Social Distributed Computing Service Framework

by

Daniel Latimer

Bachelor of Computer Science, UNB, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Weichang Du, PhD, Faculty of Computer Science
Examinining Board: Eric Aubanel, PhD, Faculty of Computer Science
Virendra Bhavsar, PhD, Faculty of Computer Science
Fan-Rui Meng, PhD, Faculty of Forestry and Env. Management

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2013

©Daniel Latimer, 2013
Abstract

Social Distributed Computing, as a new concept, intersects conventional distributed computing with the Online Social Networks which have become important parts of our lives. Using the pre-existing social links, we can increase the amount of software functionality we have available to ourselves by sharing our applications with friends and using the applications our friends have shared. The software can be run on the friend’s machine and input/output can be streamed through the Social Distributed Computing services allowing the requesting friend to access it through a light-weight client such as a web browser. The Social Distributed Computing framework also opens the door to allow friends to share computing resources with one another when they are not being used. This provides in essence, a social grid computing environment, making grid computing accessible to anyone instead of only large scientific endeavors.

The Social Distributed Computing framework consists of 3 main modules which each perform a service. The Control Module interacts with the Online
Social Network allowing the users to declare which resources and applications they would like to share, and shows which are available to the current user. The Application Sharing Module sets up and runs an application sharing instance allowing the remote execution of a program by a friend. This is accomplished by streaming the input/output through the service. The Resource Sharing Module allows users to take advantage of their friend’s shared resources to perform remote computation and storage. To accomplish this the service is used to forward code and data, making social grid computing possible. In this thesis I detail several use cases for possible applications of the framework. I also implemented the Control and Application Sharing modules and performed two performance experiments to create and evaluate a prototype for the concept of Social Distributed Computing.
Dedication

I would like to dedicate this work to my family who have always encouraged me to pursue my dreams.
Acknowledgements

I would like to express my appreciation to my supervisor, Dr. Weichang Du, for all his guidance and support, and for always making time for me when I needed advice.

I would also like to thank all of the instructors I’ve had at the University of New Brunswick throughout my Bachelors and Masters. Your passion for the subject matter is infectious and made the courses that much more enjoyable.

I would like to thank Dr. Ken Kent for his guidance throughout my undergrad and while I was looking to enroll in grad school.

Finally, thanks go out to my friend Laura Steeves, and my mother Gloria Latimer for helping me run one of my experiments from different locations.
# Table of Contents

Abstract ii  
Dedication iv  
Acknowledgments v  
Table of Contents x  
List of Figures xiii  
Abbreviations xiv  

1 Introduction 1  
1.1 Motivation .................................................. 1  
1.2 What is Social Distributed Computing .................. 3  
1.3 Thesis Overview ............................................ 6  
  1.3.1 Objective .............................................. 6  
  1.3.2 Problems & Challenges ............................... 7  
  1.3.3 Structure of Thesis ................................. 9
2 Background

2.1 Client-Server vs. Peer-to-Peer Computing

2.2 Ego-centric Social Network Analysis

2.3 Online Social Network Functionality

2.3.1 Asynchronous Messaging

2.3.2 Public Asynchronous Messaging

2.3.3 Synchronous Messaging

2.3.4 Friend Searching

2.3.5 Event Planning

2.3.6 Affiliation Networks

2.3.6.1 Groups

2.3.6.2 Likes

2.3.7 Media Sharing

2.4 Distributed Computing

2.5 Service Computing Models

2.5.1 Asynchronous vs. Synchronous

2.5.2 Interactive vs. Non-interactive

2.5.3 Single User vs. Collaborative

2.5.4 Single Point Execution vs. Distributed

3 Social Distributed Computing Concept

3.1 Why do we need Social Distributed Computing?

3.2 Functionality at its Highest Level
3.2.1 Application Sharing ........................................ 30
3.2.2 Resource Sharing ........................................ 31
3.3 Security Considerations .................................... 31
3.4 Social Network Integration ................................. 32
3.5 What is the Service Computing Model .................... 34
3.6 Social Distributed Computing as a Service ............... 35
3.7 How Distributed Computing is realized as Social Distributed Computing ......................... 36

4 Architecture and Models .................................. 38
4.1 Use Cases .................................................. 38
4.2 Architecture Design ...................................... 41
4.3 Social Computing Models ................................ 43
4.3.1 One-to-One Social Computing Model ................ 43
4.3.2 One-to-Many Social Computing Model .............. 44
4.3.3 Many-to-One Social Computing Model ............... 46
4.3.4 Many-to-Many Social Computing Model ............. 46

5 Implementation ............................................. 50
5.1 Application Requesting .................................. 52
5.2 Command Line Application Sharing ...................... 56
5.3 GUI Application Sharing ................................ 59

6 Case Studies and Experimentation ...................... 64

viii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Case Study: Command Line Application Sharing</td>
<td>65</td>
</tr>
<tr>
<td>6.2 Case Study: GUI Application Sharing</td>
<td>68</td>
</tr>
<tr>
<td>6.3 Case Study: Parallel Computing</td>
<td>71</td>
</tr>
<tr>
<td>6.3.1 Parallel Computing Background</td>
<td>71</td>
</tr>
<tr>
<td>6.3.2 Case Study</td>
<td>71</td>
</tr>
<tr>
<td>6.4 Case Study: Distributed Computing</td>
<td>73</td>
</tr>
<tr>
<td>6.4.1 Distributed Computing Background</td>
<td>73</td>
</tr>
<tr>
<td>6.4.2 Case Study</td>
<td>73</td>
</tr>
<tr>
<td>6.5 Case Study: Social Cloud</td>
<td>74</td>
</tr>
<tr>
<td>6.5.1 Cloud Computing Background</td>
<td>74</td>
</tr>
<tr>
<td>6.5.1.1 Software as a Service</td>
<td>75</td>
</tr>
<tr>
<td>6.5.1.2 Platform as a Service</td>
<td>75</td>
</tr>
<tr>
<td>6.5.1.3 Infrastructure as a Service</td>
<td>76</td>
</tr>
<tr>
<td>6.5.2 Case Study</td>
<td>76</td>
</tr>
<tr>
<td>6.5.3 Social Cloud over Distributed Social Computing Platform</td>
<td>77</td>
</tr>
<tr>
<td>6.6 Experiment 1: Application Sharing Network Latency</td>
<td>79</td>
</tr>
<tr>
<td>6.6.1 Purpose</td>
<td>79</td>
</tr>
<tr>
<td>6.6.2 Background</td>
<td>81</td>
</tr>
<tr>
<td>6.6.3 Hypothesis</td>
<td>83</td>
</tr>
<tr>
<td>6.6.4 Experiment</td>
<td>83</td>
</tr>
<tr>
<td>6.6.5 Analysis</td>
<td>84</td>
</tr>
<tr>
<td>6.6.6 Conclusion</td>
<td>87</td>
</tr>
</tbody>
</table>
6.7 Experiment 2: System Latency and Server Stress Testing . . . 88
   6.7.1 Purpose ........................................... 88
   6.7.2 Background ...................................... 89
   6.7.3 Hypothesis ...................................... 89
   6.7.4 Experiment ...................................... 90
   6.7.5 Analysis ......................................... 92
   6.7.6 Conclusion ...................................... 93

7 Conclusions ........................................... 95
   7.1 Conclusion ......................................... 95
   7.2 Contributions ..................................... 97
   7.3 Future Work ....................................... 98

Bibliography ........................................... 107

Vita
List of Figures

2.1 Client-Server Architecture - convenient yet requires horizontal scaling (creating more servers) as the number of clients increase 12

2.2 Peer-to-Peer Architecture allows for each computer to connect to one another, however this means each computer must have N-1 connections (where N is the number of computers) to have a fully connected network 13

2.3 A Network Address Translator (NAT) multiplexing multiple computers over one public IPv4 address. Here we see how packet headers are stripped and replaced when traveling through a NAT 16

2.4 The bolded lines and users show an ego-centric view of the social network focused on the subject in red. Note that links between friends of the subject are not considered 19

4.1 Use case possibilities for the Social Distributed Computing system broken down into the roles of requester and provider 39

4.2 Social Distributed Computing Modules 42

4.3 One-to-One Social Computing Model 45
6.5 Here we can see how Social Computing could be implemented using the Social Distributed Computing Platform.

6.6 The tasks that are required in performing a round trip of the application sharing software.

6.7 In this figure we see Total Latency to perform a round trip vs. Network Latency to perform the round trip. These are the two values measured by the experiment. Each trial was executed 100 times, averaged, and the standard deviation was calculated, which you can see in this figure denoted by the error bars.

6.8 This figure has the same scale as the previous, and shows how much System Delay fluctuated in our experiment by subtracting Network Delay from Total Delay.

6.9 Latency vs. number of users, we can see that there is a linear relationship between latency and the number of users accessing the system concurrently.
List of Abbreviations

COTS  Commercial Off The Shelf software
GIS   Geographic Information System
GUI   Graphical User Interface
GUID  Globally Unique Identifier
IETF  Internet Engineering Task Force
IM    Instant Messaging
IO    Input/Output
IPv4  Internet Protocol Version 4
ISP   Internet Service Provider
JNLP  Java Network Launching Protocol
JXTA  Juxtapose, a P2P framework
KISS  Keep It Simple Stupid
LAMP  Linux, Apache, MySQL, PHP
LAN   Local Area Network
NAT   Network Address Translator
OSN   Online Social Network
P2P   Peer-to-Peer
UIA   Unmanaged Internet Architecture
WAN   Wide Area Network
YAGNI You Are not Going to Need It
Chapter 1

Introduction

1.1 Motivation

Online Social Networks are online applications, usually built as part of the World Wide Web, that are designed to allow users to exchange information and keep in touch the way one normally would outside the digital domain. They have a decent range of social interactions that are supported at this time. Most of these interactions are based on real life social actions that would be performed with friends and family. These interactions might include having a conversation through text, audio, or video, playing a game together, organizing events, and communicating with groups who have similar interests. There have also been some attempts to integrate other functionality such as collaborative work on projects or even the same document.
Online Social Networking became a reality for most people around 2004 with the popularization of Facebook. There had been Online Social Networks before Facebook, however this was the application that saw an explosive growth in the market bringing in not only early adopters but everyday people. It has become so ubiquitous that in some cases it became the reason for people to become computer literate. Many elderly people before this point had not seen the need to learn to use a computer. Having the ability to use the web to get news and weather wasn’t a good enough reason when their older mediums such as newspapers and television still provided this information. Online Social Networking changed that by introducing new web functionality such as the possibility to interact, and keep up to date with family and friends who may be across the country. Since that time Online Social Networks have become an everyday part of many people’s lives. Used as a tool to connect with people and exchange information, these applications have allowed people to reconnect with long lost friends, stay in touch while being thousands of kilometers away, and plan local events where they are able to meet new people from their area. While these functionalities have revolutionized the way we interact with one another, the potential Online Social Networking presents us has not truly been explored outside of the domain of information exchange.

The potential for Online Social Networks to include functionality outside the domain of information exchange leads one to question what other possibilities could this framework for communicating between friends bring? Using
the social links created by an Online Social Network we could provide a platform that allows friends to perform other functionality that is fundamental to computers - computation. This functionality can be further described as the sharing of private computational resources. These functionalities facilitate the temporary lending of a computational asset owned by an individual, with the social promise that if that individual needs to borrow, the gesture will be reciprocated. This we refer to as Social Distributed Computing. [5] [6] [16] [18]

1.2 What is Social Distributed Computing

Social Distributed Computing at its highest level is a framework to allow computation to occur between friends on an Online Social Network. This means that some possible applications of the framework could include Application Sharing, and Resource Sharing. Application Sharing would allow users to execute their friend’s applications remotely and interact with them from their own machine, in essence having the friend temporarily share their application. Resource Sharing would allow friends to set an amount of their computational resource (CPU time, Memory, Hard Disk, Network Resources) to be shared, this would allow the creation of an environment that could be referred to as a Social Grid Computing framework.
The advantages of Social Distributed Computing are three-fold. Firstly, it allows for greater application diversity for the participants. Secondly, it provides additional computing power for those few times that you need a computationally difficult problem to be completed quickly. Thirdly, it enhances the use of online social networks by providing a new level of functionality not previously considered.

With the introduction of social application sharing, participants gain a much wider range of applications to use. In addition these social applications can be run from whatever workstation the user is currently using. Consider the normal workflow a user follows to acquire a new application when additional functionality is required. The first step would be to either go to a software store and browse the available products to find a commercial off the shelf (COTS) piece of software, or do research on the web to find an appropriate program that will perform the functionality required. Then the user must purchase a license if it is not an open source application, as many are not. With the next step being downloading the application, and installing it. All of this work to acquire a set of functionalities that may only be needed for a day. It would be much simpler if the user could just search his network of friends for someone who already has an appropriate application and is willing to lend the use of it for a period of time. With our system no copyright problems arise as the software being run is still executed on the owner’s machine and only one instance of the application is ever used at a time. We
simply redirect the input and output through the Social Distributed Com-
puting Service and allow the user to access them through a thin client such
as a web browser via their Online Social Network. After using a friend’s
application the user could decide to purchase a copy if they think they will
use it enough to make it worthwhile. This process we refer to as Application
Sharing, and is one of the two main applications of Social Distributed Com-
puting discussed in this thesis.

The second main application of Social Distributed Computing discussed is
referred to as Resource Sharing. The idea of Resource Sharing can be derived
from the advantages of Grid Computing. The idea is that when executing
distributed applications, we take advantage of many lower level computers
to do things that were once only possible by super-computers. In Resource
Sharing we have users volunteering to set aside a part of their computational
resources in exchange for the same treatment from their friends. This doesn’t
mean that the amount set aside cannot be used by the owner of the machine,
merely that if it is not in use and a friend requests to use that resource they
are given the opportunity. Consider the scenario where each friend sets aside
10% of their resources for use by other users. If the user would need over
90% of their resources only 5% of the time, the remaining 95% of the time
they could share their resources that are not in use. Since they still have
the capacity to use all of their resources when needed, they really are los-
ing nothing. Next consider the performance gains by participating in this
Resource Sharing system. The majority of the time that we run applications we don’t require a computationally difficult function to be executed, if we had additional resources when we did run these functions they could be completed in a fraction of the time. This could improve user experience and productivity. Some examples of this type of operation might include GIS calculations, video encoding/conversion, and file compression.

1.3 Thesis Overview

In this thesis, Social Distributed Computing is proposed as a new organization of computation that takes advantage of the pre-existing social links found in social networks. Leveraging these connections we can increase the diversity of applications available to each user, and provide additional computational resources when needed. Social Distributed Computing can be thought of as the next functionality to be included in social networks.

1.3.1 Objective

The objective of this work is to define the concept and models of Social Distributed Computing then detail a service based framework that realizes it. As this topic is a composition of several areas of computer science research, to define Social Distributed Computing we first must detail the work done previously in the areas that intersect to create this new topic. We will then
discuss the topic itself, from the ideological model to the architecture and design, we will look at what possible applications this topic has the potential to create.

The objective of this thesis will also be to implement a working prototype of Social Distributed Computing and perform some experiments on it. The example will serve more as a proof-of-concept than a definitive work with all functionality implemented. The thesis will also look into possible case studies where Social Distributed Computing can be applied to improve a social network’s functionality.

Another objective of this thesis is to provide a good starting point for future research into this topic.

1.3.2 Problems & Challenges

To create a Social Distributed Computing system there are several problems and challenges we must consider. One challenge for this thesis is to consider the possible applications of Social Distributed Computing and document as many unique possibilities as possible.

In order to make a proof-of-concept we first need to either create an Online Social Network or integrate with an existing social network. This will give us that ability to use the social connections between friends. Creating
an Online Social Network from scratch would be a considerable amount of work. However, there are multiple open source web-based social networks that can be freely used to create your own instance of an Online Social Network. The other option is to obtain the social network’s functionality by integrating into an existing social network. This has the advantage of adding the system to a service that millions of people are already using. However, it also means that we have less control over how we can modify the social network components of the system. Integration into an existing social network would also be fairly simple. Most social networks today are highly extensible through Application Programming Interfaces (APIs). Via these APIs an extension can, given the permission of the user, obtain their list of friends and send messages to those friends. This functionality is really all that is needed for a Social Distributed Computing system to interface with a social network.

We must also find a way to implement Application Sharing. Through Application Sharing, when the need arises users can find friends who have software they need and can request to make use of that functionality. This means we need a provider client application, a requester client application, and either a server application or a peer-to-peer framework. Ideally the client applications would be highly portable and could run in the web browser with minimal dependencies required. The provider client application would need to capture the output of the application to be shared and send that to the requester client. The requester client application would in turn capture the
input from the user and send that to the provider client to be applied to the shared application.

While the Resource Sharing component will not be implemented in this thesis, it will be a challenge to detail the ways in which it can be designed to work. This will be a part of Chapter 3, Social Distributed Computing Model, and Chapter 4, Architecture and Design.

1.3.3 Structure of Thesis

This thesis is divided into six chapters. The first chapter being the Introduction, containing the motivation behind this topic, the problems and challenges required to overcome, and an overview of the thesis. The second chapter contains background information on all the topics to be discussed in the thesis. These topics range from network architecture, social network analysis, and online social networks, to distributed computing and service computing models. The third chapter contains the main idea of the thesis, the Social Distributed Computing Model. In this chapter we will look at what Social Distributed Computing is and what applications it can be applied to. The fourth chapter discusses the architecture and design of the system, the functionality that is possible and how it can be realized. The fifth chapter contains information on my implementation of Social Distributed Computing. It describes how my system works, and what it can do. The sixth and final chapter contains a set of case studies which explore how different func-
tionalities described in this thesis could be implemented in the real world. It also contains two experiments that were written to test some performance metrics of the current implementation.
Chapter 2

Background

2.1 Client-Server vs. Peer-to-Peer Computing

Client-Server and Peer-to-Peer (P2P) Computing are two different models for computers to communicate with each other. In the Client-Server architecture, we have computers taking one of two distinct roles, the role of Server or the role of Client. The Server’s responsibility is to listen for requests coming from other computers. An illustration of this architecture is depicted in Figure 2.1. When the Server receives a request it then takes an action. An example of this is a Web Server which holds data and web pages. When a certain webpage is requested the server may run some software to access its database and file system returning a webpage as the result. Another example would be a Game Server that is responsible to facilitate multi-player games. Computers acting in the Client role on the other hand will initiate commu-
nication and generally make requests of the server to perform an action or return some desired information. An example of a client would be a web browser, or a File Transfer Protocol (FTP) client.

Figure 2.1: Client-Server Architecture - convenient yet requires horizontal scaling (creating more servers) as the number of clients increase.

In a pure P2P architecture on the other hand all computers have the same role. These computers are acting as peers. To make a comparison with Client-Server Architecture, each peer must act as both a server by fulfilling requests and act as a client by making requests of the other peers.

As you can see in Figure 2.1 the number of links required in a Client-Server architecture increases linearly with respect to the number of users; which
Figure 2.2: Peer-to-Peer Architecture allows for each computer to connect to one another, however this means each computer must have N-1 connections (where N is the number of computers) to have a fully connected network.
in this case are represented by the clients. While this architecture does require less links for users to communicate with each other, it produces a star topology which inherently means there will be a bottleneck at the centre, as well as a single point where failure means the entire system goes down. In Figure 2.2 we see that in a P2P architecture the number of communication links increases exponentially and can be calculated by the following formula.

\[
\text{Number of Duplex Links} = \frac{(\text{Number of Peers} - 1) \times \text{Number of Peers}}{2}
\]

(2.1)

In communications, a duplex link is a connection in which it is possible to communicate in two directions simultaneously. While the number of connections required increases, we also don’t see a single point where all communication must travel through. This provides better reliability and throughput.

Originally ARPAnet, the predecessor of the Internet, was created to link university computers. The ARPAnet came out of our frustration that there were only a limited number of large, powerful research computers in the country, and that many research investigators who should have access to them were geographically separated from them [17]. The purpose of the Internet was to create a network of computers, making it possible for any computer to contact any other computer on the network; However as time went on the network technology used in the Internet changed from the original Peer-to-
Peer architecture into a hierarchical Client-Server architecture. The reasons for this have more to do with network security and business organization than architecture principles. Today the Internet is made up of a many large public networks usually referred to as Wide Area Networks (WAN), and many smaller private networks called Local Area Networks (LANs). These LANs are often referred to as Edge Networks because they usually have few connections to the Internet and are best placed on the outer edge of a graph depicting the Internet, leaving room in the middle for more connected components.

The networks joined together to create the Internet often have artificial boundaries where they connect created with security in mind. Some examples of this would be firewalls and Network Address Translators (NATs).

Although NATs aren’t strictly designed for security purposes they have had security protocols added to them as the technology developed. The main purpose of NATs is to multiplex multiple communication streams through a single IPv4 address. Since the number of IPv4 addresses is limited we often must share them. When an Internet Service Provider (ISP) provides connectivity for a household they typically only give the house one IPv4 address that can be accessed on the WAN. Network Address Translators are required to be built into the modem to allow the many internet enabled devices within the household to communicate with the Internet without having their return
Figure 2.3: A Network Address Translator (NAT) multiplexing multiple computers over one public IPv4 address. Here we see how packet headers are stripped and replaced when traveling through a NAT.
traffic broadcast to all devices on the LAN. To accomplish this we use a section of IPv4 addresses that is reserved for small networks, see Figure 2.3. These numbers reserved by the Internet Engineering Task Force (IETF) for private networks are within the following three ranges:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

Since each private network can use these addresses to communicate between devices and still use the other ranges to access Servers on the WAN this works out well. When a computer from a private network sends a request to the WAN it must go through the NAT. The NAT remembers the sender’s LAN IP address and then wraps the packet with another header that indicates that it came from the one WAN IPv4 address assigned to that private network. It also includes a port number with the sender’s WAN IPv4 address. The NAT then redirects any responses on that port to the LAN IPv4 address that originally used it. This technique works very well when the server is in the WAN and is therefore publicly addressable. The client can easily send it a message, then the server can send a reply by directing it to the NAT/port number required to get back to the computer in the private network. However this architecture makes it very difficult for two devices that are behind different NATs to communicate. This is mainly because most NATs have
built in security protection. When a computer sends a request out through the NAT, the NAT keeps the used port open to wait for a response. However, if the NAT receives a response on a port that did not request anything the NAT will drop the packets, ignoring the communication and making it look like the LAN doesn’t exist. This is to prevent hackers for discovering private networks and accessing the machines therein. This causes challenges for P2P as the two clients cannot communicate without an intermediary in the WAN [12]. There are techniques to get around this, such as TCP Hole Punching [12], but they do not work with all implementations of NATs. This makes Client-Server communication more reliable than P2P in the Internet.

The downside to Client-Server communication is that we need more Servers as more clients attempt to communicate with each other. In P2P we don’t see this restriction making it much more affordable. For example, many gaming consoles such as the Xbox360 that allow for multi-player games use a P2P architecture instead of purchasing vast amounts of servers and hosting the games on those machines.

### 2.2 Ego-centric Social Network Analysis

When we talk about an ego-centric view of a social network we are always focusing on a specific person that we refer to as the subject, see Figure 2.4.
The view consists of the subject, the persons the subject is connected to, and the links that join the subject to others [32]. Much of this thesis will be looking at a social network from an Ego-centric viewpoint.

Figure 2.4: The bolded lines and users show an ego-centric view of the social network focused on the subject in red. Note that links between friends of the subject are not considered.

Traditionally Online Social Networks (OSNs) are constructed using Client-Server architecture; however in the past decade much research has been done on the feasibility of implementing OSNs using P2P architecture [6]. While the Client-Server architecture is the foundation of the web and thus highly accessible and well tested, it does have several properties that researchers find undesirable. The main property these Client-Server OSNs have that is troublesome is that the company providing the social networking service is privy to everyone’s personal and private information [29]. This company must store all this information in a centralized data warehouse which creates a honey pot for hackers. These companies, being for-profit entities, usually monetize
on the personal information of their clients by selling targeted advertising or worse selling personal information to interested third parties without their client’s consent or knowledge [6]. The benefit of a P2P implementation of an OSN is there is no central data warehouse, each peer is responsible and in control of their own personal data. This architecture by its nature is a less attractive target for hackers, and there is no company with access to all the data to sell the personal information for profit.

While P2P based Online Social Networks do have several advantages they also introduce challenges we don’t see in the Client-Server architecture. The most obvious of which is that there is no central server that is always online. This makes storing messages for other users more difficult if both users aren’t online at the time the message is sent. One solution to this is to store these messages encrypted on other peers machines using a redundancy factor [29]. It also makes friend suggestions more difficult as the friend lists are distributed and the entire social network cannot be seen by any one user at any time. This makes clustering algorithms more difficult to execute.

2.3 Online Social Network Functionality

Online Social Networking functionality defines what currently can be done using an OSN. This set of features includes elements of synchronous and asynchronous applications. The functionality of most OSNs can be broken
down into the following categories.

2.3.1 Asynchronous Messaging

Asynchronous Messaging is used when you want to leave a message for another user to read the next time they log in. It is analogous to snail mail as it can be sent at any time and when the user gets a chance to check their messages they will find if they have received one. This type of communication is often referred to as pull, as the user only finds out if they have new messages by actively requesting or pulling the data.

2.3.2 Public Asynchronous Messaging

Public Asynchronous Messaging allows for users to talk to each other in a public setting where more than two users can read the messages that were left. The amount of people able to see the messages could be anywhere from a small group to the public web. An example from current OSNs of this would be a user’s wall in Facebook or a user’s Twitter feed.

2.3.3 Synchronous Messaging

Synchronous Messaging on the other hand is used when two users are online at the same time. It is analogous to Instant Messaging (IM) applications such as MSN Messenger. This type of communication is often referred to as “push”, as the messages are pushed to the client as soon as they are received.
by the server. This feature was a later addition to social networks as people started to realize that since your social network contained a list of all the people you might want to chat with anyway you might as well add a feature to do so there.

2.3.4 Friend Searching

Most OSNs provide a functionality to allow a user to find other friends on their network. This could as simple as checking a user’s contact list by scanning their email client, or by a more complicated scenario such as suggesting friends via social network analysis. In either case the purpose is to allow users to find their friends that are already using the service and allow them to invite those who aren’t using it to start.

2.3.5 Event Planning

Event planning consists of providing a space to create and discuss upcoming events. The guest list can be open or closed depending on the users wish. Similarly the event can be publicly visible to friends or hidden to those not invited. The creator of the event can set the date and time as well as the location. Users can even have a reminder sent to them before the event.
2.3.6 Affiliation Networks

There are many ways that OSNs can determine how close two users are. The majority of the time this is accomplished by analyzing the affiliation networks that naturally occur in real life, as well as digitally on the OSN.

2.3.6.1 Groups

In OSNs as in real life social circles, people have a need to associate themselves into groups. These groups could consist of members of a club, colleagues in a business, or classmates from school [32]. The functionality required for a group is to be able to send asynchronous messages to the entire group, and provide a space for group discussion. The group discussion can be public or private and the membership can be open or closed depending on the creator’s desire.

2.3.6.2 Likes

Some OSNs allow a user to “like” something. This functionality is often used to provide the operator of the network more information on which to perform targeted advertising. While I suspect that in most cases this functionality was not included for other reasons than targeted advertising, it does provide a great source of social networking information. For example if two friends have selected that they like 100 things each, and have 80 things in common it might indicate they are more interested in viewing each other’s public posts.
2.3.7 Media Sharing

Often OSNs will allow users to share media such as photos and videos they took. In some cases OSNs will be more popular for allowing users post music they made such as MySpace. This allows the users to show what they’ve been up to in the real world and provides a public space to comment on each other’s creations. By allowing the users to “tag” friends in the pictures the social network can even obtain more affiliation network information at the same time as posting more content on each person’s profile.

2.4 Distributed Computing

Distributed Computing is an architectural model where multiple software applications in multiple locations work together to accomplish a single goal. The software running on a node often performs functionality that is different from that of other nodes. Together these nodes complete a process to create an information product.

Object Oriented Programming consists of objects containing both data as well as methods. An interesting way to think about Distributed Computing is as a software application that has objects that not only exist on your local PC, but on other devices within a communication network [30]. Since these objects perform specific jobs we can create new instances of them on more machines to perform load balancing. This also provides us with improved
reliability, as if a node in the distributed computing system goes down, another can take over its responsibilities.

“Possibly the most important potential advantage of a DCS [Distributed Computing System] is extensibility. Extensibility is the ability to easily adapt to both short and long term changes without significant disruption of the system. Short term changes include varying workloads and subnet traffic, and host or subnet failures or additions. Long term changes are associated with major modifications to the requirements of the system.” [30].

There are also cases where Distributed Computing is used as the application requires computation to be done in geographically different places. An example of this would be nuclear power plants and process control applications where hardware is inherently distributed and real-time constraints are very important [30]. An example where real-time requirements are more lax would be airplane reservation applications where by their nature they are also inherently distributed, however the system’s reliability is less critical [30].

### 2.5 Service Computing Models

While there are many different ways applications can be designed and executed in this thesis we will divide these applications based on four criteria:

2.5.1 Asynchronous vs. Synchronous

The difference between an asynchronous application and a synchronous application is whether or not the timing of communication is important. Synchronous applications run in real time and will react with the presence or absence of interaction or message at any given time. Asynchronous applications wait for an interaction or message to be received, the timing isn’t important.

2.5.2 Interactive vs. Non-interactive

When applications are designed to receive input at the start, take some time to process it, and then return output when the software terminates they are considered to be non-interactive. However, if an application is designed to require constant input while displaying the results back to the user it is considered to be interactive. An example of an interactive application might be a real time game such as car racing, while a non-interactive application might be something more like an application to plays video files.
2.5.3 Single User vs. Collaborative

An application can be designed to be used by one person or designed to be used by multiple people at the same time. An excellent example of a collaborative application would be a multi-player video game; another example would be collaborative document editors such as Google Docs. Of course we are much more used to the single user application model when it comes to computing. An example of this would be an email client, or web browser.

2.5.4 Single Point Execution vs. Distributed

With a single point execution application we have one computer process the instructions of the application. This is usually the case when we run a program on our laptop or smart phone. However there is also the distributed programming model where instructions and IO are sent to a network of computers, each to play a part in the running of the application. Several possibilities exist for Distributed Computing, the two most popular are to have nodes run the same instructions on different data, and have nodes run different instructions on different data. In the case of the former, this is usually referred to as Parallel Computing and an example would be the @Home projects such as SETI@Home. These projects have volunteers install their software which allows a controller to send data to each volunteer to be processed in the same way. In the case of the latter, we usually refer to this simply as Distributed Computing. It can have the different nodes running at the same time, but
doesn’t have to. Generally the idea is to take care of different parts of a problem on different nodes. There also exists the possibility to have nodes run the same instructions on the same data, and have nodes run different instructions on the same data, although these are used less frequently.
Chapter 3

Social Distributed Computing

Concept

3.1 Why do we need Social Distributed Computing?

With ubiquitous computing our computational resources are becoming more and more diverse and we are becoming ever increasingly connected through the social and technological networks we create. With these advancements, the time is right to consider a new computing paradigm.

Instead of hoarding our resources so they will be available when we need them, we can combine Online Social Networks and high speed communication networks to create circles of trust allowing us to pool some of our resources. By having multiple computers cooperate we can draw on vast
amounts of resources when we need something done quickly. We call this Resource Sharing and it in essence creates a social grid computing platform, allowing participants to host and deploy services on the grid.

Social Distributed Computing also provides the possibility to create a shared library of available applications that can be executed on demand. By letting users contribute the software they have to a pool, each participant benefits in the form of additional functionality.

### 3.2 Functionality at its Highest Level

At its most abstract level Social Distributed Computing is a service that allows friends to share computing resources over a social network. This could be in the form of sharing access to their applications with each other, or allowing others to use a portion of their computing resources to run their code. The aim of this is to provide an environment where all participants benefit from increased software and hardware capabilities.

#### 3.2.1 Application Sharing

Application sharing in a nutshell is a user borrowing the use of an application from a friend. This typically would occur when one user requires software that performs a certain functionality that they don’t currently have. The user can then access someone else’s machine that does have this functionality
and remotely run the application with input and output being redirected appropriately. The user in need can use Social Distributed Computing to locate a friend willing to share an application with the required functionality, and then access the required program running on the friend’s machine. An example of this might be a user who needs to do video editing but doesn’t have the proper software. Using Social Distributed Computing that person is able to access a friend’s computer and perform the task.

### 3.2.2 Resource Sharing

Resource sharing on the other hand has more flexible possibilities as to what it can be used for. The basic idea is that most users only require the full computational power of their machines a fraction of the time they use them. If they decide to volunteer use of some computing resources (CPU time, Memory, etc) to allow another user to perform some kind of distributed computing, they in turn will have additional resources when they need to perform computationally intense operations. Using this framework any distributed computing application could be run. Examples range from communication networks to cloud computing platforms.

### 3.3 Security Considerations

The security proposed in this thesis would be similar to that of the Turtle file sharing system [28]. “While existing peer-to-peer architectures with similar
aims attempt to build trust relationships on top of the basic, trust-agnostic, peer-to-peer overlay, Turtle takes the opposite approach, and builds its overlay on top of pre-existent trust relationships among its users. This allows both data sender and receiver anonymity, while also protecting each and every intermediate relay in the data query path.” [28]. In their article [28] the authors propose to create a social network to maintain security, based on social trust. For us, we already have a social network that can be used to let our users choose which friends they want to participate with. This could be thought of as making a logical level Peer-To-Peer network, or as they call in their paper Friend-to-Friend (F2F) communication. In this proposed system we are only considering ego-centric networks, meaning the user will only participate with friends they have on their social network. This is a simpler problem than a distributed file sharing system where users must interact not only with their friends but with all and any other user in the system. However, the Turtle technique could be used in future work to allow users to interact with friends of friends.

3.4 Social Network Integration

Integrating Social Distributed Computing into an existing OSN requires a middleware framework to facilitate application and resource sharing between friends, as well as a control mechanism to allow the users to control what they request/share and when. Preferably the middleware for sharing would
consist of user side code that would facilitate a P2P solution; this would ensure scalability would not be a problem. When we talk about application sharing we need several components; the first being a way for friends to advertise their needs and their available applications. Another component is required to allow users to control what they share, with whom, and at what times. The final component needed would facilitate the actual sharing between friends.

Advertising applications that are available and requirements could be done in multiple ways within the confines of an Online Social Network. One option being that users list applications they own that they are willing to share and the users requiring the applications perform searches for appropriate programs. Another option would be for users requiring software to make a public announcement to their friends of their software needs, the friends can then respond if they desire to share an application.

When we talk about control mechanisms there are many considerations. The system needs to provide the user with the option to share certain applications, computing resources, or both. Since this system is based on social trust it is important to also allow the user to select what users they will share their resources with. The system must also make it simple for the sharing user to indicate what times they are willing to share their resources. If the system makes it inconvenient for the user to share, the system as a
whole will fail as its value is based on the amount users share with each other.

The final component is to facilitate the actual sharing process, this could be redirecting input and output of an existing application running on a friend’s machine, or it could be passing executable code and data to be processed around and returning the results. This part of the process would be resource intensive for the servers in a client-server architecture. That is why a proper implementation should attempt to do this in a P2P framework.

### 3.5 What is the Service Computing Model

The service computing model can be more easily considered in the context of business than computer science. In this context the word service is used in the sense of goods and services. The “as a Service” model takes an entity that used to be a good, something you would sell someone for a one-time fee such as a server, and turns it into a service, such as web hosting. A non-computing analogy would be managing a boarding house. One option is to buy all the equipment, and hire all the staff that have the required knowledge and experience required to open a kitchen. This would have a lot of one time fees and how much you spend each year would be difficult to predict. Another option would be to have a catering service provide the residents with meals. This would mean a set contract with predictable payments. In essence instead of buying a lot of goods, you are now paying for a service.
3.6 Social Distributed Computing as a Service

Distributed computing can be made into a service in much the same way as our catering example. Instead of purchasing multiple computers and having them at the ready when you need to perform computationally difficult tasks, you can commit a small amount of your resources for others to use when they need them; and in return you are allowed to use a small amount of resources of multiple friends when you need them. This makes more efficient use of our computing resources in general as the time we need 100% of our resources are few and far between. However, if in those instances we were given 10% of 50 friend’s resources we would complete our tasks significantly quicker. Think about mobile computing, the most precious resource a mobile device has to conserve is battery power. If we could pawn off computationally difficult tasks off our mobile devices into a Social Distributed Computing Service made up of computers that have constant power supply we could conserve battery power and extend the battery life of our mobile devices.

We can think of the benefits of sharing software in much the same manner. Consider the situation where each friend has ten applications on their computer. If we hoard access to our applications we will have ten applications that we can use at any time. However if we each share access to our applications the amount of unique applications we have access to will increase and our productivity will also increase with our new abilities.
3.7 How Distributed Computing is realized as Social Distributed Computing

Distributed Computing is the process of accomplishing a task by running software concurrently or in parallel on different machines, then aggregating the results to form the final information product. Parallel Computing can be thought of as a subset of Distributed Computing where the software is always run concurrently, often on different processors within the same machine, all with access to the same main memory. Social Distributed Computing provides the means to accomplish both of these computing models with only the hardware within your social network.

There are several ways to facilitate Distributed Computing; the differences mostly have to do with the organization of how the distributed units communicate. In Social Distributed Computing we always start with an ego-centric social network. Beginning this way we have the most obvious means for communicating, creating direct communication links through the service between the user and participants. We can then have the user’s machine organize what work is done by whom and compile the returned results. Another possibility is to have the service perform the organization of what computer does what work based on its knowledge of what participants’ resources are available. In this model the user would provide the input data and the service would serve as a liaison agent who organizes what participants do what work, receives
the output from participants, compiles output and returns the result to the user.

Since 2007, there has been a trend in CPUs away from growth via ever increasing clock speeds and towards growth via parallelization by creating multi-core units [27]. This is mainly brought on by the hardware limitations incurred by increased clock speed, including increased power consumption, and increased design complexity [27]. Unless there is a radical new discovery it is safe to assume that this trend towards parallelization will continue. Social Distributed Computing provides a way to take the next generation of applications which will be already created to be executed over multiple processing units and execute them with the power of an entire social network of machines.
Chapter 4

Architecture and Models

4.1 Use Cases

When users interact with the Social Distributed Computing service we can consider the user taking on one of two roles, Requester and Provider. Each user will at different times take on one of these two roles so any of the six use cases from the diagram below in Figure 4.1, can be considered possible for each user.

In Figure 4.1 we see there is an option for the provider to register an application they have installed on their computer as available to be shared. This could be set up so any application they have registered is available for friends to use without asking, or it could be set up in a way that friends can see the applications available and request to use the ones they want. We
Figure 4.1: Use case possibilities for the Social Distributed Computing system broken down into the roles of requester and provider.
could include an option to allow the provider to set the number of instances of a particular application that can be shared at any time. There is also a use case for the requester to perform a search for suitable application their friends have installed that could be requested. The functionality for application discovery could be done in several ways including different workflows. Another possibility would be to have the requester post a written request for an application with certain functionality, then the provider could reply to the request if they have a suitable application.

As mentioned above there is also functionality to allow the user to request the use of an application one of their friends have listed as available. Once this is done the provider can either choose to accept the application sharing request or reject it by using the respond to application sharing request use case. If the request was accepted the provider can include conditions as to when and how long the application can be used for.

The remaining use cases are for resource sharing. They allow the user to select an amount of their CPU, Memory, Hard Drive, and or Network resources that they would like to allow others to use when they are not currently in use. In return the provider gets the same service from those of their friends that also share resources. By using the requester’s “run applications using shared resources” use case, the user is able to run their application over a grid, or take advantage of the shared resources in other ways such as simply storing
some files on their friend’s computers. This can be done via a distributed hash table data structure providing more reliable access via redundancy and distributing the stored data over multiple machines.

4.2 Architecture Design

The Social Distributed Computing architecture relies on three main modules, see Figure 4.2. The first module is called the control module and it creates a way for users to discover software and resources that are being shared by others. It also allows users to declare what they are willing to share, at what time, and how much of their resources they’re willing to donate. This module must integrate fully with the social network to allow users to access their friend’s lists and communicate with each other. Since this module provides the main interface to the system it needs to be implemented in such a way that it can be accessed along with the rest of the OSN.

The second module is called the application sharing module. It facilitates the IO (input and output) redirection required to let one user share an application with another. This IO could be in different forms depending on the application being shared. There would need to be support for command line, GUI, and perhaps other forms of interface in the future such as audio or gesture based. Each interface type would have different IO redirection needs. The module is also required to start and stop the application being shared if
Figure 4.2: Social Distributed Computing Modules
there is a given time that application is to be shared for.

The third module is called the resource sharing module and is responsible to facilitate the lending of computational resources. This has two main functions. The first is to allow the transport of data and code to the remote computers and return the resulting output to the user to be compiled into the final result. The second is to perform accounting functions to keep track of who is sharing what, and how much of it they have shared. Using the accounting functions there is the possibility of including an economic system of trade where users gain credits by allowing others to use their resources [1], or it could be an opt in system where anyone who has resources shared is entitled to use their friends resources.

4.3 Social Computing Models

My solution involves the creation of four new computing models that merge social networking with distributed computing. A middleware solution was designed and implemented to test the concept of social computing.

4.3.1 One-to-One Social Computing Model

The One-to-One execution model, see Figure 4.3, combines the Single User and Single Point Execution computing models. It allows one user to execute an application on one machine. In our proposed solution this would refer to a
user running one of their friend’s shared applications on the friend’s machine. All input would be sent through the middleware to the friend and output would be redirected back through the network to the user. This One-to-One model could be Interactive or Non-interactive, and could be a Synchronous or Asynchronous application depending on the needs of the application.

4.3.2 One-to-Many Social Computing Model

The One-to-Many execution model, see Figure 4.4, combines the Single User and Distributed computing models. It refers to one user running an application using many of their friend’s machines as points of execution. In our proposed solution, we foresee this happening in three ways. The first being that each friend would have a copy of the program and be provided some data to process it. The second would be one user having multiple applications that they need to have work together to achieve a desired result. If these applications are on multiple friend’s computers we could have the Social Distributed Computing service send the output of one application to the next. An example of this would be to have one friend convert an image into a format that is needed by another friend’s program so that we can do some photo editing. The third would be that data and instructions are passed together to the friend’s machines who have agreed to donate some of their resources. Obviously the second way would be more generic and would work very well for applications running on a virtual machine such as Java or .NET.
Figure 4.3: One-to-One Social Computing Model
4.3.3 Many-to-One Social Computing Model

The Many-to-One model, see Figure 4.5, combines the Collaborative and Single Point Execution computing models to allow multiple users to run a shared application on one friend’s machine. In this model our middleware solution would keep all users in sync with the latest state of the application being used, as well as facilitate the interactions between the users and the application.

4.3.4 Many-to-Many Social Computing Model

The Many-to-Many model, see Figure 4.6, combines the Collaborative and Distributed computing models to allow multiple users to run a program each
Figure 4.5: Many-to-One Social Computing Model
contributing some resources. This model could be useful in collaborative applications that allow many users to participate. Consider online gaming, in most cases the way this is accomplished is using a Many-to-One model where many gamers connect to a server and the server can only accommodate a finite number of users. If the responsibilities were distributed into a Many-to-Many model no server is needed. While the overhead for the game provider does decrease, do note that another bottleneck may occur where there would be additional network communication required to keep all users in sync. Consequentially having a Bandwidth bottleneck instead of a CPU and Memory bottleneck on the server could be a good thing or a bad thing depending on the needs of the application.
Figure 4.6: Many-to-Many Social Computing Model
Chapter 5

Implementation

While designing my system I tried to keep two design principles in mind, KISS and YAGNI. Most software developers are familiar with KISS, it stands for “Keep It Simple, Stupid” and means do not over design your application. There isn’t any reason to make complex functionalities when a simple way to get something done is better. The second principle, YAGNI, stands for “You Aren’t Going to Need It”, it advocates only writing code for the functionality you currently need instead of thinking of all the possibilities down the road. “Build what you need as you need it, aggressively refactoring as you go along; don’t spend a lot of time planning for grandiose, unknown future scenarios”[2].

I also wanted to use open source software as often as possible, this was one of the reasons I decided to use a Ubuntu based LAMP (Linux, Apache, MySQL,
PHP) configuration for my server. Since LAMP is one of the most popular configurations for web servers it also made my life easier when trying to find third party software that I could use to implement my system.

For my implementation I realized that I would need a social network running on my server to implement Social Distributed Computing in. After researching several alternatives I discovered that an open source social network called Elgg would be my best choice. I went with Elgg mostly for its extensibility. Mostly all of the social network functionality in Elgg is implemented as “mods” which are basically plugins that you install onto your instance. Since new mods can be created in the same way as the existing features I had lots of example code to look at when implementing my Social Distributed Computing mod.

For my system I decided to implement the Application Sharing and Sharing Control functionality leaving the Resource Sharing module as future work. After creating my Social Distributed Computing mod which allowed users to control what applications they share, with who, and when, I needed a way to allow applications to redirect their IO over the social network. I considered several P2P methods for communicating between users such as Unmanaged Internet Architecture (UIA) [14], and JXTA (pronounced Juxtapose) [34], but I ultimately decided that for this implementation reliability of use was more important than low server resource consumption. For this reason I
decided to go with a classical Client-Server architecture.

5.1 Application Requesting

The Application Requesting process implemented is a fairly simple mechanism. First the requester enters the Applications section of the Online Social Network (OSN). There they can see application requests that have been filled or proposed applications to fill requests. There is also a button to create a new application request. When the user clicks to create a new Application Request a form is displayed on the screen. The purpose of this is to get as much information from the user about the application that they need. The implementation uses a form instead of free form text to hopefully get the user thinking about different aspects they need to include in their request, see Figure 5.1.

After the form is completed the user selects submit and the information from the form is transformed into free form text where the user has a chance to edit or include other information they didn’t yet get a chance to mention. The user then selects submit and the request is publicly submitted to all of their friends by being placed on the activity feed, see Figure 5.2.

When a friend sees the application request and decides they have an application that they think would be suitable to meet the needs of the request,
Figure 5.1: Form to request an application to be shared

Figure 5.2: Application request posted on activity feed
they can click the request and select reply. They are then presented with the same form the requester filled out, this time defaulted to contain the information about the request. The provider then makes the appropriate changes to reflect the application they are proposing to share and clicks submit. The information is once again converted to free form text and the provider is given the chance to modify that text before it is sent to the requester. Once the information about the proposed application is sent to the requester they receive a notification via an unread indicator on the website (see Figure 5.3), and through an email.

When the requester receives the information about the proposed application to be shared, three possibilities occur. They either don’t have enough information about the application, decide that the application is acceptable, or decide that the application is not acceptable. In the case that the requester requires more information they have an option to click a button called request more information. After doing this a list of fields from the form is displayed with checkboxes next to each. The requester can check off fields they would like more information about and include a message in a text box.
Figure 5.4: Request more information interface displayed. An example of this procedure can be seen in Figure 5.4. The request for more information is sent off to the provider and they are notified in the same manner as before.

In the case where the user decides the application proposed is not applicable they can simply ignore the response. If the requester decides that the application satisfies the requirements needed they can click a button to signal that the application sharing can now begin, a notification is sent to the provider once again through unread indicator located on the top bar of the website (as seen in Figure 5.3) and via email.

After the application is selected there will be a new entry in the accepted
applications section of the Applications page. There are three subsections of the applications selected section, one for asynchronous application sharing, one for synchronous Command Line application sharing, and one for synchronous GUI application sharing. When either user clicks the new entry in the application selected section they will see different things based on the actions they must take. These actions will be described in the following sections.

5.2 Command Line Application Sharing

To redirect standard command line IO, I decided to write my own Java based server application using an application sharing GUID (Globally Unique Identifier) that was assigned by my Elgg mod to distinguish between sharing instances. The service consists of three Java applications, one for the provider, one for the client, and one running on the server to facilitate the connections between the other two. The provider’s application consists of a runnable jar file, and an XML configuration file. The configuration file contains the GUID for the application sharing, as well as the negotiated start and end times. An example of the configuration file can be seen in Figure 5.5.

The provider then puts the jar and configuration file into the folder with the executable they wish to share and runs the jar file specifying the executable’s filename as a command line parameter. The jar creates a process
and loads the executable, then create two threads and a socket connections to the server. One thread to listen to the socket for incoming data to use as input to the process just started, and the other gather output from the process and send it through the socket through the server to the requester.

The requester’s application consists of a Java Applet that is hidden within a web page. The applet communicates with Javascript to allow for a HTML interface to display output and accept input. To the user it appears to just be a normal webpage. On initialization, the applet requests a socket connection to the server. If the server does not already have a connection to the provider the requester is notified. If the server has a connection from the provider that it accepts the socket, a sequence diagram of the complete interaction can be seen in Figure 5.6. A UML class diagram representing the Command Line Application Sharing server can be seen in Figure 5.7.
Figure 5.6: Sequence diagram showing the process the two Command Line Application Sharing clients go through to interact with one another via the server.
The requester application then creates two threads. The first listens for input entered into the webpage and sends that input to the provider via the server. The second listens for output sent from the provider and displays that on the webpage.

The server application must listen for socket connection requests from both provider as well as requester. When the provider contacts the server, the server accepts the socket, and then passes it along to the connection manager who holds connections and looks to pair providers with requesters. Eventually the requester will contact the server, the server will create another socket, and pass it along to the connection manager where it will be paired the appropriate requester. Once this happens a thread is created to listen to each socket. When something from the provider is heard it is written to the requester’s socket and vice-versa. Once either socket is closed all threads related are terminated and the other socket is closed. A similar process occurs for providers that have initiated their connections and then stop sharing before the appropriate requester attempts to connect to them.

5.3 GUI Application Sharing

The second type of application sharing implemented was that for Graphical User Interfaces (GUIs). To do this I needed to find software that would let
Figure 5.7: Command Line Application Sharing Server class diagram
me capture the section of the screen the application was running on and transmit that through the Social Distributed Computing framework to the requester. This would allow them to see the current state of the application. For the requester to be able to control the application I needed to be able to send the mouse clicks and keyboard presses back through the framework to the provider’s computer and simulate actual mouse and keyboard input on that system so the program would respond.

By searching the web I was able to find an open source application called Red5 Screenshare that is implemented using Java Web-Start. This system provides all necessary functionality. The fact that it was created to be a Java Web-Start application made it even more applicable for this use, as I wanted to keep the functionality as close to within the web browser as possible.

Since the application was open source I was able to modify it, letting it accept the meta-information required to transmit the video and control streams through a channel specific to the current application sharing instance.

The system required the Red5 Media Server to be installed on our web server to act as an intermediary between provider and requester. On the requester’s side all that is needed is an Adobe Flash application that displays the graphical interface, accepts keyboard and mouse input and transmits the input to the provider via the server. I was able to set up the requester’s side to
Figure 5.8: Sequence diagram showing the interactions between the two GUI Application Sharing clients and the Red5 Media Server
automatically interact with the correct channel for this application sharing instance as well. This solution allowed the application sharing to occur without either party needing to install any software, provided that they had Java installed, and a browser that could support Flash. A sequence diagram of the interactions can be see in Figure 5.8.
Chapter 6

Case Studies and Experimentation

In this chapter we will explore several practical applications of Social Distributed Computing detailed in the case studies. This will provide some real life examples of Social Distributed Computing to help understand the utility of the research. Some of these case studies have been implemented in our system, while others have been left for future work. In the experiment subsections we will examine the efficiency of the given implementations.
6.1 Case Study: Command Line Application Sharing

The system for Command Line based Application Sharing was designed and implemented as a part of this thesis. It consisted of three separate applications, one for the requester, one for the provider, and a server application to allow these two to communicate. The implemented solution uses an executable Java JAR on the provider side to launch the desired application, redirect IO to the server to be relayed on to the requester, and receive the incoming IO sent from the requester via the server. The interface to start the application sharing is quite simple. First the requester would access the Applications section of the social network and place a request to their friends. They would include as much detail about the needed application as possible such as name, category, platform, interface type, synchronous or asynchronous, and the dates required for the start of sharing and end of sharing. Then the user’s friends would see the request and if they had a suitable application they could make a form response using similar information but modifying it to be appropriate for the application they intend to share. The user may get multiple responses and then will select one to start the application sharing. Once the application sharing has been confirmed the requester and provider open the page that contains that application sharing trial. The provider will see a link to download the provider JAR, a custom XML configuration file with the details of the transaction, and instruction
on how to run it. The requester on the other hand just sees a web interface containing an output section and a line to enter input in. This is controlled by JavaScript and a Java Applet. The JavaScript makes calls to the running applet which in turn relays input through the server to the provider. The Applet is also responsible to be listening to the server for output to display on the webpage via JavaScript calls.

In this case study the requester has a video file that they need converted. The requester submits a request to his friends for an application to convert the file to the required format. Luckily one of their friends has a command line video conversion tool and agrees to allow them to use it. In Figure 6.1 you can see what the requester’s screen looks like once the command line application sharing has started. They have a large area to see the output of the application and an area to enter input.

In Figure 6.2 you can see what the provider needed to do to set up the sharing. In this case the provider wasn’t sure what command line arguments the requester wanted to use so since they trust one another that person decided to just share an instance of Window’s Command Prompt (cmd.exe) so the other user could execute the program as desired. Another approach would have been to request further information and then just run the application requested through the sharing interface.
Figure 6.1: The requester’s view after command line Application Sharing has been initiated

Figure 6.2: The provider’s view of command line Application Sharing
6.2 Case Study: GUI Application Sharing

GUI Application Sharing was implemented into the system using a number of technologies and open source applications. As the scope of this thesis doesn’t include research on improving existing screen sharing technologies, I started the work by searching for open source screen sharing tools. I wanted to find one that used existing web technologies so the implementation would stay as close to in the browser as possible. I was able to find an application called Red5 Screenshare written by Dele Olajide [26] that uses Java Webstart to launch an executable JAR through a JNLP file. This JAR (see Figure 6.3) launches an application that is used to give the user control over what area of the screen is shared and at what time.

Since the application is open source I was able to configure it to send the IO on a pre-determined stream, specific for that Application Sharing trial. This allows multiple users to perform Application Sharing at the same time and ensures that the only people who see the application being shared are the provider and requester. The JAR captures the screen in the form of a RTMP video stream and sends it along to the Red5 Media Server installed on our web server.

This implementation came with a flash based viewer for the requester to view and control the application being shared. Since it is flash, it can be run
Figure 6.3: JAR that allows the provider to share a part of their screen
Figure 6.4: Application requester uses an Android tablet to control the shared application right inside the web browser which is perfect for my web based OSN. For this case study I decided to have the requester use an Android based tablet (See Figure 6.4).

The application being shared in this case study is an open source geographic information system (GIS) application called OpenJUMP. The performance was very good in that there were no missing frames, however when run on the tablet there was a noticeable delay that did not occur in trials when the requester was using a PC.
6.3 Case Study: Parallel Computing

6.3.1 Parallel Computing Background

Parallel Computing is a way of solving a computational problem in which it is subdivided into many smaller problems to be worked on concurrently. These instances can be worked on within a single computer on multiple cores, or could be distributed over a grid of networked computers. An example of this would be the @Home projects where users volunteer some of their computer’s down time to run scientific applications in the goal of accomplishing something. The most popular @Home project would be SETI@Home SETI standing for Search for Extra-Terrestrial Intelligence, the project listens to the electromagnetic spectrum and then transmits that data to volunteer computers for analysis to look for patterns of communication of extra-terrestrial intelligence. The EM Spectrum is so vast that it would take a very expensive super computer to even attempt to do all the analysis, but by breaking it down and sending it to computers at home they are able to perform much more analysis than they would have been without it.

6.3.2 Case Study

In this case study we have a user playing the board game Go with an artificial intelligence opponent, implemented using a parallel algorithm. The Social Distributed Computing system allows the user to have the AI work
be completed by multiple friends computers. In this case the organization of the parallel computation is done in a two tier architecture. The main work is done by the bottom tier computers. To accomplish this, data about the current board configuration as well as code instructing how to calculate the possible moves is passed on to the bottom tier computers. When those computers finish their job their output is passed back to the Social Distributed Computing service which passes the output on as input to the second tier computer. This computer combines the output of the lower tier and decide on the optimal move.

This case study can be completed with the Social Distributed Computing service proof of concept if we write a small resource sharing module which would pass Java code and data from users to friends who have agreed to run that code. The friends would only need to load a webpage that includes an applet very similar to the command line synchronous applet already existing in the system. The applet would connect in the same way and then code and data would be sent back and forth by sending objects and data through the existing socket connections, a task that is very simple in Java. Any object can be sent through a socket if it implements the java.io.Serializable interface.
6.4 Case Study: Distributed Computing

6.4.1 Distributed Computing Background

Distributed Computing has some similarities to Parallel Computing as it is performed on multiple machines, however in the case of Distributed Computing we see different jobs be performed by different machines, and sometimes performed at different geographic locations. The jobs don’t necessarily have to be executed at the same time, many distributed applications consist of jobs being performed one at a time sequentially. Others are organized into a network of nodes where the result of one node triggers an action in a connected node. There are even distributed systems where the node topology matches the topology of the problem that is attempting to be solved.

6.4.2 Case Study

In this case study we have a user who needs to do some photo editing, in particular he needs to merge two photos together to create a panorama. One of his friends have an application that does this task nicely, however it does not work on the image format the images are currently encoded as. Luckily another friend has an application that can do the conversion. Using the Social Distributed Computing framework the user is able to borrow the use of both applications and configure the system to feed the output of the conversion program into the merging program as input.
This can be done using the current implementation of our Social Distributed Computing proof of concept using the Application Sharing module. We simply need to add a little functionality to the control module. We add the ability to have the service receive the output of the first application and send it along to the second application and this case study becomes a reality.

6.5 Case study: Social Cloud

The social cloud is the idea of creating a cloud computing platform that would deliver SaaS (Software as a Service), PaaS (Platform as a Service), and possibly even IaaS (Infrastructure as a Service). The idea is that by each friend allocating certain resources a cloud could be created that would allow for pay model revolving around resources provided granting credits and resources consumed costing credits.

6.5.1 Cloud Computing Background

Cloud Computing is a method for running services over multiple physical machines. It abstracts the physical layer allowing for services to dynamically adjust their resource consumption over one or more machines. The three main forms of Cloud Computing are described below.
6.5.1.1 Software as a Service

In Software as a Service (SaaS) the service provider allows the user to choose from a set or market of software that can be provided via a light-weight interface such as a web browser. Once the user decides what software they will require for themselves or their company the provider allows access to those applications that will automatically scale depending on the number of users accessing those services at any time. This eliminates the need to own, maintain and administer multiple servers on the user’s end, and the user only needs to pay for the amount of resources they consumed [31].

6.5.1.2 Platform as a Service

Platform as a Service (PaaS) is a concept where a service provider creates a platform that individual users can deploy their software onto. The PaaS platform can automatically provide additional computing resources, and create additional instances on other servers to allow for seamless scaling as more users or less users attempt to use the software. The two main advantages of PaaS from the user’s perspective are automatic scalability and not needing to maintain and administrate servers. The advantages from the provider’s perspective is that since the use of these software applications will fluctuate, they can have many applications running on their bank of servers, dynamically allocating resources to get optimized use of their hardware. Since the pay model is on a resource consumption basis, the more the software is used the more money the provider makes [1].
6.5.1.3 Infrastructure as a Service

Infrastructure as a Service (IaaS) is concept similar to SaaS and PaaS except instead of allowing a user to deploy applications the service provider allows the user to allocate servers either as physical hardware or more often as virtual machines. This allows the user to install any operating system they wish and configure it in the manner they require. In the case of virtual machines they are in essence files stored in the cloud, hence they can be duplicated and run on multiple servers in different locations around the world to provide reduced latency, and load balancing when required. Again the advantage from the user’s perspective is that they don’t need to own servers around the world, they can rent them at a fraction of the cost it would take to buy, maintain, administrate, and upgrade them. The advantage of the provider is similar to SaaS, as the service provides virtual machines, there can be multiple VMs running on a single machine and they can be given more computing resources as needed. The pay model is still on a per resource used basis so the more the virtual machines are used the more money the company makes [1].

6.5.2 Case Study

The Social Cloud could be implemented using our Social Distributed Computing proof of concept fairly easily if we already developed the Resource Sharing module as mentioned in the Parallel Computing case study. We
could run a distributed application that would provide SaaS, PaaS, and/or IaaS by sending objects through sockets to our friends. These objects would serve as a platform providing applications in the case of SaaS, it would allow users to deploy their own applications in the case of PaaS, and would allow users to deploy Virtual Machines (VMs) in the case of IaaS. The largest amount of work to implement Social Cloud Computing on our framework would be to write the Social Cloud service itself.

6.5.3 Social Cloud over Distributed Social Computing Platform

This social cloud could be implemented as an application running over Social Distributed Computing as seen in Figure 6.5. To do so we would need to deploy an application that would provide the basic cloud computing functionalities such as resource accounting, load balancing, and providing a platform for software execution such as a virtual machine. Since we are not using dedicated servers with 99.9% uptime we would require implementing these functionalities using existing technologies that specialize in redundancy such as Distributed Hash Tables (DHTs), or Distributed Databases for data storage to store and communicate data. We could make use of runtime environments such as .NET or Java for execution of code that is passed between peers. Using these techniques with an appropriate redundancy factor to allow for machines being turned off or disconnected from the network, we could
Figure 6.5: Here we can see how Social Computing could be implemented using the Social Distributed Computing Platform.
implement a social cloud over the Distributed Social Computing platform.

6.6 Experiment 1: Application Sharing Network Latency

In the following two experiments we will see the Application Sharing implementation be tested using a few metrics for performance. The information derived from these experiments can serve as feedback to how well the proof of concept is implemented.

6.6.1 Purpose

The round trip time of this system can be defined as the amount of time it takes to send a command from the requester’s machine to the provider’s and receive a change in the output from the provider. This process can be broken down into many phases as illustrated below in Figure 6.6.

The application sharing round trip can be broken into the following tasks which each take an amount of time:

1. T1: The requester transmits the command over the network towards the provider via the server.
Figure 6.6: The tasks that are required in performing a round trip of the application sharing software

2. T2: The server’s application sharing thread waits to get time on the CPU, then accepts the command through the socket and retransmits it towards the provider.

3. T3: The command travels through the network to get to the provider.

4. T4: The provider’s application sharing client waits to get a turn on the CPU, then receives the command, enters it into the running application and transmits the resulting output towards the requester via the server.

5. T5: The output travels through the network towards the server.

6. T6: The server’s application sharing thread waits to get a turn on the processor, then receives the output and transmits it towards the requester.

7. T7: The output travels through the network to the requester and is
The purpose of this experiment is to measure the system latency of our implementation by calculating the time a command waits in our server and provider during a round trip (T2 + T4 + T6 from Figure 6.6). We aren’t interested in the network latency as much as the system latency as we don’t control how packets are sent through the Internet. However, the total round trip times will be indicative of whether the system as a whole provides an adequate level of service. If the system is too slow the users will not be willing to use it. For this reason another purpose of this experiment will be to observe the overall latency experienced by the users.

### 6.6.2 Background

The Latency in this experiment can be attributed to multiple sources. Typically, the Network Latency is defined as the amount of time required to send a packet through the communications network. It is composed of the Propagation Delay, the Transmission Delay, and the Routing Delay.

Propagation Delay is defined as the amount of time it takes for a signal to be sent through a communication medium and is based on the speed the signal can travel in that medium and the amount of distance that need be traveled.

\[
Propagation Delay = \frac{Distance To Travel}{Velocity Of Signal Within Medium} \tag{6.1}
\]
Transmission Delay relates to the amount of time it takes the modem to transmit the data onto the medium. In most circumstances this is the largest component of Network Latency. The speed at which a modem can transmit is generally how we describe the bandwidth of our communications network. Take for example, a 100Mbps (Megabits per second) connection, this means that 100 million bits can be transmitted onto a medium per second. This does not relate to the speed the signal propagates within that medium which is measured in distance/time.

Routing Delay is a component of Network Latency that exists only in packet switching networks. It is defined as the amount of time the data packet must wait in queue after being received at a router before the router is available to retransmit the data onto another network connection.

In this system, the delay consists of both Network Delay, and System Delay. System Delay can be thought of as the amount of time between when a computer receives a packet on its modem and when it takes the appropriate action.

\[
Latency = PropagationDelay + TransmissionDelay + RoutingDelay + SystemDelay
\]

(6.2)
Latency = NetworkDelay + SystemDelay \quad (6.3)

6.6.3 Hypothesis

I expect the System Delay to remain relatively constant for all trials while the network delay should vary based on network topology.

I also think that the latency experienced will be within a reasonable amount for all trials. I would define a reasonable amount of latency to be between 0ms and 500ms for a one way trip. I also expect the trial that includes a wireless hop to have the most variance.

6.6.4 Experiment

To scientifically calculate this latency we have several options. The first that comes to mind would be transmitting the current timestamp from the requester’s client to the provider, and check the discrepancy. While this would directly measure the amount of time it takes for a command to travel one direction, this would be an inaccurate approach as the system clocks on both machines are very likely not perfectly synchronized. To be able to compare the system clocks one would need to send a packet through the network which of course would be affected by the current delays incorporated with the latency. The second and more reliable way to test the latency is to test the
return trip latency. By doing this we will be testing a timestamp against the same system clock that generated it eliminating the clock synchronization problems incorporated in calculating elapsed time. To get a reliable measure of the latency, for each trial we will be sending 100 round trip commands and recording the amount of time each command took, with this data we can get the mean latency as well as the standard deviation for each trial. We will then perform a 100 ping test between requester and server, and between provider and server. This will allow us to obtain the mean network latency and standard deviation for that trial. Using these two numbers we will be able to see how much the System Delay affects the latency.

To get a variety of network conditions we will perform multiple trials, each with different network topologies. The first of which will be to execute the experiment while within UNB’s private network. The second will be within UNB’s private network while on WiFi. The third trial will be within the same city but on a separate ISP’s network. The fourth will be from a city in another province of Canada. The fifth and final trial will be from a country on another continent.

6.6.5 Analysis

Using Figure 6.7 we can see the amount of Latency incurred by the system when used from different locations, and thus via different network topologies.
Figure 6.7: In this figure we see Total Latency to perform a round trip vs. Network Latency to perform the round trip. These are the two values measured by the experiment. Each trial was executed 100 times, averaged, and the standard deviation was calculated, which you can see in this figure denoted by the error bars.
Figure 6.8: This figure has the same scale as the previous, and shows how much System Delay fluctuated in our experiment by subtracting Network Delay from Total Delay.
If we look solely at the red bars, which indicate purely the network latency measured using TCP Ping commands, we see the results conform mostly to what we would expect: UNB Wired < Fredericton, NB < Sydney, NS < London UK < UNB Wireless. The nodes topologically closer incurred less Network Latency with the exception of UNB Wireless, which I would propose incurred larger latency due to the wireless hop. Of course wireless latency depends on many factors, such as the wireless standard used (802.11g/n, 802.13 etc), the amount of wireless devices attempting to contact the router, the Signal to Interference ratio which dictates what encoding (bit-rate) can be used etc.

If we take a look at Figure 6.8, we can observe that the orange bar representing the amount of latency derived from non-network causes, A.K.A. the System Delay; fluctuated quite a bit in these trials. If we look at the two most extreme cases, UNB Wired, and Fredericton, NB, we can see a jump of almost 100%.

6.6.6 Conclusion

The conclusions we can draw from this experiment are that indeed the latency of the Wireless connection did vary more than most trials, as expected. This is visible by looking at the standard deviation bars on Figure 6.7. The variance that was not expected however, was the trial from London, UK, who varied almost as much. I am unsure via what path the packets were
sent from London to Fredericton; however I will speculate that the increased number of hops lead to packets being sent on an increased number of different routes through the Internet. This is caused by routing protocols avoiding sending packets over links that are currently experiencing high network loads.

Another conclusion to be made is that the observed performance of the system was within the range specified to be satisfactory that we defined in our hypothesis, 0ms to 500ms. The trial with the largest mean Total Delay, London, UK, was still less than 400ms; and the trial with the smallest mean, Fredericton, NB, was less than 150ms. Generally in enterprise web applications, the application is deployed on multiple servers so users would generally not have to send packets as far as from London to Fredericton.

6.7 Experiment 2: System Latency and Server Stress Testing

6.7.1 Purpose

The purpose of this experiment is to see how the application sharing server holds up to a heavy load, and preferably find the optimum amount of users that can be supported by a single machine (of my server’s specifications) before we need to perform load balancing or other techniques to distribute
the users over multiple servers.

6.7.2 Background

This implementation runs on a single server to provide the intended functionality. There are however, architectures that allow better performance when a single server isn’t able to keep up with the demand of users. The most common techniques are Load Balancing, and Clustering.

The main difference between Load Balancing and Clustering is that a Cluster is defined to be a set of homogeneous machines, each with the same hardware and software. It is also usually controlled by a single master that distributes work to the slaves. In contrast, the technique of Load Balancing allows the use of a set of heterogeneous machines, and usually has multiple machines that can act as masters reducing the possibility for the system to go down in the occurrence that one machine fails [21]. In both solutions the master receives requests and then delegates them to the slave machines to perform the request based on an algorithm to determine which machine has the lowest load.

6.7.3 Hypothesis

I expect to see a linear increase of latency with respect to the number of concurrent users until we reach a tipping point where another bottleneck
will take over; increasing the rate of latency experienced by the users.

6.7.4 Experiment

To accomplish this experiment I will modify the application requester’s client as well as the application provider’s client to produce a number of normally operating requester/client combinations each on their own threads. Once I have the number of threads I need I will start them all executing and watch the task manager to make sure that having too many clients running from the same computer isn’t creating a bottleneck that would skew the results.

To record the latency experienced I will have each requester send a command and measure the elapsed time it takes for the response to return back. Since all threads will start at the same time we should see a consistent level of latency for each requester/provider pair of the experiment. As each trial completes we will choose a new number of users to simulate by creating that many requester and provider threads. We will keep the network topology the same to eliminate the network delay as much as possible from factoring into the results.
Figure 6.9: Latency vs. number of users, we can see that there is a linear relationship between latency and the number of users accessing the system concurrently.
6.7.5 Analysis

As we can see in Figure 6.9 there is a linear relationship between the number of concurrent users accessing the system and the response time to make a round trip between requester and provider.

1000 concurrent simulated users was the largest trial I could successfully complete. As I increased the number of users over 1000 I experienced socket communication timeouts on the client end. This means the server was so busy it was not responding to requests on the sockets within the time limit, causing the socket to close.

If we take two points that fall on the linear approximation of the data such as 10 and 700 we can calculate the slope of the line, and thus the approximation of how latency increases with respect to users being added.
Point 1 : (10, 4.76) \hspace{1cm} (6.4)

Point 2 : (700, 219) \hspace{1cm} (6.5)

\[
\frac{\Delta Y}{\Delta X} = \frac{219ms - 4.76ms}{700users - 10users} = \frac{214.24ms}{690users} = 0.31049ms/user \hspace{1cm} (6.6)
\]

This relationship is derived from our observations and we cannot assume it holds true for any amount of users as different bottlenecks in the system will eventually be reached causing the rate of performance per user to change. However, while this linear approximation cannot reliably be used for extrapolating our data, we can use it to fairly confidently interpolate our data keeping in mind the error bounds shown by the standard deviation bars in the graph.

6.7.6 Conclusion

As we saw socket timeouts when attempting 1100 concurrent users and did not see any when executing 1000 users we can say that the system is capable of serving 1000 users before needing to scale.

As we hypothesized the relationship between additional concurrent users and system latency is linear. We were able to calculate that relationship to be 0.31049 ms/user, meaning that for each concurrent user added to the system all communication will be slowed down on average by 0.31049 millisecond.
onds. The reason for this has to do with additional server load from having more threads running to handle the additional interactions. This means each thread has more of a delay between when input is received by the modem, and when the thread gets time on the CPU to retrieve the input and send output. Although our experimental results had a few spikes the results were in general very close to the linear approximation described in the analysis section. Some possibilities for the data points that did not conform to the approximation could include network latency, or other processes on the server requiring higher amount of resources.
Chapter 7

Conclusions

7.1 Conclusion

In conclusion, this thesis introduces the concept of Social Distributed Computing. This can simply be defined as a new set of functionalities to be added to Online Social Networks. These functionalities include Application Sharing and Resource Sharing, and are made possible by modules which allow people to connect through the Social Network to find Applications and Resources, setup which Application and Resource they would like to share themselves, as well as when and how much. This service allows us to take advantage of our Online Social Networks to provide additional application diversity via Application Sharing, and to provide an increased pool of computational resources to draw on when something needs to be done quickly via Resource Sharing.
While Resource Sharing may not be as useful of a technology at this point, since most software is designed to operate on a single machine and is not able to take advantage of grid computing, in the future we could see it become useful in one of two ways. The first being the creation of more applications using distributed computing that could be deployed over a grid, and the second being the creation of a middleware solution that would be able to run an application designed for a single machine over a grid of computers.

Application Sharing on the other hand is something that we have no technological challenges to figure out before making a reality today. If we perfect the mechanism to allow users to share applications with minimal effort we will see an increase of application diversity for all users, meaning more incentive to use the system.

When we consider the functionality of Online Social Networks, we haven’t seen any dramatically new functionality since their conception around 2004: messaging, event planning, media sharing, these have been the staples of Online Social Networks since their beginning. Some functionality that could be considered newer than those original pieces could be online social gaming such as FarmVille where the objective of the game is to build a better universe than your friends. These types of games rely heavily on social incentives to make the game enjoyable. For instance, consider the amount of times in
one session the game will suggest you share some insignificant news about an accomplishment in the game with your friends. Hopefully this thesis will start people thinking of unconventional functionality that can take advantage of the pre-existing social connections inherent in Online Social Networks.

7.2 Contributions

The contributions made by this thesis include the proposition of the theoretical Social Distributed Computing concept and models, the theory realized as the designed Social Distributed Computing architecture and services, and the implementation of that design in the form of a set of prototype Social Distributed Computing services running on an open source Online Social Network that can be expanded as desired by future researchers.

The proposed Social Distributed Computing concept and models act as a foundation on which we can design applications that would use SDC. The introduction of them in this thesis allows future researchers to consider the concept of sharing computational resources through social networks, and lead them to consider other kinds of services could be shared over Online Social Networks that have not been listed in this thesis.

The architecture and services designed in this thesis provide a concrete example of applications that could be created using the concept of Social Dis-
tributed Computing. These examples can be expanded on to improve the functionality of the services in several ways by future researchers.

The prototype implemented provides a proof-of-concept and allows readers to experience the functionality in the real world. Since it is implemented into an Open Source Online Social Network people can take the source of my implementation and improve upon it as much as they want.

7.3 Future Work

The most obvious first step that could be taken to further research in this area would be to implement the parallel social computing module. The current implementation of the system omitted doing that. One possible idea for this would be to use a programming language that runs in a runtime environment, then a parallel application could be deployed by simply sending a class along with the data and have that class execute on the remote machine.

If the researcher is more interested in furthering the control module, that is to say how the users interact within the social network in regards to application sharing and resource sharing, they could investigate other workflows to organize the application sharing such as having each user create a list of applications they are willing to share. One could also look into how it could
be automated so that any of their friends could access those applications without interacting with the user. This would make the system more convenient and usable.

For the researcher interested in the Application Sharing module they could look into better implementations for screen sharing. The current implementation uses a program that takes control of the provider’s keyboard and mouse. It should be possible to run the application and simulate the mouse and keyboard clicks without preventing the provider using the computer themselves.

Another interesting area of research would be to introduce collaborative applications into the Social Distributed Computing framework. This could be done in two ways, the first of which by taking applications already created to be collaborative and setting them up to use the framework so multiple friends can work together. The second would be by adding functionality to the SDC framework itself that would allow for application built to be single user could be used as collaborative applications [23]. There is already research out there to turn single user applications into collaborative applications. There are also a few business enterprises such as GoInstant based in Halifax, NS, that offer a collaborative web browsing solution.

Another possibility to further this research would be to look into other interfaces than video/mouse/keyboard. As mentioned earlier in the thesis, Social
Distributed Computing is about sharing applications and resource that use any interface. Currently there exist applications using such alternate interfaces as audio interfaces, gesture based interfaces (body movement as well as eye movement), and touch based interfaces. These non-conventional interfaces have less research as to the best way to share an application that uses them through a network, and could perceivably be used more in the future.

An API could be developed to facilitate different streams of data through the Social Distributed Computing service. For instance the current implementation is socket based for both GUI and command line application sharing and the clients interface with those sockets. By creating an API describing these interfaces it would leave the system open for developers to create their own clients. Moreover the current interface is GUI applications access here, command line access here. It would be more generic if an application could mix and match between interfaces that could include video, audio, keyboard, mouse and command line input and output stream. Then have the web based requester’s client include those different streams as needed, for example they could have an audio stream and a section for command line access. While this situation may not come up very frequently, as new interfaces are added such as gesture it would allow for more useful mix and matches.

Another area that research could be done into would be to create a middleware solution that would allow mobile applications to utilize the social grid
computing possibility that arises from resource sharing to push computation-
ally difficult tasks onto machines that have a permanent power supply.
Bibliography


[28] Bogdan Popescu, Bruno Crispo, and Andrew Tanenbaum, *Safe and private data sharing with turtle: Friends team-up and beat the system.*


Vita

Candidate’s full name: Daniel Latimer

University attended (with dates and degrees obtained):
September 2011 - May 2013
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada

Bachelor of Computer Science
September 2005 - May 2008
University of New Brunswick
Fredericton, New Brunswick, Canada

September 2004 - May 2005
Faculty of Science
Cape Breton University
Sydney, Nova Scotia, Canada

Poster:
Daniel Latimer, “Poster: Social Distributed Computing,” Proc. of the 10th UNB Computer Science Research Exposition, April 11, 2013, Fredericton, NB, Canada

BCS Technical Report:
Daniel Latimer and John DeDourek, “Video Compression and Encoding,” UNB Faculty of Computer Science, April, 2008, Fredericton, NB, Canada