tasks, upload results from the server side and average time to discover nodes in the network) of the testnet (test network). The end the test reflects on how the peer to peer networks run as a whole.

4.4 Server Side

On the server side the measure would be how long it takes to create a virtual environment. Specific tests on the GPU performance using different virtualization environments. To check if the server side is working in a production environment the client assigned SSH into the server virtual environment. If the SSH is successful the client can start batching tasks to the server. The server will also broadcast metrics such as GPU. CPU and memory usage to the client and server owners using Prometheus [37].



	min	max	avg
dockerelk_elasticsearch_1	0.04%	2.75%	0.32%
dockerelk_kibana_1	0.08%	1.66%	0.30%
dockerelk_logstash_1	0.13%	3.53%	0.51%
logspout	0%	0.02%	0.01%

	min	max	avg
dockerelk_elasticsearch_1	1.22 GiB	1.22 GiB	1.22 GiB
dockerelk_kibana_1	70.81 MiB	81.05 MiB	75.07 MiB
dockerelk_logstash_1	341.49 MiB	367.46 MiB	348.01 MiB
logspout	1.76 MiB	2.37 MiB	2.21 MiB

Fig 4.1 Prometheus dashboard using grafana [38]

4.5 Rendering

The main comparison will be running renders in a single computer vs running in multiple computers. The metric would be to measure the recommended bandwidth required for different types of rendering tasks if they are live renders. Evaluate if this project can reduce the amount of power used for rendering by sending the tasks to a server which has CPU or GPU equally or more powerful but consumes less power.

Chapter 5: Project management

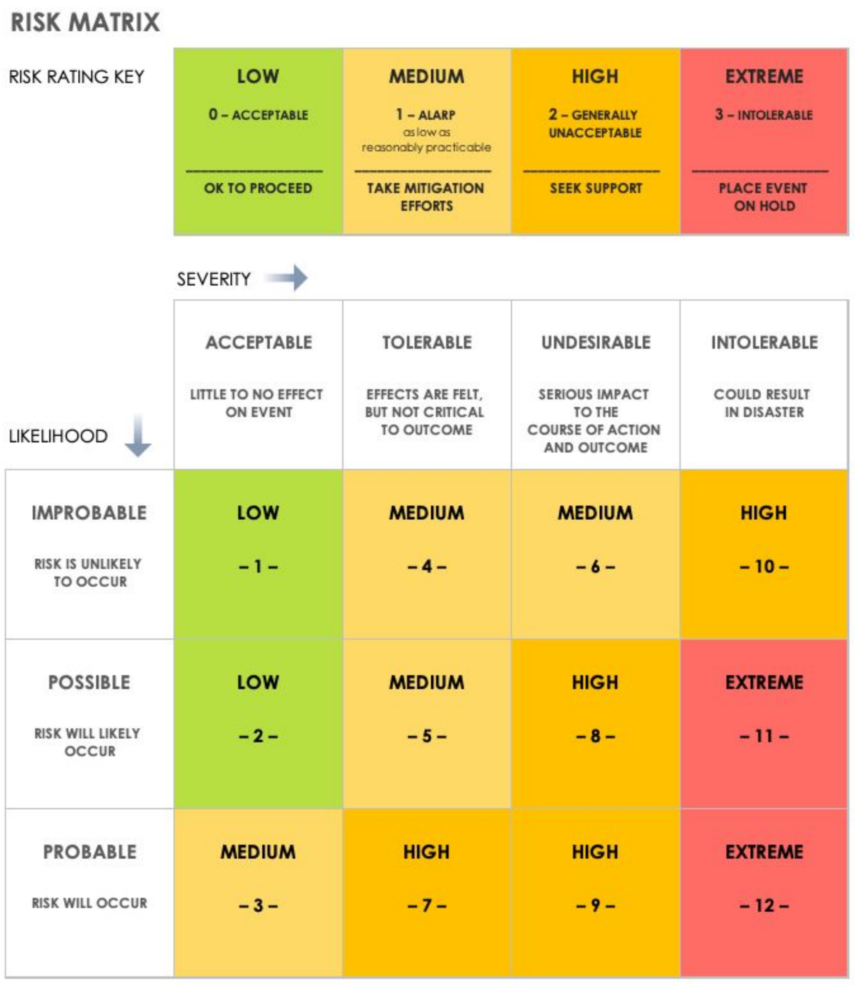
This chapter focuses on the strategies used to manage the project while it's being developed. This includes the possible risks that can occur during the development process and testing process. This chapter includes a risk assessment matrix to determine the possible risks and types of risks that can occur. Based on the risk assessment steps can be taken to mitigate or minimize these risks.

The second part of this chapter focuses on the timetable and how it can determine the tasks of the project that can be achieved. This part will be done using a Gantt chart. The Gantt chart gives a visual representation of how long each task takes. Using the Gantt chart, it will be easier to determine if the project is on course and to predict the steps coming up next.

5.1 Risk Management

Risks are bound to happen no matter the size of the project. It is very important to determine the risk at early stages so that the necessary steps can be taken to minimize or mitigate them.

This risk management matrix will be a 3X4 matrix. The 3X4 matrix uses a non numerical scale for likelihood and severity. [40]



RISK MATRIX

RISK RATING KEY	LOW	MEDIUM	HIGH	EXTREME
	0 – ACCEPTABLE	1 – ALARP as low as reasonably practicable	2 – GENERALLY UNACCEPTABLE	3 – INTOLERABLE
	OK TO PROCEED	TAKE MITIGATION EFFORTS	SEEK SUPPORT	PLACE EVENT ON HOLD

LIKELIHOOD / SEVERITY	ACCEPTABLE LITTLE TO NO EFFECT ON EVENT	TOLERABLE EFFECTS ARE FELT, BUT NOT CRITICAL TO OUTCOME	UNDESIRABLE SERIOUS IMPACT TO THE COURSE OF ACTION AND OUTCOME	INTOLERABLE COULD RESULT IN DISASTER
IMPROBABLE RISK IS UNLIKELY TO OCCUR	LOW – 1 –	MEDIUM – 4 –	MEDIUM – 6 –	HIGH – 10 –
POSSIBLE RISK WILL LIKELY OCCUR	LOW – 2 –	MEDIUM – 5 –	HIGH – 8 –	EXTREME – 11 –
PROBABLE RISK WILL OCCUR	MEDIUM – 3 –	HIGH – 7 –	HIGH – 9 –	EXTREME – 12 –

Fig 5.1 3X4 matrix for risk analyses [40]

Potential risk	Likelihood	Severity	Level of risk before precaution	Precautions	Level of risk after precaution
Server-side : Task corrupting OS (Operating System)	POSSIBLE	INTOLERABLE	EXTREME	To execute tasks in a sand-box environment or virtual environment	MEDIUM
Server-side: Freezing server-side	PROBABLE	UNDESIRABLE	EXTREME	To set the sandbox or virtual	MEDIUM

				environment on how much computing power can be used in the operation	
Peer to peer network: Server suddenly goes down in middle of the operation	POSSIBLE	TOLERABLE	MEDIUM	Inform the client that the server of that task is down and the client chooses to execute that task on another server in the network.	LOW
Peer to peer network: Client side goes down and tasks batched from client to the server	POSSIBLE	TOLERABLE	MEDIUM	Inform the server that the client is down and cancel the task running and kill the virtual or sandbox environment	LOW
Covid19 restrictions	PROBABLE	TOLERABLE	HIGH	To use microsoft teams to discuss the status of the project.	MEDIUM
Inadequate computation devices available to test the network.	POSSIBLE	UNDESIRABLE	MEDIUM	Consult supervisor for more access to more computers or test smaller tasks on local virtual machines	LOW

Supervisor illness	IMPROBABL E	TOLERABLE	MEDIUM	Consult MACS department for a solution	LOW
Author illness	IMPROBABL E	UNDESIRABLE	MEDIUM	Work from home and fill in MC form if extension needed	LOW
Loss of source code	PROBABLE	INTOLERABLE	EXTREME	Backup code in multiple remote GIT repositories.	MEDIUM

Table 5.1 Risk management table

5.2 Timetable

This section is to create a timeline for this project. Each module of the project will not take the same amount of time to implement due factors such as the amount of support a library provides. For example It is very easy to set up a custom peer to peer network with libp2p which takes care of helping the developer set the exchange protocol, routing algorithm and transport protocols but it is relatively challenging to design applications to run custom tasks on the network as that requires a lot of middleware code and specific setups on the clients and servers.

The timetable will be represented using a Gantt chart. In the development phase from January 2021 onwards the Gantt chart will be generated from issues created from the github repository. The github repository will stay private till the 4th year project is completed. The project will be made public with the consent of the project supervisor.

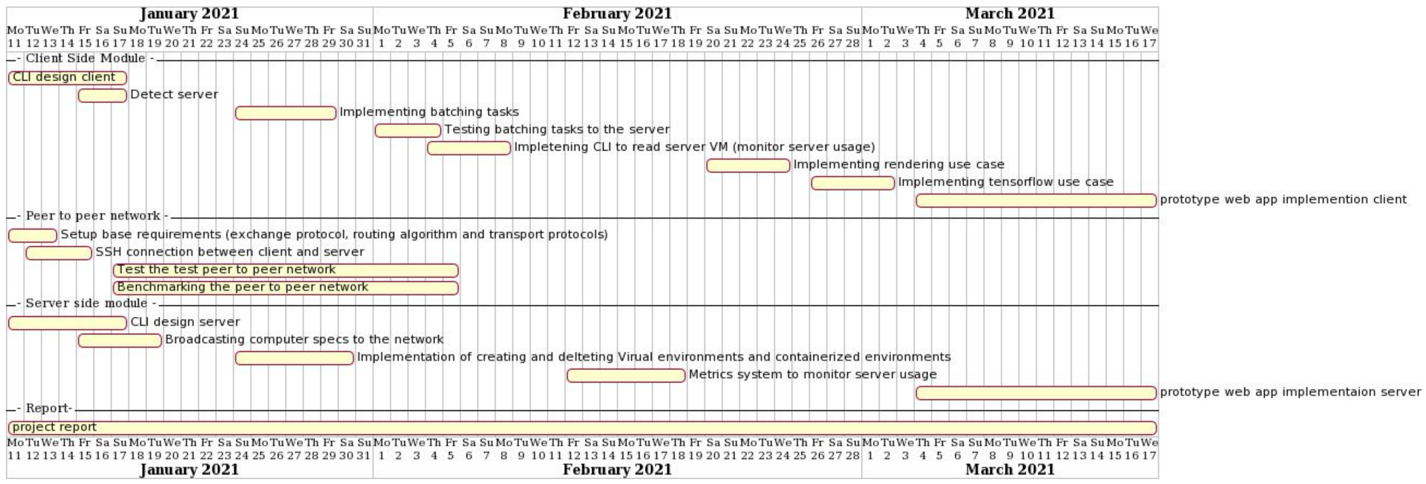


Fig 5.1 Gantt chart for this project [41]

The Gantt chart is subject to change based on unforeseen challenges in the project and semester 2 teaching and coursework schedule.

5.3 Professional, Ethical, Legal and Social issues

This project aims on dispatching custom tasks with the test case of Rendering and other computation on the peer to peer network. An important consideration is to ensure that the client running tasks on cannot tap into personal information of the server administrator. The tasks the user runs can be from a proprietary software.

5.3.1 Professional

This project follows the professional guidelines of the BCS code of conduct [42]. This project is developed at the best interest of the users and public interest. The project aims by allowing different clients to batch tasks to different servers. The clients and server administrators are aware about what is being modified in their system.

5.3.2 Ethical and Social Issues

When running tasks in the peer to peer system if the client or the server administrator has any doubt about the tasks running, Both parties can cancel the run of the task. All the tasks executed in the server node are executed in a virtual and containerized environment so that the client executing tasks cannot access the server administrators personal information and computation power not assigned to the client.

5.3.3 Legal

Proprietary software can be runned on the network as long as both the client and server have the license to run them. The source code of this project will be released using the GPLv2 license [43]. This is to ensure the developers are not responsible for the client running any pirated software on top of the network. There will be no royalties when this project is released. The source code of this project can be freely distributed and should be without any warranties.

Chapter 6: Conclusion

This project focuses on decentrally batching tasks (test cases are on rendering and machine learning libraries like tensorflow) for computation to nodes decentrally instead of using central servers to increase bandwidth. The related works section (2.2.2) will help combine multiple other open source projects to reduce the time to build this project. This project does not focus on the economic incentive part as it is out of the scope of this project.

References

[1]"Peer-to-peer", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Peer-to-peer. [Accessed: 15- Nov- 2020].

[2]"Client–server model," *Wikipedia*. Oct. 20, 2020, Accessed: Nov. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=984503284.

[3]"IPFS - Content Addressed, Versioned, P2P File System", *ipfs.io*, 2020. [Online]. Available: https://raw.githubusercontent.com/ipfs-inactive/papers/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf. [Accessed: 26- Nov- 2020].

[4]"What is a CDN? | How do CDNs work? | Cloudflare UK," *Cloudflare*. https://www.cloudflare.com/en-gb/learning/cdn/what-is-a-cdn/ (accessed Nov. 26, 2020).

[5] D. M. (IPFS) and E. Lee (Netflix), "New improvements to IPFS Bitswap for faster container image distribution," *IPFS Blog*. https://blog.ipfs.io/2020-02-14-improved-bitswap-for-container-distribution/ (accessed Nov. 26, 2020).

[6]"Virtual machine," *Wikipedia*. Oct. 28, 2020, Accessed: Nov. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=985899745.

[7]"What is a Container? | App Containerization | Docker." https://www.docker.com/resources/what-container (accessed Nov. 26, 2020).

[8]"Docker (software)," *Wikipedia*. Nov. 24, 2020, Accessed: Nov. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=990463656.

[9]"Rendering (computer graphics)," *Wikipedia*. Nov. 16, 2020, Accessed: Nov. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rendering_(computer_graphics)&oldid=988943275.

[10] “3D real-time rendering - how does it work?,” *Unity*. https://unity3d.com/real-time-rendering-3d (accessed Dec. 07, 2020).

[11] M. Heller, “What is CUDA? Parallel programming for GPUs,” *InfoWorld*, Aug. 30, 2018. https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html (accessed Nov. 26, 2020).

[12] C. Reaño, A. J. Peña, F. Silla, J. Duato, R. Mayo and E. S. Quintana-Ortí, "CU2rCU: Towards the complete rCUDA remote GPU virtualization and sharing solution," *2012 19th International Conference on High Performance Computing*, Pune, 2012, pp. 1-10, doi: 10.1109/HiPC.2012.6507485.

[13]“Strategy Behind Virtualizing GPUs,” *HPCwire*, Aug. 20, 2018. https://www.hpcwire.com/2018/08/20/strategy-behind-virtualizing-gpus/ (accessed Nov. 26, 2020).

[14]“OpenGL,” *Wikipedia*. Oct. 20, 2020, Accessed: Nov. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenGL&oldid=984492616.

[15]*Eyescale/Equalizer*. Eyescale Software GmbH, 2020.

[16]“Display server,” *Wikipedia*. Jul. 02, 2020, Accessed: Oct. 05, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Display_server&oldid=965564877.

[17] M. A. Dolan and L. Hare, "X window system servers in embedded systems," Digest of Papers Compcon Spring '90. Thirty-Fifth IEEE Computer Society International Conference on Intellectual Leverage, San Francisco, CA, USA, 1990, pp. 314-319, doi: 10.1109/CMPCON.1990.63693.

[18]“X Window System,” *Wikipedia*. Nov. 25, 2020, Accessed: Nov. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=X_Window_System&oldid=990674231.

[19] N. Saini, N. Pandey and A. P. Singh, "Enhancement of security using cryptographic techniques," 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Noida, 2015, pp. 1-5, doi: 10.1109/ICRITO.2015.7359224.

[20] O. Gasser, R. Holz and G. Carle, "A deeper understanding of SSH: Results from Internet-wide scans," 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1-9, doi: 10.1109/NOMS.2014.6838249.

[21]“SSH (Secure Shell),” *Wikipedia*. Nov. 27, 2020, Accessed: Nov. 27, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SSH_(Secure_Shell)&oldid=990966773.

[22]“VirtualGL | About / A Brief Introduction to VirtualGL.” https://virtualgl.org/About/Introduction (accessed Nov. 27, 2020).

[23] “VirtualGL | About / VirtualGL Background.” https://virtualgl.org/About/Background (accessed Nov. 27, 2020).

[24] J. N. Onuchic and P. G. Wolynes, “Theory of protein folding,” *Current Opinion in Structural Biology*, vol. 14, no. 1, pp. 70–75, Feb. 2004, doi: 10.1016/j.sbi.2004.01.009.

[25] S. Moharana, “Analysis of Load Balancers in Cloud Computing,” *International Journal of Computer Science and Engineering*, vol. 2, pp. 101–108, May 2013.

[26] J. V, *English: This is a graph of the Folding@home distributed computing project’s computational performance between April 8, 2004 and October 15, 2013. The graph shows Folding@home’s performance in x86 teraFLOPS and the rMax speed of the top three supercomputers from top500.org. Folding@home’s performance is significantly above the most powerful supercomputer until April of 2011 when the K Computer overtook it*. 2012.

[27] A. Beberg, D. Ensign, G. Jayachandran, S. Khaliq, and V. Pande, *Folding@home: Lessons from eight years of volunteer distributed computing*. 2009, p. 8.

[28] W. Metcalfe, “Ethereum, Smart Contracts, DApps,” 2020, pp. 77–93.

[29] “Golem whitepaper” https://golem.network/crowdfunding/Golemwhitepaper.pdf(accessed Dec. 1, 2020).

[30] “Install Clay Beta for Windows - Golem.” https://golem.network/download/clay-beta/windows/ (accessed Dec. 01, 2020).

[31] “Golem-Releases-Brass-Golem-Beta-0.16.0.png (PNG Image, 875 × 376 pixels).” https://bitnewsbot.b-cdn.net/wp-content/uploads/2018/06/Golem-Releases-Brass-Golem-Beta-0.16.0.png (accessed Dec. 02, 2020).

[32] E. Westrup and F. Pettersson, “Using the Go Programming Language in Practice,” 2014.

[33] “BitWrk - Distributed Rendering for Blender.” https://bitwrk.net/ (accessed Dec. 02, 2020).

[34] Laurencitow, *Laurencitow/Farmer-Joe-Render-Farm*. 2020.

[35] *DrQueue/drqueue*. DrQueue, 2020.

[36] “What is Testground?” https://docs.testground.ai/ (accessed Dec. 06, 2020).

[37] “Overview | Prometheus.” https://prometheus.io/docs/introduction/overview/ (accessed Dec. 06, 2020).

[38] “Monitoring a Dockerized ELK Stack With Prometheus and Grafana - DZone Performance,” *dzone.com*. https://dzone.com/articles/monitoring-a-dockerized-elk-stack-with-prometheus (accessed Dec. 06, 2020).

[39]“Pre-rendering,” *Wikipedia*. Nov. 30, 2020, Accessed: Dec. 07, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pre-rendering&oldid=991588674.

[40] “Download Free Risk Matrix Templates | Smartsheet.” https://www.smartsheet.com/all-risk-assessment-matrix-templates-you-need (accessed Dec. 09, 2020).

[41] “PlantUML Web Server,” *PlantUML.com*. http://www.plantuml.com/plantuml (accessed Dec. 10, 2020).

[42] “BCS code of conduct” https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf (accessed Dec. 10, 2020)

[43] “GNU General Public License version 2 | Open Source Initiative.” https://opensource.org/licenses/gpl-2.0.php (accessed Dec. 10, 2020).

[44]C. Kilcioglu and J. Rao, *Competition on Price and Quality in Cloud Computing*. 2016, p. 1132.