INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Redes Sociais para Cycle-Sharing

Social Networks for Cycle-Sharing

Nuno Miguel Silvestre Apolónia

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Alberto Manuel Rodrigues da Silva
Orientador:	Prof. Luís Manuel Antunes Veiga
Co-orientador:	Prof. Paulo Jorge Pires Ferreira
Vogal:	Prof. Bruno Emanuel da Graça Martins

Lisboa, Outubro 2010

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Luís Veiga, whose encouragement, guidance and support enabled me to develop an understanding of the subject and the help to make this dissertation possible. Also, I would like to thank my co-supervisor Prof. Paulo Ferreira.

I would like to show my appreciation to my colleagues Pedro Oliveira, for his support throughout this work, and Raul Silva, with his expertise in Latex and his inspirational words.

And also, I would like to thank my colleagues at INESC-ID and IST, for additional motivation and incentives.

Last but not least, my deepest gratitude goes to my family and friends, especially to my mother, Maria Albertina Silvestre, and to my brother, Luís Apolónia, without doubt, have always supported me in my decisions and to pursue my dreams.

Abstract

The growth of the Internet, and consequently the number of interconnected computers is the basis for global distributed computing and public-resource sharing. Meaning that, these resources have been used for computation intensive projects that could not be completed in a short time frame or that the use of supercomputers would not be significant to complete them in time. These kind of projects use idle cycles from willing user's computers to complete their research on specific topics, such as SETI@Home [ACK$^+$02].

Furthermore, the Internet has been overwhelmed by social connectivity. Internet users have been using Social networks to interact and share information, knowledge and services with each other.

This work presents an overview of Peer-to-Peer networks and Grids, to understand their advantages and problems. So that, we can grasp the fundamental ideas that sprout the global distributed computing and the problem of locating resources and services efficiently.

We also analyze Social networks and social interactions to understand how they can be explored for other uses rather than what they were initially created for.

In the last chapters we explain the development and resulting evaluation of a Web-enabled platform, called Social Networks for Cycle-Sharing (SNCS), that uses Social networks as a starting point for resource and service discovery and integrating it with Ginger Middleware for distributed computing of Jobs.

Also, to conclude that using a Social network for public-resource sharing can give common users the possibility of releasing resources for other applications usage.

Keywords

Social networks, resource discovery, distributed systems, Peer-to-Peer, Grids, public-resource sharing, global distributed computing.

Resumo

O crescimento da Internet e consequentemente do número de computadores interligados é a base para a computação global distribuída e da partilha de recursos públicos. Neste sentido, estes recursos são utilizados para projectos computacionalmente intensivos que poderiam não ser concluídos num curto espaço de tempo ou que o uso de super computadores não seria significativo para os completar a tempo. Estes tipos de projectos utilizam ciclos inactivos de computadores de utilizadores dispostos a completar a investigação sobre temas específicos, tais como o SETI@Home[ACK$^+$02].

Além disso, a Internet tem sido inundada por conectividade social. Ou seja, utilizadores da Internet têm vindo a usar as Redes Sociais para interagir e partilhar informação, conhecimento e serviços entre pessoas.

Este trabalho apresenta uma visão geral das redes Peer-to-Peer e Grids, para compreender as vantagens e problemas destas. O que nos poderá levar a compreender as ideias fundamentais que originaram a computação global distribuída e os problemas de localização de recursos e serviços de forma eficiente.

Iremos também analisar as Redes Sociais e as interacções sociais para apreender a forma de como se podem explorar para outros fins, sem serem aqueles para que estas foram inicialmente criadas.

Nos últimos capítulos explicamos o desenvolvimento e os resultados da avaliação de uma plataforma *Web-enabled*, denominada de *Redes Sociais para Cycle-Sharing (SNCS)*, que utiliza as Redes Sociais como ponto de partida para a descoberta de recursos e serviços, como também a integração com o Ginger Middleware para a computação distribuída de Trabalhos.

Concluímos também que utilizando uma Rede Social para a partilha de recursos públicos pode trazer aos utilizadores comuns a possibilidade de libertar recursos para a utilização em outras aplicações.

Palavras Chave

Redes Sociais, descoberta de recursos, sistemas distribuidos, Peer-to-Peer, Grids, partilha de recursos-publicos, computao global distribuida.

Contents

List of Figures

List of Tables

List of Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BOINC	Berkeley Open Infrastructure for Network Computing
CCOF	Cluster Computing on the Fly
CPU	Central Processing Unit
DHT	Distributed Hash Table
FoFs	Friends of Friends
GIMPS	Great Internet Mersenne Prime Search
Ginger	Grid Infrastructure for Non-Grid Environments
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JXTA	Juxtapose
OWL	Web Ontology Language
P2P	Peer-to-Peer
PHP	Hypertext Preprocessor
REST	Representational State Transfer
SNCS	Social Networks for Cycle-Sharing
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

Contents

The computing power has been significantly increased in the past few years (more or less like Moore's Law [Sch97]), but there are still many computation problems that need an enormous amount of computing resources, e.g. applications for scientific research, financial risk analysis or multimedia video or image rendering and encoding. These resources are composed by computing elements like CPU, memory or data storage, and all of them can be found in the millions of desktop computers all around the World. Meaning that they can be found on every house hold or in offices and even in our daily devices, such as notebooks or mobile phones.

The idea to use idle cycles for distributed computing was proposed in 1978 by WORM computation project at Xerox PARC [Sho98]. It was only after, that the scientific community started to see the benefits that such systems can give. Furthermore, the possibility of having supercomputers for their disposal was very tempting and made the scientific community realize that they could harvest the idle processing time to suite their own needs. These networks are called Grids [FKNT02], a combination of computational resources from multiple administration domains, and also defined as a coordinated resource-sharing and problem solving environments, that made possible to make distributed processing of large computation (and scientific) problems.

With the computer global network (Internet), the available resources for those kinds of projects were extended. Such projects like SETI@Home [ACK$^+$02], Folding@Home[1], Distributed.net[2] gathered the gigantic potential of using desktop computers from any house hold (also known as global distributed computing), allowing them to process their data much quicker than in traditional supercomputers. This is usually done by, Internet users willing to participate in such projects, that *install* an application, which runs in the background when the computer has idle cycles to spare.

A lesson to be taken from such projects is that to attract and keep users, such projects should explain and justify their goals, research subject and impacts. Users may not be interested in systems that would steal their idle cycles without their consent.[3]

The Internet has also enabled information and content sharing by using Peer-to-Peer networks. These networks are usually formed by interconnected home desktop computers. Also, they can be categorized in terms of their formation as being structured or unstructured. Unstructured systems are characterized by having a underlying topology unrelated with the placement of the contents, as opposed to Structured systems where it is attempted to place the contents in specific locations. Furthermore, there have been done optimizations to leverage the performance for locating contents and their scalability (in terms of traffic load). They are generally called Hybrid systems that highlight two types of users. The users that have more bandwidth are called super-peers and those with low bandwidths are called peers and the last ones are connected to the super-peers [TTP$^+$07].

These networks have some challenges, such as efficient resource discovery. That is, when a peer needs a resource it asks other peers for it. Some approaches try to minimize the message traffic that can be

[1] Folding@Home Web site: `http://folding.stanford.edu` accessed on 05/01/2010

[2] Distributed.net Web site: `http://www.distributed.net` accessed on 05/01/2010

[3] Plura Processing response to the Digsby Controversy: `http://pluraprocessing.wordpress.com/2009/08/24/our-response-to-the-digsby-controversy-new-terms-of-use-affiliate-auditing` accessed on 15/10/2010

generated, either by contacting fewer peers (when information is spread to others), or by creating central nodes that have all or partial information for locating the exact content.

Moreover, the Internet has made it possible to exchange information more rapidly in a global scale. One of the natural steps was the creation of Social networks, where any one in the world can share their experiences and information using only their Internet enabled personal computer or mobile device.[4] Under this scope there are many Social networks such as Facebook[5], Orkut[6] or Youtube[7] each one exporting their own *APIs* to interact with their users and groups data bases, e.g. Facebook API[8] and OpenSocial[9]. These networks have great potential for financial benefits, such as Advertising.

Furthermore, studies done on them show some properties like the Small-World property, meaning that there is a small group of users with high connectivity to others and a much larger group with low connectivity. Besides that, even the highly connected users only interact (on a daily basis) with a restrict group of users [WBS$^+$09]. Considering that these networks could be regarded as enabling Peer-to-Peer information sharing (albeit mediated by a centrally controlled infrastructure), employing them for cycle-sharing should be a great improvement for global distributed computing, by allowing public-resource sharing among trusted users and within communities.

There are already projects that use Social network concepts to improve performance on other topics, such as PeerSpective [MGD06] which enhances search results with social information from friends and Social VPN [FBJW08] that allows the creation of virtual private networks on top of the Social networks, to be used much like a local network.

The Ginger project [VRF07] serves as a middleware for creating distributed processing using a Peer-to-Peer network (P2P) for work dissemination (*Gridlets*) within its peers. Also, to do task distribution it is necessary to find available resources on the network and who can spare their idle cycles, much like other global distributed computing projects. The main idea behind this project is that any user may need processing time for common applications to be executed. Therefore, a P2P network is formed between the users, to locate idle cycles and resources to be used for those tasks.

Our project developed and evaluated a Web-enabled platform, called Social Networks for Cycle-Sharing (SNCS), that interacts with a Social network (Facebook) to be able to locate and search for idle resources among its users. Also, it makes use of the Ginger Middleware for *Gridlet* creation and aggregation.

SNCS uses a Social network already established in order to give beneficial results to communities willing to adopt the paradigm of cycle-sharing. Moreover, the users in such networks are mostly linked with each other by friendship and common interests, meaning that users may be more opened to share their resources with their own friends.

The client application developed interacts with Facebook mostly by means of the *Graph* interface they

[4]Facebook Mobile: `http://www.facebook.com/mobile` accessed on 19/08/2010
[5]Facebook Website: `http://www.facebook.com` accessed on 05/01/2010
[6]Orkut Web site: `http://www.orkut.com` accessed on 05/01/2010
[7]Youtube Web site: `http://www.youtube.com` accessed on 05/01/2010
[8]Facebook Developers: `http://developers.facebook.com` accessed on 05/01/2010
[9]OpenSocial Web site: `http://code.google.com/apis/opensocial` accessed on 05/01/2010

provide. Which gives us access to users' information, such as their friends, groups and Wall. These Walls are the main interactions between the people that uses Facebook, meaning that they record messages onto the Wall to be read by the users linked to them.

SNCS is then able to discover resources for the execution of Jobs (which are composed by *Gridlets*) submitted by the users. The Jobs are intensive computing tasks that can be partitioned in several smaller tasks which are called *Gridlets*, these contain the data files and arguments to be executed on the processing SNCS client.

The client application is composed of modules that interact with the Social network, with the users' computers and the Ginger Middleware. Furthermore, the interactions between SNCS and Facebook are helped by the use of the *RestFB* library, which gives us a simple way of using *Graph* and *REST* functionalities. And also, the use of SIGAR library to gather computers' information in order for SNCS to donate idle cycles and know which resources are provided by the Social networks' users. Moreover, the resource and application discovery enables us to distribute *Gridlets* among users and to aggregate them using the Ginger Middleware.

Furthermore, a communication protocol has to be established in order to interact between the SNCS clients and complete Jobs successfully. This means that we use Facebook Walls (users/groups/Application Walls) to send and retrieve messages sent by other SNCS clients. SNCS starts this process by sending a Job search message to the users' Wall in order for the users' friends to read and accept (or deny) the request to use their idle resources.

As we also want to gather users' computers as much as possible, we send messages to groups that have users that may be willing to help (or have the Jobs' requirements).

Furthermore, we extend the reach of gathering resources to contain friends of friends (FoFs), meaning that SNCS contacts FoFs SNCS clients for resource sharing.

However, this interaction uses the users' friends as the carriers of the messages to the FoFs, meaning that the friends redirect messages to their friends in order for them to forward the response on the Applications' Wall. This happens because Facebook does not allow users to interact with each other without being linked as friends. The use of the Applications' Wall subverts this inability.

In the development of SNCS, our concerns were with the resource discovery and also the manner which a user could submit his own Jobs to be processed on others. Moreover, to reach as many users and communities as possible we used Java for portability purposes.

A fact is that SNCS suffers from constraints due to the use of Facebook as the means of interactions between users. Meaning that the Social Network restricts outside applications with rules (Use Terms)[10] and restraints.

Also, Facebook is still developing some *Graph* functionalities and thus they may not work as expected, therefore we use the discontinued *REST* interface to access them, such as reading and writing of Comments in groups' Walls.

[10]Facebook Use Terms: `http://www.facebook.com/terms.php` access on 26/08/2010

The evaluation of SNCS is comprised of several scenarios, where each of them evaluates a portion of our works' goals, in order to know the effects each carries.

We describe 8 scenarios, starting with one that has two friends communicating with each other in order for us to know if SNCS can achieve resource and application discovery.

In the last scenarios we augment the network size used, to become more realistic in terms of users' roles (friends, FoFs, group members). And also we change the *Gridlets* to become more realistic, by using a known program (Pov-Ray)[11] to render an image.

These scenarios were made, in order for us to conclude that SNCS can gain speedups against local execution, however it was demonstrated that SNCS can be hindered by variables such as Facebook latency, and searching for resources may not return positive results, or the number of *Gridlets* surpasses the number of donating users.

We finally state that we achieved our works' goals, meaning that it is viable the use of a Social network to gather idle resources scattered among Social network users, to be used by common users applications' execution that would take more time in their computers, releasing resources for other tasks.

1.1 Current Shortcomings

The public-resource sharing and cycle-sharing systems that are widely used today, do not concern with the common users' needs. They are mostly used for intensive computation projects (and proprietary) such as Folding@Home, PluraProcessing.

Other systems allow common applications to be executed, however they do not support users' networks already established, meaning that they cannot use Social networks to be able to gather resources to be used by other interested users (such as friends or communities), while often do not provide a total sharing system between users.

Some systems however are beginning to use technologies previously unavailable to other projects, in order to cover more Internet users. Such systems can use the users' Browsers to do cycle-stealing instead of addressing the needs of the common users. Moreover, they use remote code embedded on Web sites and games (i.e. Adobe Flash[12] based games) to gain access to potential idle resources.

Furthermore, their resource discovery and scheduling are rudimentary, meaning that they do not rely on established networks of users to do public-resource sharing, instead users may have to create their own networks. Also, their scheduling process is ascertained by determined users or by occasional users (which may not be aware of the system).

1.2 Objectives and Contributions

This work follows the Ginger project and is aimed at resource and application discovery through users from Social networks, such as Facebook, to execute Jobs (which are comprised of *Gridlets*) remotely.

[11]Pov-Ray: `http://www.povray.org` accessed on 13/10/2010
[12]Adobe Flash: `http://www.adobe.com/products/flashplayer` accessed on 13/10/2010

The fundamental idea of Social Networks for Cycle-Sharing (SNCS) is to interact with a Social network, and make it easier to find resources and other applications to be used for completion of Jobs (that include execution of users' programs) that each user can submit to be processed remotely. While also, scheduling the Jobs on friends or communities that are willing to share their resources.

The objectives of this dissertation can be summarized as:

1. Analyze current P2P and Grid systems, to know how resources are found in such networks for the distributed tasks;

2. Analyze Social networks and social interactions to know how this work can leverage them to suit the need of finding resources;

3. Study how global distributed computing and public-resource sharing (such as that using BOINC) are performed, in order to have a proven starting point approach as the basis for our work;

4. Understand how to integrate distributed computing with Social networks, to improve the performance of resource discovery;

5. Develop a Web-enabled platform that can interact with the Social network and the overlay network for the distributed tasks;

6. Evaluate the work in a simulated environment, to prove that we can use a Social network for resource and application discovery among its users and use it for distributed computing;

The objectives 1, 2, 3 and 4 are addressed in the Related Work. This gives us a background on how distributed computing and resource discovery have been done and how this work can be accomplished using that knowledge. Also, it is necessary to analyze Social networks to know if it is feasible to use them for another goal, other than communication within friends.

Objective 5 gives a proof of concept application, to make it possible to take advantage of Social networks to be used as a way for resource discovery. In addition, SNCS has the following requirements:

- Finding friends and their profiles (from user's computers and components);

- Finding groups and communities that the user has joined;

- Gather information about users' state (e.g. Online, Away);

- Locate resources among the users, that are willing to participate in the Jobs;

- Schedule Jobs, when users are connected to the network, but not available to execute them;

- Integrate seamlessly into the everyday life of the Social network usage, without compromising user ability to communicate with his/her friends.

These final objectives are intended to focus this project on a path for improving the resource and service discovery on top of a Social network already established.

Furthermore, SNCS needs to be able to gather idle cycles from users' computers and communities that would be willing, and capable of doing a Job, in order to achieve cycle-sharing on a Social network.

Also, it could give the chance for common users to use the cycle-sharing paradigm to speedup their own (or common) applications' execution without needing to create a new network. Meaning that, SNCS could use an already established network as in case of Social networks (Facebook, MySpace, among others) to interact between users to share their idle cycles.

1.3 Document Roadmap

The rest of this dissertation is organized as follows.

The next Chapter, the Related Work, begins by presenting a analysis on Peer-to-Peer networks and Grids (Sections 2.1 and 2.2) and Social interactions (in Section 2.3), ending with mechanisms and code execution via Web (in Section 2.4) and a summary to conclude the related work (in Section 2.5).

Then on Chapter 3, we describe Social Networks for Cycle-Sharing (SNCS), that follows our works' goals, by presenting the aspects that compose the SNCS architecture (in Section 3.2), its communication flow (in Section 3.3) and ending with a prototypical example of SNCS use (in Section 3.6).

On Chapter 4, we give the details of the implementation of SNCS, also the technologies involved (in Section 4.1) and some of the constraints SNCS suffers by using Facebook as its Social network (in Section 4.6).

Chapter 5 describes the scenarios used to evaluate SNCS and its results, giving an overview of how it behaves on such networks, either with a more realist environment (in Section 5.5) or using a actual application to process *Gridlets* (in Section 5.6). And on Section 5.7 we discuss the obtained results to understand SNCS properties.

On the last Chapter we present the conclusions and future work of this dissertation.

2

Related Work

Contents

This chapter gives a review on network topologies such as P2P and Grid systems, to understand the related problems like efficient resource discovery. Also, it addresses some projects that have been done in the area of distributed computing, to give relevant aspects regarding our works' goals.

Moreover, it gives a review about Social networks and user interactions, addressing some of the studies done in these networks. To understand how we can achieve resource (and service) discovery using Social networks and why it will be important for global distributed computing or public-resource sharing. We finish the Related Work with some practical applications that utilizes Social networks for other purposes rather than the intended idea of social interactions.

2.1 Peer-to-Peer networks and Grids

The Peer-to-Peer (P2P) networks and Grids are the most common types of sharing systems, they evolved from different communities to serve different purposes as [TTP$^+$07] assumes.

The Grid systems interconnect clusters of supercomputers and storage systems. Normally they are centralized and hierarquically administrated, each with its own set of rules for the resources availability in order to participate in distributed computing. They can be dynamic and may vary in time but with smaller magnitude and have to be known among the network.

The Grids were created by the scientific community to run computation intensive applications that would take too much time in normal desktops (without being distributed) or on a single cluster, e.g. large scale simulations or data analysis.

The P2P networks are typically made from house hold desktop computers or common mobile devices, being extremely dynamic in terms of resource types and whose membership can also vary in time with more intensity than with Grids. These networks are normally used for sharing files, although there are a number of projects using those kinds of networks for other purposes, such as sharing information and streaming (e.g. Massive Multiplayer Online games using P2P [KLXH04] to alleviate server load, distributing tasks as SETI@Home [ACK$^+$02], data streaming for watching TV[1]). The nodes (or peers) are composed by anonymous or unknown users unlike in Grids, which may have its own problems with security or even with forged results [TTP$^+$07].

Both these systems have been converging by relaxing rules from Grid systems and opening P2P applications for more computational methods.

These two distributing systems have different resources, which may indicate a different level of computing power of the nodes comprising each one. However, it is easier to leverage more desktop computers than to have large supercomputers at our disposal. This can make P2P systems aggregate more computing power than the Grid systems.

[1]PPStream: `http://www.ppstream.com` accessed on 05/01/2010

2.1.1 P2P systems

The P2P systems can be categorized in terms of their structure in three ways (also in Table 2.1.1), as follows.

Examples of Peer-to-Peer File Exchanging Systems	
Unstructured Systems	Napster
	Gnutella
	Freenet
	Kazaa
Structured Infrastructures	CHORD
	CAN
	PASTRY
	Tapestry
Hybrid Systems	Kademlia
	Kazaa
	Gnutella2
	eDonkey

Table 2.1: Examples of P2P Systems and Infrastructures

Unstructured systems: where the placement of contents (Files) is completely unrelated to the overlay topology and they must be located (or searched). These systems, such as Gnutella[2] and (FastTrack) KaZaA [LKR04], are generally more appropriate for accommodating highly-transient node populations. To search for resources it is common to use methods such as *Flooding* [PFA+05], *Dynamic Querying*[3], *Random walks* [TR03], *direct searches* [LCC+02] (if statistical information is available) or *forwarding indices* [CGM02]. Moreover, they have obvious implications regarding availability, scalability and persistence [ATS04].

A optimization (also considered as an hybrid approach) to these systems follows the line of having two types of peers (or nodes) the super-peers (peers with higher bandwidth) which would form a unstructured overlay network and the leaves (peers with low bandwidth) connected to the super-peers. So that flooding of messages are only passed through the super-peers and not cause problems with peers which cannot handle to many search requests [TTP+07].

Structured systems: these systems are an attempt to improve the scalability issues regarding locating content, that the unstructured systems suffered from, by controlling where contents should be placed at all times. For supporting searches these systems use namely a distributed routing table (also presented as DHT - Distributed Hash Table) [Man03], in order for queries to be efficiently routed to the peer that has the content or the information where the content is located.

Every peer that joins the network has partial information where to find the contents (CHORD [SMK+01], CAN [RFH+01], Pastry [RD01]) meaning that peers need more information to join the network. These systems are more scalable in terms of traffic load, but still need more auto-organizational capabilities.

[2]Gnutella Protocol: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html accessed on 07/01/2010

[3]Dynamic Querying Protocol: http://www.ic.unicamp.br/~celio/peer2peer/gnutella-related/gnutella-dynamic-protocol.htm accessed on 05/01/2010

Hybrid approach: was created to make up for the lacks that each approach have and still retain their benefits. Like Kademlia [MM02] uses a group of peers (that are near each other) known as buckets to locate files, resolving some flooding problems. Also peers may have the ability to change themselves, enabling them to become part of the overlay network used to coordinate the P2P structure. Furthermore, some hybrid systems use a central server to bootstrap the peers (i.e. eDonkey network [OBBO04]).

2.1.2 SETI@Home System

There has already been work around in the area of global distributed computing (the use of home and office computers for distributed computing), as we can see in projects like SETI@Home [ACK+02]. Where they use these kind of resources to analyze radio wave signals that come from outer space, hoping to find radio signals originated from other planets on our galaxy. Thus, proving that there are extraterrestrial life advanced enough to send such kind of waves.

The data is processed in three phases: *"computing its time-varying power spectra;" "Calculating "candidate" signals through pattern recognition on the power spectra;" "Eliminating candidate signals that are probably natural or man-made;"*, meaning that each individual computer does these phases in an attempt to find relevant signals and as their only processing job.

For this project, having more computing power means they could cover a greater range of frequencies, instead of using supercomputers which they did not have in abundance [ACK+02], they found a way that lets them use computers around the world to calculate those wave signals.

The wave signals were divided in small units of fixed size to be able to distribute among the BOINC clients (that would be located in any user computer), then the client would compute the results in their spare time and send it to the central server asking for more work to do. In this process the clients would only need to be able to communicate with the server when they finished the computations (or for asking more data).

The users do not need to be aware of such complexity, they would only *install* a *screensaver* in their computers and this application would activate itself only when the computer had idle cycles to spare (and not being used).

The client (application) was designed to have a platform independent framework for distributed computing, in order for them to reach as many Internet users as possible. Moreover, users had a ranking system to compete against other users, to motivate them to use this system. Thus, adding that the most important lesson of SETI@Home project was that to attract and keep users, such projects should explain and justify their goals, research subject and its impact.

2.1.3 Berkeley Open Infrastructure for Network Computing (BOINC)

BOINC [And04] is a platform for distributed computing through volunteer computers, it emerged from the SETI@Home project and became useful to other projects.[4]

Folding@Home [LSS+09] is an example of a BOINC system that studies protein folding, meaning

[4]BOINC projects: `http://boinc.berkeley.edu/projects.php` accessed on 13/10/2010

when proteins assemble (or fold) themselves for a certain task or function; misfolding, which occurs when proteins do not fold correctly; and related diseases such as Alzheimer's, ALS, Huntington's, Parkinson's disease, and many types of Cancers.[5] The system uses distributed computing to simulate time scales, thousands to millions of times longer than previously achieved, which allows them to simulate actual protein folding and direct their approach to examine folding related diseases.

Another example of a BOINC system would be the climateprediction.net [SKM$^+$02], that employs climate models to predict the Earth's climate up to 2100 and to test the accuracy of such models. This allows them to improve their understanding of how sensitive their models are to small changes and also changes like in carbon dioxide and sulphur cycles. The project has many similarities with other BOINC systems, but the computational tasks are different.

Although each project has its' own topic and therefore their own computational differences, the BOINC system used for each project (client application) has to be unique.

2.1.4 Distributed Computing Projects

The subject of distributed computing has been previously addressed by several projects. And the first relevant projects to distributed computing were distributed.net[6] and GIMPS.[7]

Distributed.net uses computers from all around the world to do brute-force decryption of RSA keys, and attempt to solve other large scale problems. Their initial project was to break the RC5-56bits algorithm, which took 250 days to locate the key (0x532B744CC20999). Other consequential projects like RC5-64bits, Optimal Golomb Rulers (OGR-24, OGR-25, OGR-26), which is a mathematical term given to a set of whole numbers where no two pairs of numbers have the same difference, have also been concluded with varying times of 100 to 3000 days, and currently they are trying to break the RC5-72bits algorithm and find the OGR-27.

The GIMPS project uses the same concept of distributed computing to search for *Mersenne* prime numbers, these numbers are of the form $2^P - 1$ where P is a prime. The last known Mersenne prime (47th) that was found was $2^{42643801} - 1$, which has about 12.8 million digits.

Both projects use its own Client and Server applications, following the same idea as the BOINC projects. Also, there are many other projects for distributed computing.[8] However, all of them have only one topic of research (for each project), meaning that each system does not have the flexibility of changing its own method of research.

2.1.5 XtremWeb

The fundamental idea of global distributed computing is to harvest the idle cycles of computers from all around the world that use the Internet. The XtremWeb [FGNC01] project aims at the development of

[5]Folding@Home Web site: `http://folding.stanford.edu` accessed on 05/01/2010
[6]Distributed.Net: `http://www.distributed.net` accessed on 05/01/2010
[7]GIMPS: The Great Internet Mersenne Prime Search `http://www.mersenne.org` accessed on 05/01/2010
[8]List of Distributed Computing Projects: `http://en.wikipedia.org/wiki/List_of_distributed_computing_projects` accessed on 05/01/2010

a platform to experiment the capabilities of this concept. By gathering the unused resources of Desktop computers (such as CPU, storage, network) to build a lightweight Desktop Grid where their primary goal is to turn a set of volatile resources spread over a network into a runtime environment to execute highly parallel applications.

Some of the issues that they had in consideration were: *"sizing the environment components (servers, network, workers) according to the applications' features; high performance and secure execution (relies on sandboxing); modeling resource and workload management as inputs for scheduling algorithms; and the impact of the application characteristics, either compute- or data-intensive."*

Also, some of the related issues with global distributed computing were addressed, such as:

- *Scalability* of the network;
- *Heterogeneity* across hardware and software;
- *Availability* of the users' resources;
- *Fault tolerance* within its nodes;
- *Security* maintaining participating users protected against malicious or erroneous manipulations;
- *Dynamicity* in terms of configuration as well as communication latency and throughput;
- *Usability*, easy deployment and usage.

These issues may impact the reliability of the project to be able to compute the problems. Their approach was similar of the BOINC systems, by aggregating the nodes from the network (Internet/LAN) and share their resources within this network. And also, they add the possibility for users to register their own global distributing application, which is a step forward of only allowing one research topic to be executed to having any application that can be ran (a resource-sharing environment).

2.1.6 BOINC Extensions for Community Cycle Sharing (nuBOINC)

Users without expertise may encounter difficulties in setting up the required infrastructures for BOINC systems and subsequently gather enough computer cycles for its' own project. The nuBOINC extension [SVF08] is a customization of the BOINC system, that allows users to create and submit tasks for distributed computing using available commodity applications. In sense, they try to bring global distributed computing to home users, using a public-resource sharing approach.

In their architecture, the client was modified in order for the user to submit jobs to the server, registering the application (e.g. POV-Ray) that should be invoked to run the job. Also, defining what arguments the application should use when it is executed. These applications are integrated with the help of the Registrar application on the server side. Which contains the applications that are available for the jobs that are submitted.

On the client side a component (comBOINC) is responsible for invoking the commodity applications which are needed to process the input files.

Furthermore, by allowing users to create their own projects, on top of commodity applications, they do not have to develop a full fledged BOINC application from scratch to be executed on remote computers.

2.1.7 Grid Infrastructure for Non-Grid Environments (Ginger) middleware

The main concept of the Ginger project [SFV10, VRF07, RRV10] is that any home user may take advantage of idle cycles from other computers, much like SETI@Home. Donating idle cycles to other users to speedup other users' applications and by doing so, they would also take advantage of idle cycles from other computers, to speedup the execution for their own applications. To leverage the process of sharing, Ginger introduces a novel application and programming model that is based on the *Gridlet* concept.

Gridlets are work units containing chunks of data and the operations to be performed on that data. Moreover, every *Gridlet* has an estimated cost (CPU and bandwidth) so that they can try to be fair for every user that executes these *Gridlets*. They also say that these *Gridlets* are *submitted* by nodes and *serviced* by other nodes, and then *returned* as results to the submitting nodes. By these means the global resources would always be occupied taking advantage of all idle resources, and giving home users the opportunity of executing their own tasks in a distributed way, with acceptable performance.

This project also tries to span the boundaries of the typical grid usage, enabling the Internet users to take advantage of the Grid features, previously unavailable to the common user. The project also employs a P2P model to provide a large-scale deployment in a self-organized way.

2.2 Resource discovery in P2P networks

The term resource is used to include hardware, software, licenses, Grid services, and others alike [HHK04]. And with, distributed computing, in an underlying network topology, comes a problem of finding resources for given problems.

P2P file sharing systems have always been dealing with such problems [MRPM08], i.e., to search for files within its peers without having to contact a single node to find them.

Since there is a lack of a central administration in P2P networks, the search for files has to include all its peers and has to be redone every time any node requests a resource. This happens because those resources might be different among peers and may not be available for infinite time or always in the same position.

Some approaches regarding resource discovery in P2P networks have already been done. As such, *Flooding* [PFA$^+$05] is the earliest technique used, where a message is sent to the network until some peer answers (or the message times out or the number of hops-to-live reaches zero).

To overcome problems of excessive traffic, alternatives include other *blind* methods such as *Random walks* and *multiple random walks* [TR03], that crawl the network randomly through its peers to find the answer.

Hybrid methods that combine flooding with Random walks were also considered. Such as, *direct searches* [LCC$^+$02] and *forwarding indices* [CGM02] that can only be implemented when there is some information about the resources and its peers, meaning that it follows a not-so-random walk through its peers to locate the resources with the exact (or close) match.

With Structured P2P systems, the attempt to always control where the contents should be, lead to explore the alternative of using distributed routing tables (Distributed Hash Table - DHT) [Man03]. Meaning that, peers and contents are mapped, using a *hash* function, to a key space. Moreover, the users and resources, that are part of the system, are organized in a rigid structure that facilitates their location on the network. However, this approach can only function when the resources are well known.

Some approaches to resource discovery have contemplated the solutions of integrating the idea of Social network within a Grid [GDY06]. Meaning that, to better guide the queries to the right resources that information has to be known among some of the peers. Furthermore, by simply sharing resource information with who they know, it increases the chance for other peers to reach it. Therefore, with this approach nodes do not need to know who exactly has the resources or to crawl the entire network to locate them.

2.2.1 Cluster Computing on the Fly (CCOF)

The Cluster Computing on the Fly [LZZ$^+$04] is an open P2P system that harvests idle cycles from users. It supports the scheduling of applications in a distributed model, where any peer can share or consume idle cycles.

Peers first join a specified community that is associated with how the peers are donating their idle cycles. Afterwards, they form a cluster for resource discovery and scheduling of workloads.

Regarding the resource discovery in CCOF [ZL04], peers use a local scheduler that communicates with other peers (schedulers) for sharing their resources. Each scheduler coordinates with other peers that it knows.

For comparison reasons, they used five algorithms for resource discovery, to know how the performance would be influenced by the dynamic nature of the system.

Centralized search: when the client enters the network and is willing to donate its idle cycles, it reposts its profile information to the central server. The clients then can ask the server for idle cycles, which it tries to match with other peers (clients) that are sharing.

Expanding ring search: when the client needs cycles it asks its direct neighbors. If there are no sufficient cycles it then requests to peers one hop further, repeating the procedure until it has enough cycles.

Random walk search: the client requests cycles from randomly selected neighbors, and they request from their neighbors until it reaches a certain limit.

Advertisement based search: when the peers join the network, they propagate their information to the neighbors. In order for the information to be disseminated through the peers.

Rendezvous point search: this method uses dynamically selected points in the network for the hosts to advertise their information profile and to gather idle cycles.

The comparison between these methods suggests that having selected points (Rendezvous points) for communication would be best for performance issues. This method also showed high job completion rate and low message overhead, although it favors large jobs under heavy workloads. Meaning that resource discovery would converge to a client-server based method, instead of a P2P method.

2.2.2 Mobile Agents in Resource Discovery

A software agent is defined as a computing entity that performs some task or tasks on behalf of someone or something [CH97]. A mobile agent has an additional property of not being bound to the system of which it started. *Aglets*[9] is a Java based implementation of such agents.

Using mobile agents for resource discovery [Dun01] in P2P systems, can lead to improve performance in bandwidth. Since agents can roam freely from the starting node, they can visit other peers updating them with the latest information and be updated with the current users' information. Furthermore, these agents would only carry with them the necessary information to update or be updated for a custom usage.

2.2.3 Juxtapose Project (JXTA)

The JXTA project [TAA$^+$03] was started by Sun Microsystems[10] intended to be a open source P2P protocol. Its protocols are completely hardware and software independent. While a complete description of JXTA is beyond the scope of this work, there are some aspects worth mentioning.

For sending and receiving messages their system uses a lightweight parser that supports XML objects [BPSM$^+$00], considering this it parses documents without the peers awareness of how it is done.

Moreover, JXTA consists of six protocols, but the most relevant for this work are the Peer Discovery Protocol and Peer Information Protocol.

The Peer Discovery Protocol allows peers to locate other peers, groups of peers and advertisements in the networks, using XML based messages to communicate with rendezvous peers. This type of nodes are considered as super-peers or coordinators of the network, that store advertisements of the dependent peers.

The Peer Information Protocol enables peers to determine the status and capabilities of other nodes. It provides read-only access to properties, described as using *strings* to name them.

2.2.4 Social-P2P

Social-P2P [LAM07] is a social-like P2P algorithm for resource discovery. They describe it as being analogous to the way humans interact in Social networks.

[9]IBM Japan Aglet: `http://www.trl.ibm.com/aglets/index.html` accessed on 05/01/2010
[10]Sun Microsystems, Inc.: `http://www.sun.com` accessed on 05/01/2010

They also note that knowledge is passed on by people to others in these networks as a means of sharing information, and also it is discussed that people recall information in memory to find the right persons to interact with, when searching for a given resource. These persons, which are recalled from memory, may directly relate to their requests.

However, in most circumstances, these people are recalled because they had knowledge similar, or in the same context of the requesting resource, instead of actually knowing about it. Meaning that, a person may be recalled because of the information topics of the requested resource.

Social-P2P makes use of this information in order to direct searches appropriately, by having community-based networks, and mimicking human interactions in Social networks.

Their routing algorithm involves three phases. A node receiving a query which needs to be forwarded, firstly searches the local knowledge index for the peer nodes directly associated with the requested topic. In a second phase it searches for peer nodes that are sharing content associated with the interest area of the requested topic. The peer nodes are ranked according to the interest they showed on the related topic, and thus the search selects the peers with higher degree of correlation between the requested topic. The third phase randomly picks cached peers if there are not enough peers from the previous phases.

An added observation is that they noticed the Small-World phenomenon on their networks with a high clustering coefficient and a short average path length. Meaning that, the networks were simulating a Social network instead of random created networks.

This algorithm serves as a demonstration that human interaction strategies are successful for resource discovery in P2P networks. Although it was only considered, in their simulations, a dynamic environment with probabilistic request structure and file sharing.

2.2.5 Web Ontology Language (OWL)

Resource matching in large environments can become increasingly difficult. Thus, the specific attributes needed to describe resources are getting more complex, in the same manner the requests are getting more differentiated. Therefore, OWL [HHK04] addresses this issue and is recommended by W3C.[11]

This language serves to describe ontologies (classes and relations between them) to be used on the Internet, which are inherent in Web documents and applications. Also, describing the hierarchy of resources that different computers can have.

Every peer keeps a record of resources, where every individual resource is a member of one or more concepts, which are stored in peer catalogs. Furthermore, each peer may have its own, possibly incomplete, ontology. Meaning that, it can be completed with the knowledge distributed over the network. The idea is to have a assertional box (A-Box) and a taxonomical box (T-Box).

The T-Box stores concepts and relationships, as for the A-Box it stores concepts and role assertions, meaning the concrete knowledge about individual domains.

By classifying every resource they can assume to have a search with the right conditions to arrive on a

[11]OWL Guide: `http://www.w3c.org/TR/OWL-Guide` accessed on 05/01/2010

knowledgeable result. They also assume that peers do not need to know all the terms used. To achieve this, they combine the knowledge from all peers within a distributed classification DAG (directed acyclic graph), and queries are resolved against it.

2.3 Social networks

Social networks are popular infrastructures for communication, interaction and information sharing on the Internet. Anyone with a desktop computer and a Browser can access such Web sites, like Facebook, MySpace, Orkut, Hi5, YouTube, Linkedin and many more.[12] They are used to interact with other people for personal or business purposes, sending messages, posting them on the Web site, receiving *links* to other Web sites or even sharing files between people.

Like in real life social interactions [Sco88] people tend to interact with many others along their lives, some of those are called friends which the interaction may be daily. In the Social networks the basic (real life) behaviors or interaction patterns still applies.

By grouping people in the same areas or topics, it should be easier to exploit those interactions, because people might understand better what the distributed tasks will accomplish and may be willing to participate in those kinds of works.

Social networks have already began to sprout new ideas to exploit them for uses other than human interactions, such as using it for enhancing Internet search [MGD06] and leveraging infrastructures to enabling ad-hoc VPNs [FBJW08], which will be addressed in the next Subsections.

2.3.1 Analysis on Social networks

To begin to understand why Social networks may be important for public-resource computing we have to study and analyze how users interact with each other in such networks, or how communities are formed.

Some studies of Social networks such as [WBS+09, MMG+07, AHK+07] focus their attention into how users and groups interact with each other in the course of time and to quantify it so we can learn the evolution in time of these networks.

These studies have reinforced the idea that those networks follow a power-law graph (Fig. 2.1) (also called the 80-20 rule, or may follow a logarithmic graph) and have properties of Small-World networks. Meaning that there are more users with fewer links than users with many. A user that has many links (to other users), which can be in the thousands, does not mean that he/she will interact with everyone most of the time, these interactions are confined to a small group of users of all of those that the user is linked to. It is also assumed that the interaction degree is lower than the social degree, meaning that users tend to have more links to other users, rather than the ones they frequently interact with.

Small-World networks can be categorized by the following properties: the local neighborhood is preserved; and the diameter of the network, quantified by average shortest distance between two vertices, increases logarithmically with the number of vertices n. It is then possible to connect any two vertices in

[12]List of Social Networks: `http://en.wikipedia.org/wiki/List_of_social_networking_websites` accessed on 05/01/2010

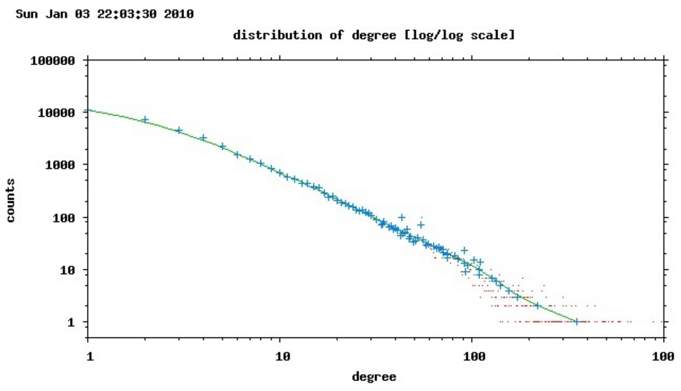


Figure 2.1: Power-law Graph (typical for a Social network)[11]

the network through just a few links [ASBS00]. Furthermore, growing networks can be hindered by two factors: *Aging of the vertices*, where vertices no longer connect to newer vertices; *Cost of adding links to the vertices or the limited capacity of a vertex*, adding links to the networks may not be possible if there are constraints of space/time.

We can also see Social networks' users in a P2P perspective, where users with many links are super-peers, connected by peers with fewer links and probably not always from the same groups.

Many Social networks also have ways of connecting users without being linked as friends, these connections are called groups or communities, where knowledge is exchanged within a specific topic of interest. The creation for such groups and consequently taking shape and evolve over time is inherent in the structure of society, meaning that each people have the tendency of coming together to share knowledge of a particular theme [Bac06].

2.3.2 Facebook and OpenSocial

Figure 2.2: Facebook and MySpace (OpenSocial-based Web site) logos[12]

There are many Social networks in the Internet[13]. The focus on Facebook and OpenSocial (Fig. 2.2 depicts the logos well known to the users) is explained by having access to the databases, by means of the APIs they export. Moreover, Facebook claims to have 500.000.000 (as of July 21 of 2010) users and MySpace (one Web site that uses the OpenSocial API) claiming to have more than 130.000.000 registered users. The potential of these networks for global distributed computing is best compared to other networks.

[11]Taken from: `http://pgp.cs.uu.nl/plot`

[12]Taken from their respectively Web sites.

[13]List of Social Networks: `http://en.wikipedia.org/wiki/List_of_social_networking_websites` accessed on 05/01/2010

Furthermore, the Facebook API[14] and the OpenSocial API[15] enables Web applications to interact with the server using a *REST*-like interface[16] or in case of Facebook also a *Graph* interface.[17] This means that the calls from outside applications are made over the Internet by sending HTTP GET and POST requests.

An example of a Facebook application is Progress Thru Processors,[18] it executes a BOINC system to do distributed computing, when the computer has idle cycles to spare. Through the applications' interface on Facebook it is able to track contributions from users and share updates with friends to intentionally promote distributed projects to other users.

2.3.3 PeerSpective: Social network-based Web Search

The World Wide Web and Web search engines have transformed the way people find and share information [MGD06]. Explicit links between contents, called Hyperlinks, are used to navigate through Web sites, and are used by search engines to crawl the Web, indexing content, ranking information or estimating the relevance of the content for a search query.

PeerSpective[19] tries to merge the Social networking with search engines, to improve their ranking system, by understanding how people do searches. They employ the search results, made by friends, into the ranking system. In overall, they claim that the system can be leveraged to improve the quality of search results for a given group of people.

The system works by indexing search content and querying friends for their searches including the extra results, which may be relevant to the question, on the search Web page.

A lesson to be taken is that *"Social networks can organize the world of information according to the tastes and preferences of smaller groups of individuals"* [MGD06].

2.3.4 Social VPN

The main objective of Social VPN [FBJW08] is to securely interconnect Internet users, where P2P network tunnel links are created, as a result of connections established through Social network infrastructures.

Friendship is determined by the Social network one is connected to (i.e. Facebook). It then provides a virtual private network, by contacting directly with other friends. Social VPN also uses a virtual IP network over the P2P system called IPOP (IP over P2P) [GABF06] for the communications between peers.

For peer discovery, they use the Social network APIs to query relationships. As Facebook provides a set of Web services that can be accessed through a *REST*-based interface. This means that all these queries are done through HTTP GET and POST requests.

[14]Facebook developer Wiki: `http://wiki.developers.facebook.com/index.php/Main_Page` accessed on 05/01/2010

[15]OpenSocial Specifications: `http://www.opensocial.org/specs` accessed on 05/01/2010

[16]Representational State Transfer: `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm` accessed on 05/01/2010

[17]OpenGraph Protocol: `http://opengraphprotocol.org` accessed on 23/08/2010

[18]Progress Thru Processors: `http://www.facebook.com/progressthruprocessors` accessed on 05/01/2010

[19]PeerSpective: `http://peerspective.mpi-sws.org` accessed on 05/01/2010

2.3.5 Social Cloud

Cloud computing [AFG$^+$09] derives from resource-sharing environments, and work with the intent of bringing those environments to Internet users. Also, it was created a relation between the resources given and received, meaning that in order to acquire resources a user can buy, sell or exchange them in "marketplaces", which provide lists of resources to be used (according to a virtual transaction) by any user.

Social Cloud [CCRB10] is introduced as being a model that integrates social networking, cloud computing and "volunteer computing". In this model, users can acquire the resources by exchanging virtual credits, making a virtual economy over the social cloud computing.

They also refer that it is a scalable computing model, where users' resources are dynamically provisioned amongst a group of friends. Also, adding that the model is similar to a Volunteer computing approach, where friends share resources amongst each other for little to no gain.

Their idea is that users can gather resources from their friends (either by virtual compensation, payment, or with a reciprocal credit model [MBAS06]), which makes this model approaching the public-resource sharing objectives.

Furthermore, they state that there are a number of advantages gained by leveraging Social networking platforms, such as gaining access to a huge user community, exploiting existent user management functionality, and rely on pre-established trust formed through user relationships. However, the trusting relationship of friends, may not be always the case[20] in Social networks such as Facebook.

2.4 Deployment Mechanisms and Code Execution via the Web

To navigate through Web sites, for common users, the most common way is to use a *Web Browser* (i.e. Internet Explorer[21], Chrome[22] among others). Browsers are user applications (named clients) that follow generally a client-server architecture and they play an important role to access Internet content and achieve communication between people [BTM07].

Furthermore, they contact servers by using a standard protocol named HTTP (Hypertext Transfer Protocol).[23] This protocol is used for retrieving interlinked resources, called hypertext documents. Also, this protocol follows a request-response sequence of messages, where the basic request methods (or verbs) are:

- *GET*: requests a representation of the specified resource.
- *POST*: submits data to be processed to the identified resource (this may result in the creation of a new resource or its update).
- *PUT*: submits a document to be stored in the server.
- *DELETE*: deletes a document stored within the server.

[20]How Facebook could make cloud computing better: `http://spectrum.ieee.org/computing/networks/how-facebook-could-make-cloud-computing-better` accessed on 15/10/2010

[21]Internet Explorer Browser: `http://www.microsoft.com/windows/Internet-explorer` accessed on 05/01/2010

[22]Chrome Browser: `http://www.google.com/chrome` accessed on 05/01/2010

[23]HTTP 1.0 specification: `http://www.ietf.org/rfc/rfc1945.txt` accessed on 05/01/2010

Other languages can also be used either on the client or server, to generate HTML dynamic content [Goo98], e.g. Asynchronous JavaScript and XML (AJAX)[24] being client side, Hypertext Preprocessor (PHP)[25] being server side.

AJAX [CPJ05] is a integration of consolidated technologies, such as JavaScript and XML, used to obtain new functionality and more control over the Browsers' contents. It is generally used to develop Web applications, that serves to interact with Web servers without the users' knowledge or perception.

Also, it is used to provide the user with a continuous method of interaction (within the browser environment), meaning that the Javascript module fetches Web contents and displays it to the user without having to switch to another Web page (also called non flickering effect).

Representational State Transfer (*REST*) [FST02] is a style of software architecture for distributed Hypermedia (including graphics, audio, video, plain text and hyper links) systems. The main concept is that resources existing can be referenced with a global identifier (e.g. URI in HTTP), and also the exchange of a representation of a resource can be applied without any constraint of state.

However, the client may need to understand the format which the information (representation) is returned. Typically the format used can be one of the following.

- HTML (HyperText Markup Language) consists of a document format with structural markers.
- XML (Extensible Markup Language) has the same concept of HTML documents, although it is generally used to represent arbitrary data structures, for example in Web services.
- JSON (JavaScript Object Notation) is described as being a lightweight data-interchange format, made in order to ease the computational parsing of data. It is generally used on the Web, because it is simpler for Browsers to parse and generate it, consuming less CPU time than other formats.

The Open Graph protocol[26] was originally created at Facebook, and it is an extension to the HTTP protocol in order to enable Web pages to become rich objects in a social environment. Meaning that, any Web site could use this technology to organize their information in a structured way, similar to Facebook pages. Also, they tried to build it on standards (RDFa)[27] to create a more semantically aware Web.

The idea of integrating distributed computing with Web browsers has already surfaced on the Internet. An example to this, is the Collaborative Map-Reduce,[28] this application code uses Javascript to interact with the Web server, requesting jobs to be fulfilled by the users' Browser and posting the results back on the server.

This method does not account for the lack of resources that the users' computers might have, or even a cycle-sharing environment.

[24] AJAX article: `http://www.adaptivepath.com/ideas/essays/archives/000385.php` accessed on 05/01/2010
[25] PHP Web site: `http://php.net` accessed on 05/01/2010
[26] Open Graph Protocol: `http://opengraphprotocol.org` accessed on 23/08/2010
[27] RDFa standard: `http://www.w3.org/TR/rdfa-in-html` access on 23/08/2010
[28] Collaborative Map-Reduce in the browser: `http://www.igvita.com/2009/03/03/collaborative-map-reduce-in-the-browser` accessed on 05/01/2010

Furthermore, their concern was to apply the Map-Reduce algorithm [DG04] on the data collected from the server which it was retrieved. This algorithm is composed by two steps:

- Map step: where the master node takes the input, partitions it into smaller sub-problems, and distributes those to worker nodes.
- Reduce step: where the master node takes the answers to all the sub-problems and combines them in a way to get the output (the answer to the problem it was originally trying to solve).

This algorithm ensures parallelism in the steps that are performed, meaning that the maps (or reduces) created can be processed in parallel with each map (or reduce).

The Collaborative Map-Reduce, would then use this algorithm combined with the processing power from users' computers from all over the World, to perform the algorithm steps while the user is browsing a Web site.

Another example of distributed computing using Web browsers is Plura Processing[29], which is a proprietary executable code made to enable idle cycle-stealing.

Their idea is that everyone that browse the Internet, has idle cycles that could be used for other purposes, and thus they "steal" idle cycles from users' computers to perform determined tasks.

Their reason is that any user can sacrifice their CPU time, even without their knowledge, to benefit computation intensive projects (much like SETI@Home). However, this programming style may not be best to suite the users, because they need to understand the tasks' relevance.[30]

Moreover, they use simple Web pages and games[31] (Adobe Flash[32] based) to embed their processing code to execute the needed tasks.

2.5 Summary

The P2P networks and Grids are the most common types of sharing networks [TTP+07], where Grids were created by the scientific community in order to process computation intensive applications, which would take more time in normal desktop computers without parallelization, or by using supercomputers (which may not be available to everyone), i.e. large scale simulations or data analysis.

P2P networks, on the other hand, are mostly used for file sharing between users (either with desktop computers, or mobile devices). However, these networks can also be used in other situations.

As we previously discussed, in Section 2.1, P2P networks can be categorized in terms of their structures as being unstructured systems, where the placement of contents is unrelated to the overlay network. Structured systems, where it is attempted to control the placement of contents and its peers. And also, hybrid systems, which were created to merge both systems and retain their benefits.

[29]Plura Processing: `http://www.pluraprocessing.com` accessed on 05/01/2010

[30]Plura Processing response to the Digsby Controversy: `http://pluraprocessing.wordpress.com/2009/08/24/our-response-to-the-digsby-controversy-new-terms-of-use-affiliate-auditing` accessed on 15/10/2010

[31]Examples of games that use Plura Processing: `http://www.pluraprocessing.com/games/seeexamples.html` accessed on 12/10/2010

[32]Adobe Flash: `http://www.adobe.com/products/flashplayer` accessed on 13/10/2010

We also described, in Subsection 2.1.4, some global distributed computing projects that make use of these technologies, to solve their computer resource shortage (CPU time), by using the millions of Internet enabled users' computers all over the world.

Such projects like SETI@Home [ACK$^+$02], Folding@Home [LSS$^+$09], among many others,[33] use the BOINC platform to do distributed computing for their own projects. This platform, which emerged from the SETI@Home project, sends data to be processed on volunteer computers, and retrieve their results by only using their Internet connection and the idle cycles that they can spare, in order to further advance the studies comprising their projects.

Other earlier projects such as Distributed.net[34] and GIMPS[35] also use this concept, although they use their own systems to do distributed computing.

On all these projects we can say that they do not have the flexibility to change their own topic of research, and also only used to further advance their own research.

Furthermore, we described in this section, other platforms that intended to give the ability of cycle-sharing to common users, such as XtremWeb [FGNC01], nuBOINC [SVF08], or Ginger [SFV10, VRF07, RRV10]. Each of them employ their own platforms to enable common applications to be executed on peers, to process data from users' tasks. Although, their interaction is only done on P2P networks, meaning that their networks have to be created by the users.

Also, to leverage the process of sharing resources, Ginger introduces a novel application and programming model that is based on the *Gridlet* concept. These *Gridlets* are work units containing chunks of data and the operations to be performed on them.

On Resource discovery, Section 2.2, we discussed that P2P networks have already dealt with such problems. One of the earliest techniques used was Flooding [PFA$^+$05], which would send a message to anyone in the network until the resource was found (or a limit was reached).

Other blind methods were also considered, such as Random walks or multiple random walks [TR03], that crawl the networks randomly to find the location of a specific resource.

Also, other methods that combine the previous were considered, such as direct searches [LCC$^+$02], or forwarding indices [CGM02], which follow a not-so-random walks through the peers to locate the resource, they use information about the requested resource to match them exactly (or close).

Moreover, we described that structured P2P systems use distributed routing tables (DHTs) [Man03], to map resources to their locations, making a more rigid structure than the other systems.

We also reviewed other cycle-sharing projects that use some resource discovery mechanisms to efficiently search for them. CCOF [ZL04], in Subsection 2.2.1, compared some algorithms to know which would perform best in their platform. And the most promising method was Rendezvous Point Search, which uses dynamically selected points in a network for the hosts to advertise their information profiles and to gather idle cycles. Meaning that, the resource discovery would benefit with more centralized solutions.

[33] BOINC projects: `http://boinc.berkeley.edu/projects.php` accessed on 15/10/2010
[34] Distributed.Net: `http://www.distributed.net` accessed on 05/01/2010
[35] GIMPS: The Great Internet Mersenne Prime Search `http://www.mersenne.org` accessed on 05/01/2010

Also, we discussed the usage of a mobile agent for resource discovery [CH97], which would lead to an automated system where a peer would only need to release its agent onto the network, and it would roam freely gathering (and updating) information on computers. Although, this may be inefficient or impractical in case of low resource networks (or devices).

In the JXTA project [TAA$^+$03], the resource discovery uses XML based messages to coordinate resources scattered on the network, and advertise them as well. Giving structure to resource request information, in order to locate them within certain parameters.

We also described, Social-P2P [LAM07] which is a algorithm that mimics human interactions in Social networks in order to locate contents and files in a more appropriate way. Meaning that, the resource topic is used to search for it, within groups of users. Also, they demonstrated that human interaction strategies are successful to locate resources within P2P networks.

We finished this section with OWL [HHK04], which serves to describe ontologies to be used in Internet contents, making resources well known within peers, and facilitate their search.

Social networks, in Section 2.3, are popular infrastructures for communication, interaction and information sharing on the Internet. A user only needs his/her Internet enabled device (including desktop computers, notebooks, mobile phones, among others) to access Web sites, such as Facebook, MySpace, Orkut, LinkedIn, and many others,[36] to be able to send or receive personal or business information.

We analyzed some of the aspects of Social networks, in studies like [WBS$^+$09, MMG$^+$07, AHK$^+$07] which reinforce the idea that these networks share similarities with Small-World networks, in fact they follow a few properties, such as being able to connect any given user with another, with only a few links, and also a small group of people are highly connected to others, while a larger population is less linked, on these networks.

Although, a person can have as many as thousands of connections, they do not interact with all, most of the time. The interactions are only confined to a small subset of their group of friends. Also meaning that, the interaction degree is lower than the social degree.

Moreover, these networks have other ways of connecting users, normally called groups or communities, where knowledge is exchanged from a particular topic of interest.

Also, we can see these networks in a P2P perspective, having users with high connectivity (as super-peers) connected to users with fewer links (peers), meaning that we could extend Social networking to P2P models.

Social networks, such as Facebook and OpenSocial-based, caught our attention, because they are well known within the common users (i.e. Facebook claims to have 500.000.000 accounts), and also because they export their own APIs,[37] making it possible for outside applications to interact with those users, while also obtaining their information. Meaning that, the resource potential of these networks for global distributed computing is greater than to create a new network.

Furthermore, some projects have surfaced which utilize these concepts to their own needs, such as

[36]List of Social Networks: `http://en.wikipedia.org/wiki/List_of_social_networking_websites` accessed on 05/01/2010
[37]Facebook developer Wiki: `http://wiki.developers.facebook.com/index.php/Main_Page` accessed on 05/01/2010

PeerSpective [MGD06], which tries to merge Social networking with search engines, to improve their ranking systems.

Recent projects such as Social Cloud [AFG+09] introduce a model to integrate social networking, cloud computing and "volunteer computing" in order to take advantage of their properties. Their idea is that users can share resource with their friends, using a payment method, or with a reciprocal sharing method [MBAS06]. Also, they sought out these networks, because they could gain advantages by leveraging Social networking, such as gaining access to a huge community, and rely on established trust through users' relationships.

The most common way, as reviewed in Section 2.4, for users to view Web sites such as Facebook are the Browsers, which are client applications that display the contents of Web pages to the users, in a organized and graphical way. Also, the generally used protocol to retrieve and send this information from these Web servers is the HTTP protocol.[38]

Although the main language used to create Web pages is HTML, other languages can be considered either on the Browser side or on the server side [Goo98], such as PHP, Javascript, AJAX, Adobe Flash among others.

Each of these technologies were created to provide developers a more reliable or controlled functionality over the Browsers, and also to provide users with a continuous method of interaction.

We also reviewed the *REST* [FST02] and *Graph*[39] protocols that are used by Web sites, such as Social networks, in order to provide outside applications with an easy way to interact (and also retrieve information) with them.

In the Section 2.4, we also described two executable codes that use the users' Browsers to do cycle-stealing, instead of having a desktop application for it. They embedded their code in Web applications (such as Web sites or games) in order to gain idle cycles from users' computers to be used by their projects, as was done by distributed computing projects such as SETI@Home.

[38]HTTP 1.0 specification: `http://www.ietf.org/rfc/rfc1945.txt` accessed on 05/01/2010
[39]Open Graph Protocol: `http://opengraphprotocol.org` accessed on 23/08/2010

3
Architecture

Contents

This work uses a Social network (Fig. 3.1), such as Facebook, to be able to discover resources for the execution of Jobs (which are composed of *Gridlets*)[VRF07] submitted by the users. Also, to discover computer capabilities and users' profiles, such as the groups which they belong to and their friends.

Users should be able to *install* SNCS, which is a Web-enabled platform, (Fig. 3.2) into their computers, giving afterwards the ability to *log in* into their Facebook account, by means of the *Facebook Connect*,[1] which is a Web page given by Facebook to enable the *log in* process for outside applications (known as Facebook applications).

Afterwards, the client application is able to interact with the Social network server, meaning that it intercepts/sends messages from/to other users or groups, while also discovering users' computer profiles by contacting the *Graph* server.[2] The client application also gives the user the ability to initiate a Job, by using SNCS' user interface.

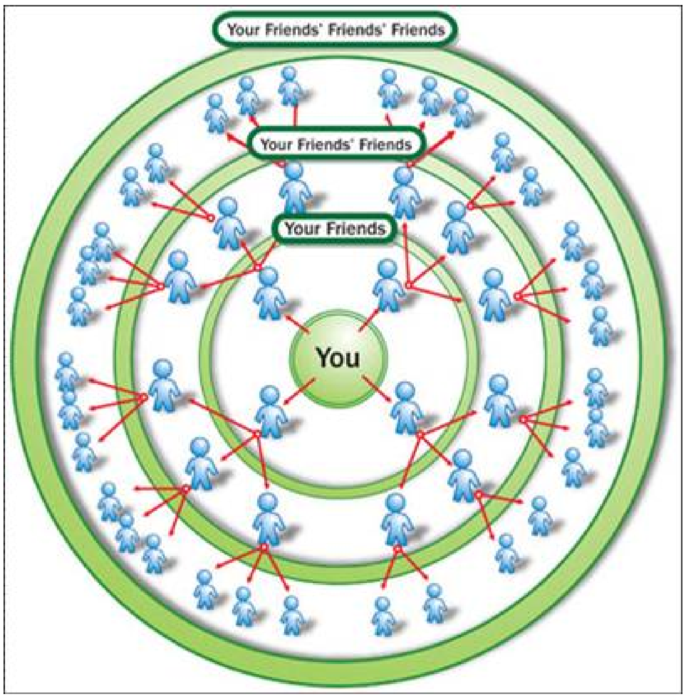


Figure 3.1: Social network interactions[3]

To actually locate resources through the Social network, SNCS has the ability of searching the local resources, by means of the *SIGAR* library, that gives information, such as processors status, memory available, number of processors, processors frequency. Such information is sent to other users upon request, or it can also be sent to the users' Wall, in order for everyone (people that has the ability to see the Wall) to get it without the users' permission. Also, this information may contain the programs that can be executed by the computer to process *Gridlets*, it is a configuration parameter that the user can deal with.

Social Networks for Cycle-Sharing should also have access to friends and groups through the Social

[1]Facebook Connect: `http://developers.facebook.com/docs/guides/web` accessed at 04/10/2010
[2]Facebook API: `http://developers.facebook.com/docs/api` accessed on 25/08/2010
[3]Taken from: `http://www.defenseindustrydaily.com/images/MISC _Social _Network _Circle _lg.jpg`

network API. It advertises users' availability to others, sending messages and scheduling tasks (i.e. search for informations, *Gridlet* acceptance) on them (Friends, Friends of Friend, Groups) in order to execute the tasks when users can spare their idle cycles (usually when they are in a *idle* or *away* state).

Eventually, SNCS starts listening and looking for requests that can appear on the users' Wall, friends' Wall or Registration post on the Applications' Wall. As Facebook does not allow people to interact with each other without being friends, the latter option was added to circumvent this inability, making it possible to gather resources from people outside the friends' domain.

The main approach for SNCS is to have a client application split into two parts: one that interacts with the Social network; and the other to interact with the users and the Ginger Middleware. The latter, should be the one responsible to create and regroup *Gridlets*, which is out of the scope of this work [VRF07].

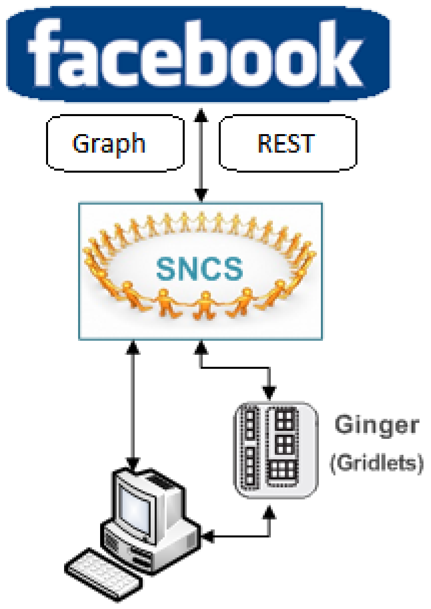


Figure 3.2: Social Network for Cycle-Sharing overview.

3.1 Design Requirements

This work aims to improve the Ginger Middleware[3] for resource and service discovery, by leveraging the possibility of having a network of users (Social network) already established. Therefore, the client application interacts with the Social network (Facebook) through Web Protocols named *Graph* and *REST* (which are an added layer to the HTTP protocol). As Facebook is still developing the *Graph* protocol and discontinuing the usage of *REST*, current operations within the client application makes use of the first protocol, although some operations can only be executed by *REST*. This requires that the client application has to understand both protocols and interact at the same level (*Graph* or *REST*), which is dealt with the *RestFB* library.[4]

[3]Nonetheless, the work described in this document could be applied in the context of other cycle-sharing systems.
[4]RestFb Web site: `http://restfb.com` accessed on 24/08/2010

Another requirement for SNCS is to know the computer's information that it should have at its disposal, such as number of processors, available memory, or the programs that can be executed to process *Gridlets*.

Moreover, SNCS should not interfere with the users normal usage of its computer or Facebook page. For that purpose, it schedules *Gridlets* according to user preferences, meaning that friends can have priorities for executing their *Gridlets*. To prevent overuse of the computer, while it is in a *Online* state, SNCS must be able to stop its activities, meaning that the processing of requests and *Gridlets* only happens when there are idle cycles to spare. And also, it should remove any unnecessary posts that could prevent the normal usage of the Facebook page.

3.2 SNCS Architecture

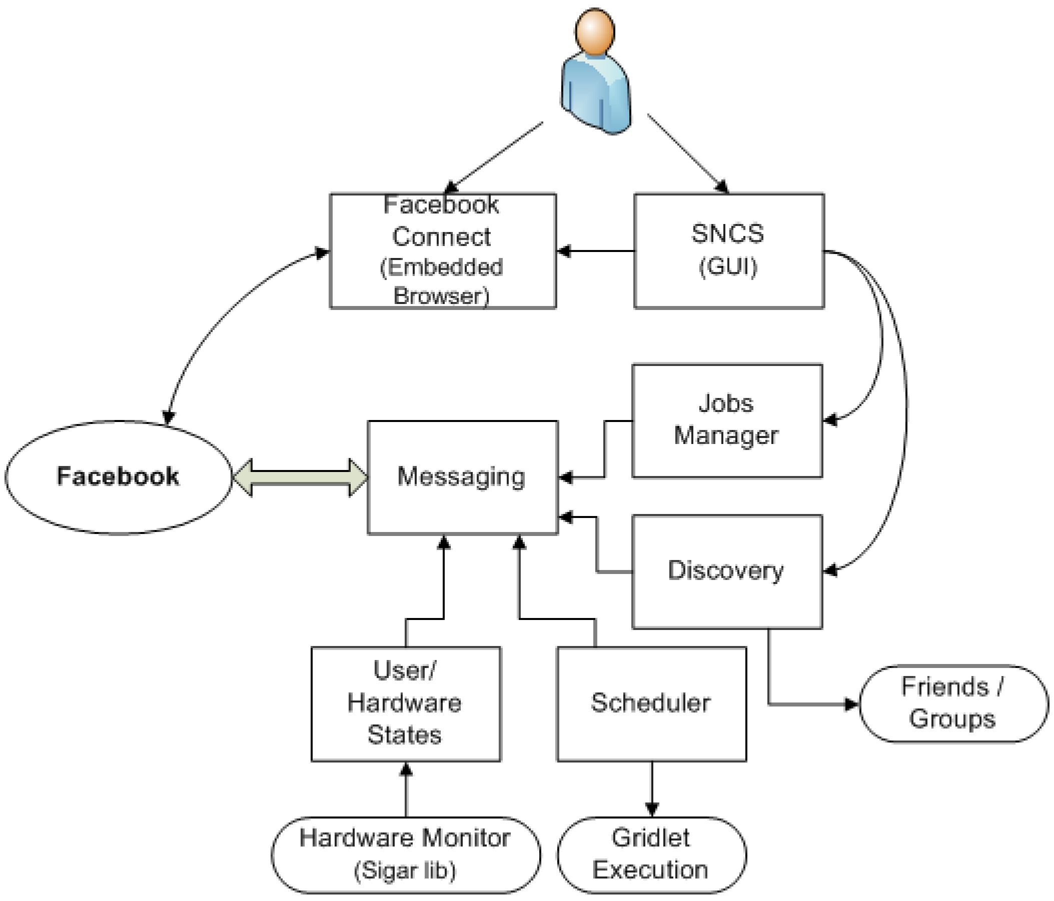


Figure 3.3: Social Network for Cycle-Sharing module view.

The SNCS architecture, depicted in Fig. 3.2, relies on a interaction with the Social network through the Social networks' API (*Graph* or *REST* protocols) for the purpose of searching and successfully executing Jobs; with the Ginger Middleware for *Gridlet* creation; and also the user's operating system to acquire the informations and hardware states that are needed.

Jobs are then consider to be tasks initiated by the users, and containing at least one *Gridlet* to be processed in someone else's computer, all Jobs should state what they need to execute those *Gridlets*, in order for the client application to search for specific users or groups.

A *Gridlet*, should contain the information necessary to process it, meaning that it has a data file to be transfered to another user and the arguments to be given to the executable program. The process of creating and aggregating the *Gridlets* should be managed by the Ginger Middleware and is outside the scope of this work [VRF07].

The architecture for SNCS is comprised of a set of components and their functions, depicted in Fig. 3.3. Each of them are described as follows.

SNCS (GUI): this module performs the main interaction with the user, meaning that it contains a graphic interface, described more throughly in the next chapter. It is responsible to establish the connection to Facebook, by starting the Facebook Connect module. It also loads all the necessary information onto the client application, such as the configuration of priorities, the Jobs that have been submitted, accepted and *Gridlets* in progress.

The user can submit a new Job using the appropriate user interface (Fig. 4.7), and this module is responsible to start the chain of events for processing that Job (search for users, acceptance and execution of *Gridlets*).

Facebook Connect (Embedded browser): this component serves to authenticate the user to Facebook, it displays the Web page given by Facebook for that purpose. Afterwards, it extracts the necessary *access token* for consequent access to the Facebook server. This token is given by Facebook to everyone that accepts this Facebook application, and has to be renewed within a determined time frame (the time frame is given by Facebook and not specific for every token).

Furthermore, it makes use of the JDIC library[5] to display the Web site for the user's authentication.

Messaging: is the main module for interacting with the Social network. It makes use of the *RestFb* library, that creates the JSON or XML objects, which are required to access Facebook *Graph/REST* functions. For any actions on Facebook the *access token* is required to be present, meaning that if Facebook Connect does not retrieve it then the messaging module fails to connect to Facebook. This module also contains the options necessary to read and write to the users/groups/Application Wall (or feeds) Posts or Comments and removing them as well; to gather information such as users' Facebook ID, friend lists and groups lists; and also to search for public Objects (Groups, Users).

Furthermore, some Facebook restrictions may apply to the interactions between the module and the Social Network, such as limiting the size of the messages, inability to erase Posts or Comments (made by other users).

The module also contains the schemas (Fig. 4.8) applied to the messages sent and retrieved, to specify what actions should be taken, whose details are described more throughly in the next chapter.

[5]JDIC: `https://jdic.dev.java.net` accessed on 15/10/2010

Jobs Manager: is the module that runs a cycle of the following tasks, named “checking” cycle.

- Verify submitted Jobs that the user has in progress, meaning that it tries to acquire the required information to send *Gridlets* to other users.
- Check for new Jobs from the users’ Wall, groups’ Wall or Registration Post that can be processed by the users’ machine, making sure that the required properties of the *Gridlets* are compatible, and thus accepting a Job.
- Verify accepted Jobs, meaning that after accepting a Job a *Gridlet* message should have been sent to the user, although it is not guaranteed that the requesting user still has *Gridlets* to be processed.
- Check for Job completion, when the client application has submitted a Job or a *Gridlet* it should be able to detect if it has been completed. When a *Gridlet* is not completed successfully, the module can retry to send it to someone that has accepted the Job.
- Check for messages that the client application needs to redirect to its friends, this method is necessary because Facebook restricts conversations to only the users that are considered friends.
- Check for messages that have been redirected to the user, in order for SNCS to answer on the Registration Post (in the Applications’ Wall), that was made prior by the requesting user, and thus adding the functionality of reaching other people rather than only the users’ friends, also the content of these messages should be requests to fulfill a Job or to send their computer information to the requesting user.

Also, after SNCS has acquired a *Gridlet* message, it hands it to the scheduler module for later execution. Moreover, this module has the task to remove all the Posts that are no longer necessary. And as the main module for checking messages on Facebook which uses the Internet, it can be stopped if the computer is in a *Online* state, in order to not interfere with the normal computer’ usage.

Discovery: this module serves as an addition to the latter module. Meaning that, it searches for friends and groups, in order to reach as many people as possible, to complete a Job. It sends messages to friends so that they can redirect those to their own friends (Friends of Friends method), while also sending messages to groups of interest for that specific Job. Making this module attached to the concepts of friends and groups.

It also is responsible to *register* the user in the Applications’ Wall, meaning that every user has a Post on this Wall, in order for other users, that are unable to directly contact them, to interact as if they were friends.

User/HW States: this module determines the state of the local resources, and takes in consideration the processors’ idle times, the Internet connectivity (that is essential to all processes) and the users Facebook state, in order to yield all the modules until a later time, when the processor has idle cycles to spare. Also, it sends the state of the SNCS client (*Online*, *Offline*, *Idle*) to the Social network.

The state *Online* should be performed when the user has decided that the client application should run.

The *Idle* state happens when the computer has idle cycles to spare, but it does not take into account the fact that the computer is being used and also if the user does wish that the client application needs to be *Offline*, the latter state prevails.

The *Offline* state means that the client application does not process any messages or *Gridlets*, stopping all processes related to this fact, because either there is no Internet connection (which is needed on the overall process) or that the user explicitly does not want SNCS to be running.

This module uses a submodule, named Hardware Monitor depicted in Fig. 3.3, that is comprised of the *SIGAR* library, which reports the system information needed to determine the availability of the resources.

Scheduler: this module is an addition to the *Gridlet* processing, making use of the priority lists (in the configuration), while also stopping that process when the computer does not have idle cycles.

The priority lists consists of friends and other people added by the user, in order for the client application to use the idle cycles on *Gridlets* belonging to the people with the highest priorities. Meaning that some *Gridlets* waits for a conclusion of others even if they arrived first.

The module starts a submodule that is responsible for processing the *Gridlet*, meaning that it transfers the data files to either sides, it executes the program that processes the data and upon completion it informs the originator of the *Gridlet* state, by sending a message to Facebook telling where it should retrieve the completed *Gridlet* or if the *Gridlet* was not completed successfully (may occur when there is an error on the executing program or client application).

3.3 SNCS Communications

SNCS, as it was described before, interacts with the user and the Social network, and therefore a protocol or flow of communication has to be established. The following demonstrates how the creation and execution of the *Gridlets* is being carried out.

The task for creating a Job, depicted in Fig. 3.4, which can be comprised of several *Gridlets*, is initiated by the user, by submitting the Job on the SNCS GUI. The information for a Job consists of the following items.

- The program that executes the *Gridlets*;
- The commands or arguments that are given to that program;
- The data file(s) that the client application needs to transfer;
- The number of *Gridlets* that comprises the Job (although this should be determined by the Ginger Middleware);
- And what are the requirements to execute the *Gridlets*.

Meaning that, the search is specified by the *Gridlets* requirements, i.e. a Job that consists of generating a image on POV-Ray, needs 4 processors and 2048Mb of memory, is specified on the client application by this information in order to search for users that could have these resources available.

Afterwards, it starts to perform the actions to complete the Job, such as sending a message onto the users' Facebook Wall and waiting for other users to respond to it; starting the discovery process that is able to find friends and groups that would be interested and or have the capability of executing the specified Job.

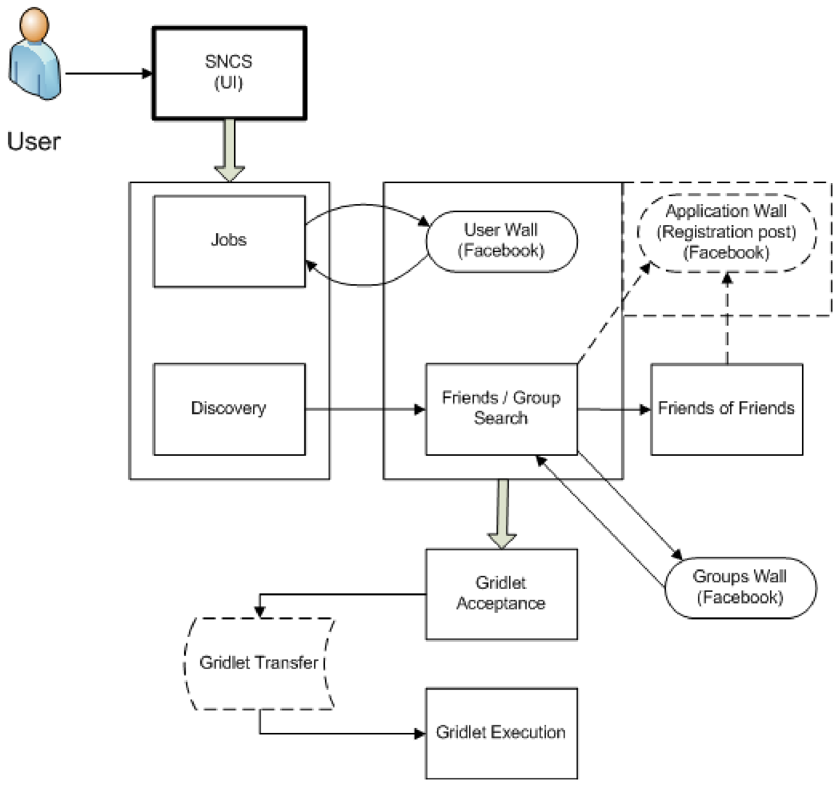


Figure 3.4: Communication Protocol in SNCS

The impossibility of directly contacting people and groups that are not in the friends' domain, such as Friends of Friends (FoFs), makes SNCS client route messages to the users' friends, in order for them to forward those messages to their own friends, making then viable to contact FoFs. The scale for this type of messaging could be larger, as in, the message could reach people that are our Nth degree friend, but it may end up *Spamming* users, and such actions are considered as a violation of the Facebook Use terms,[6] and as such SNCS only goes as further as FoFs (2nd degree). Also, the client application only contacts the users' groups that should be able to help for the specific Job. Meaning that, it searches for their computer information in order to know if they would be capable of processing the Job.

The discovery mechanism of SNCS tries to gather as much computer information as possible, and sends messages to the corresponding users and groups. While the Job part stays alert for incoming messages on the users' Wall.

[6]Facebook Use Terms: `http://www.facebook.com/terms.php` accessed on 26/08/2010

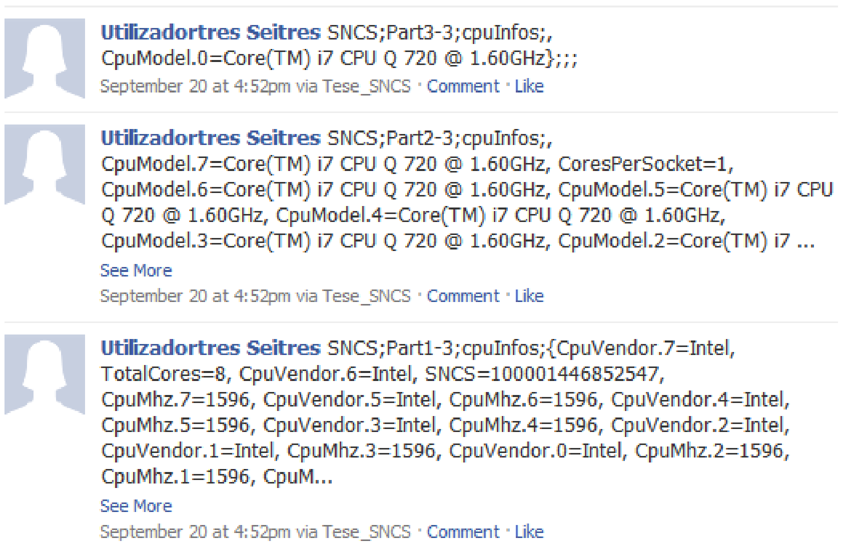
The people that receive messages (FoFs) and are not capable of directly contacting the originator, use the Registration Post on the Applications' Wall to respond to the redirected messages. This serves as a means of interaction with everyone that has the Facebook desktop application (client application), which enables the process of searching for people outside the scope of friendship.

Their client applications then try to match their own information to the expected Jobs and accept them accordingly, by sending an *Accept* or *Deny* message back to the originator. If the Job has been accepted the client application tries to fetch a *Gridlet* in order to execute it locally.

The transfer of *Gridlets* occurs after a client application has retrieved the *Gridlet* message, and determined that it has idle cycles to execute it. The transfer method can use a direct connection between the SNCS clients (acting as peers), however this method can be replaced by having a repository server or by sending the data file along with the message (if permitted by the Social networks' Use terms).

If the processing SNCS client determines that the execution of a *Gridlet* was in error, meaning that the processing program may return an error code, it sends a message to the originator informing that the *Gridlet* could not be completed. In case the error was within the client application, such as a client application crash, SNCS can still reacquire the *Gridlet* from the users' Wall, if the message was not deleted.

Afterwards, the originator of the Job receives all the *Gridlets* that have been processed, using the same means of transfer, and pass them to the Ginger Middleware for aggregation. Finally, the originators' client application leaves a Post message on the users' Walls, thanking them (or if the work was not properly done, another type of message is used).


Utilizadortres Seitres SNCS;Part3-3;cpuInfos;,
CpuModel.0=Core(TM) i7 CPU Q 720 @ 1.60GHz};;;
September 20 at 4:52pm via Tese_SNCS · Comment · Like

Utilizadortres Seitres SNCS;Part2-3;cpuInfos;,
CpuModel.7=Core(TM) i7 CPU Q 720 @ 1.60GHz, CoresPerSocket=1,
CpuModel.6=Core(TM) i7 CPU Q 720 @ 1.60GHz, CpuModel.5=Core(TM) i7 CPU Q 720 @ 1.60GHz, CpuModel.4=Core(TM) i7 CPU Q 720 @ 1.60GHz,
CpuModel.3=Core(TM) i7 CPU Q 720 @ 1.60GHz, CpuModel.2=Core(TM) i7 ...
See More
September 20 at 4:52pm via Tese_SNCS · Comment · Like

Utilizadortres Seitres SNCS;Part1-3;cpuInfos;{CpuVendor.7=Intel,
TotalCores=8, CpuVendor.6=Intel, SNCS=100001446852547,
CpuMhz.7=1596, CpuVendor.5=Intel, CpuMhz.6=1596, CpuVendor.4=Intel,
CpuMhz.5=1596, CpuVendor.3=Intel, CpuMhz.4=1596, CpuVendor.2=Intel,
CpuVendor.1=Intel, CpuMhz.3=1596, CpuVendor.0=Intel, CpuMhz.2=1596,
CpuMhz.1=1596, CpuM...
See More
September 20 at 4:52pm via Tese_SNCS · Comment · Like

Figure 3.5: Example of the Computer Informations on Facebook

3.3.1 SNCS Protocol

SNCS has an explicit communication protocol that it follows in order to discover resources and eventually use idle cycles from donor's computers.

As depicted in Fig. 3.6, when a user has submitted a Job for SNCS to execute, it sends a JobSearch message to the users' Wall as a Post, waits for replies which can be accept or deny messages and are comments from that Post. Moreover, each client application that respond to the search determine at that time if it is capable of performing the request, meaning that it determines whether it fulfills the necessary requirements and if it has idle cycles that can be spared.

SNCS also determines which of the users' groups are to be considered to have users that may be able to help with the processing. Moreover, the client application tries to read the Posts from the user's Walls to search for their computer information (CIs read). It then determines if there are users' computers that can carry out the Job, and starts the JobSearch message on those groups.

As FoFs cannot be directly contacted, we introduced a redirection protocol in order to reach them. Meaning that, the client application sends a redirection message (Redir message), as a Post to the user's friends' Walls containing a specific type of message, and also the location to where the FoFs' client applications should send their response.

After the FoFs' client application receives that message, it sends the response (determined by the message type) to the location specified in the message, which is normally sent as a comment to a Post in the Applications' Wall (note that the location written on the message is a Post created specifically for this type of communication, meaning that this Post was created by the requesting user, as the registration Post, on the Applications' Wall).

Afterwards, the requesting client application can start the JobSearch message on the Applications' Wall using the potential processing users' Post as the interaction point.

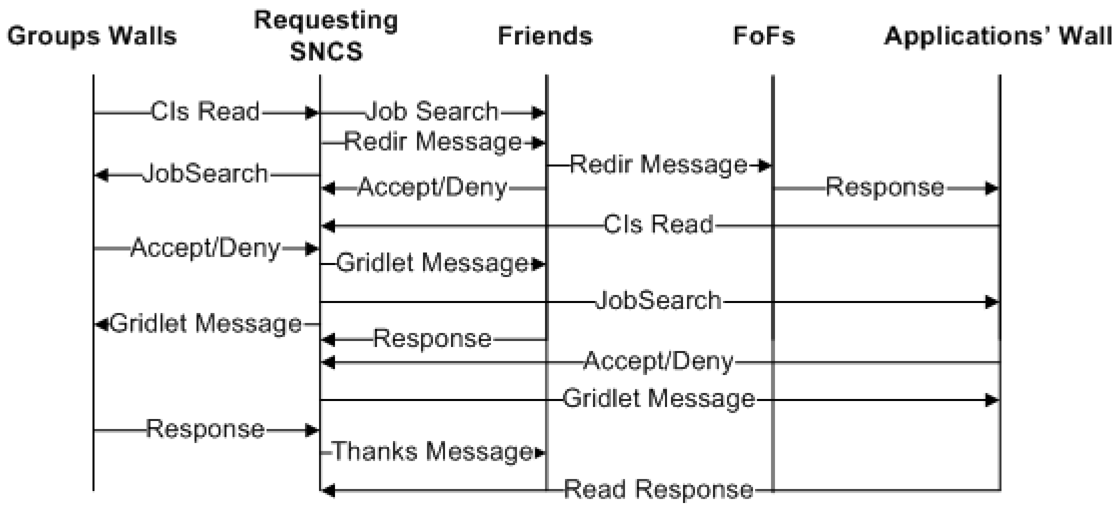


Figure 3.6: SNCS message flow

SNCS then sends *Gridlet* messages to those users that have accepted the Job, by sending a Post or Comment depending on which the user role is (Friend, FoFs, Group person). In case of friends, the *Gridlet* message is sent to the processing users' Wall, because the user can always read his own Wall. In case of FoFs the message is sent as Comment on the registration Post of the processing user, because as Facebook does not allow one person to directly contact another without being friends, and because the user can always read his own Posts. In case of people from Groups the message is sent to the JobSearch Post as a Comment, because there is no other way of contacting those persons (except for friends that are from the same groups) and for them to read the messages.

The client application then waits for responses to the *Gridlet* messages as Comments (on all the Walls that the messages were sent). When it receives a response, it determines where the resulting data is to be retrieved, or in case of an error (from the processing side) it retries to send the *Gridlet* to someone else that accepts the Job.

To end the communication and for future processing works (to determine the viability of sending a *Gridlet* to someone), it records a message on the processing users' Wall, as a "thank you" note (or it can be another type of message). However, FoFs and people from the groups cannot be contacted by the same way, thus that message is sent to the originator users' Wall.

3.4 Discovery Mechanism

The discovery mechanism consists in a "pull" and "push" methods. It tries to search for people that have the capability to do the work, by either retrieving their computers' information from the users' Walls (Fig. 3.5) or by requesting it. Furthermore, it verifies if there are any users capable of accepting a Job on the users' Group list in the same manner. To cover as much work as possible, the client application also searches between its friends' Walls if they have any Jobs that can be fulfilled.

The information gathered should contain a list of resources and properties of the users' computers, such as the number of processors, processor type, processor clock speed or frequency, processor model, total memory, free memory, list of user defined programs, although it should not be constrained only to these types of information.

SNCS attempts to match the needs of a Job to the information gathered, although it is out of the scope of this work, the matching of the informations should not fully constrain the execution or acceptance of a Job (as this system is comprised of volunteers), meaning that a semantics should be taken into account in order to approximately match these properties to what is needed [SFV10].

3.5 Gridlet Execution

A relevant entity in SNCS is the concept of Job, and it is important to understand how to perceive them. On that note, a Job that is submitted by a user should have associated with it what the *Gridlets* require in order to process them successfully.

As we said before, a Job is comprised of *Gridlets*, each of them has a piece of data that needs to be transfered and processed on another user's computer.

In order to process a *Gridlet*, SNCS has to send a *Gridlet* message to the users that accepted the specific Job. This message generally contains a JobID (which is the ID of the Job Search Post) identifying that it is the Job which the user has accepted; a *Gridlet* number for that Job, in order for SNCS to know which *Gridlet* the user is processing; the program and arguments that are given to it in order to process the data; and a location of a file(s) that should be *downloaded* by the processing client application.

The method of transfer is indicated on the file variable of the *Gridlet* message, meaning that the variable should be like Protocol://IP:Port/File, indicating which protocol should be used, the address that has be contacted, and the data file. The protocol that is mainly used on SNCS is a direct transfer method (from application to application), e.g. the file variable becomes 127.0.0.1:52392/data.file (without protocol tag meaning the application to application method), but other methods can be used such as a repository server, e.g. http://web.site/data.file. Another approach that can be pursued is to include the data file with the Facebook message, in this case the files have to be *uploaded* and *downloaded* using the *REST* or *Graph* protocols.

After SNCS makes sure that it is able to process the data file(s), it launches (or executes) the processing program with its arguments, waiting for a result, which can be success or error. In both cases, a message is sent back to the originator, informing that it completed the *Gridlet*. This message can either contain an error code, in order for the originator to be able to retry the *Gridlet* on someone that accepts the Job; or it can have the location of the resulting data, using the same transfer methods as explained above.

3.6 Prototypical example

Following we give a more detailed example of a Job submission and the steps SNCS takes to process it, as depicted in Fig. 3.7.

A user submits a Job, using the SNCS GUI (explained in a Section 3.2), with the following properties.

The processing client application needs to execute the program named pvengine64.exe, with the arguments "-A0.3 -W1280 -H720 -D -O"$dir.output$balcony_rend.bmp" -P +Q9 +R5 /EXIT /RENDER "$dir.exec$balcony.pov"", meaning that the Pov-Ray program will render an image with 1280x720 dimensions, using Anti-aliasing, with high quality and it requires the file balcony.pov to start the process, which should be downloaded from 127.0.0.1:52392/balcony.pov. The user also specifies what the program should need in order to be executed, such as "TotalCores=2", "MemorySize=1024", "Program=pvengine64.exe" and the number of *Gridlets* that comprises this Job.

SNCS would then start the search for resources using the specified requirements as a point of reference, on this note the requirements should be as accurate as possible to search for specific groups and users. The actions to locate the resources starts by sending a Job search message to the users' Wall. Meanwhile, it also requests the computers informations (CI discovery) from the people that are in the user's groups, in order to know which of the groups would be more willing to accept the Job. Also, it tries to send a message to its friends in order for them to redirect to their own friends (FoFs method), waiting for a reply on the Applications' Wall. The last message contains the information necessary to redirect it, meaning the Post ID for which it should be sent, and the type of message that the user should send (in this case computer information).

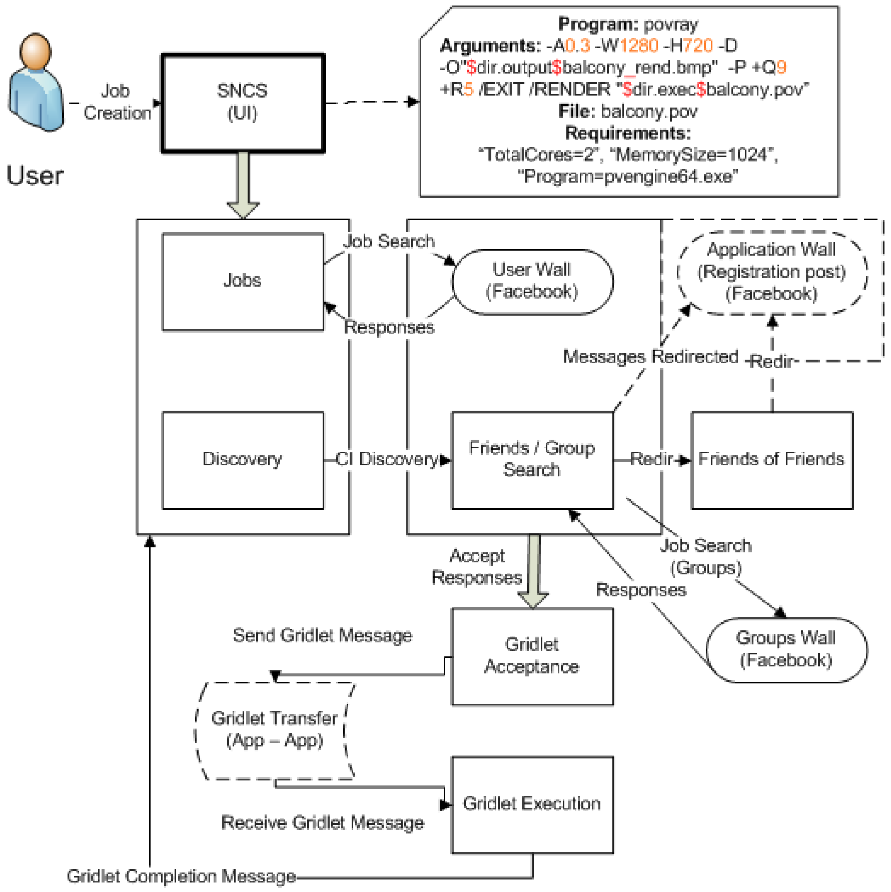


Figure 3.7: SNCS Prototypical example

Afterwards, SNCS reads the responses to the Job search, which can be accept or deny messages, meaning that even if someone would have the requirements to process the *Gridlets*, a user may not have idle cycles to spare, and thus denying the request. For the users that accepts the Job, SNCS sends a *Gridlet* message, until all the *Gridlets* have been sent, this message is then received by the processing client application.

SNCS does not have a specific way to choose between users that have accepted the Job request, except accounting with the “thanks” messages (which may not be guaranteed), thus it sends the *Gridlet* messages according to the order that the messages were received.

The client application does not verify the correct completion of the *Gridlets*, although this process should be included in order to give greater reliability assurance. However, the reassignment of a *Gridlet* occurs in case the processing client application encounters an error while executing it, sending an error message back to the originator.

In this example, the client application would use a Application to Application transfer method to send the “balcony.pov” file, although other methods can be used such as sending the file to a Web server and retrieving the results with the same method. Moreover, the *Gridlet* message should contain the necessary information in order for this client application to locate and retrieve the necessary data to be processed. Meaning that the protocol used for the transfer method should be read from the file variable on the message received, in order for SNCS to be able to retrieve it.

The receiving SNCS client, before it receives the data file, needs to consider the execution of the *Gridlet*, according to the computers' state and from whom it has originated, creating a queue of *Gridlets* when necessary.

From this example, the recipient client application would then call the program pvengine64.exe, that the user specified its location on the "Programs List", i.e. D:\Pov-ray\bin\pvengine64.exe, with the right arguments, waiting until the process finishes. After that, it sends a message to the originators' client application informing that it has completed and where it should retrieve the resulting file, this process also uses the same method as for the transfer of "balcony.pov" file, although it is also considered that other methods can be used.

To finish the interactions between the two users, the originators' client application sends a message to the users' Wall that has completed the *Gridlet*, thanking them for the time they have spent on it, when this is not possible (FoFs case) the message is sent to the originator users' Wall, in order to have a record of people that helped in a Job.

The originator of the Job requires that every *Gridlet* finishes, before it can pass them to the Ginger Middleware. Thus, it waits for all completion messages before it can erase the resulting messages from Facebook. Moreover, while the originator client application is performing this overall process, it also listens for Job search messages that can appear on friends and groups, in order to give its own idle cycles to other users.

4 Implementation

Contents

The implementation of SNCS aims for a simple use by the end-users. Also, the different types of operating systems used by people lead us to favor portability and therefore we used Java as the main language.

The choosing of the Social network was a natural process, the use of a very active network was considered as most important. Thus, the choice between using Facebook and other alternatives, such as OpenSocial-based networks, were almost irrelevant considering the fact that Facebook has much more registered users than any other Social network. Another consideration was the fact that Facebook exports its own API for outside applications (Facebook applications), and therefore many libraries (such as *RestFB*) have been created to facilitate the usage of the API. However, OpenSocial based networks also export their own APIs for outside use, which caused some dilemma on how much they could be compatible, but has mention before Facebook was the primary choice.

This chapter gives an insight on how the technologies were used, such as *Graph* and *REST* protocols. It also explains the internal representation of the data structures used on SNCS, and the Schemas used for the messages sent/received to/from the Social network chosen. The end section gives a view on how SNCS should be executed and some of the constraints that suffered from using Facebook as the Social network for users' interactions.

4.1 Used Technology

Social Networks for Cycle-Sharing was developed using Java for its portability purposes, it uses Facebook as its Social network for interactions between users' client applications. This Social network was chosen because it provides access to many features, and it is well known within the common users. For the purpose of interacting with the *Graph* and *REST* servers, the client application makes use of the *RestFb* library, that gives a simple and flexible way of connecting to them and conceal the use of XML or JSON objects.[1] However, the functions[2] or connections[3] (in Graph) have to be known, in order to use this library, e.g. to read the Posts on a users' Wall using the Graph protocol, we need a users' ID or Name in order for the library to access Facebook and retrieve that users' Wall, also we would need to know that the connection from the user object is "feed".

The Facebook Graph protocol gives the possibility to access any public object, such as users, Walls (or feeds), Posts, Comments, among others. Either using their unique identifiers (UIDs) or by their names, in order to get the object desired more easily. However, Facebook is still developing this technology and as such the functionalities may change the way of interaction or even may not work as supposed, for that purpose the use of the *REST* server is still an option.

We can also state that the IDs generated for each object are dependent on the previous objects, meaning that the UID for a Comment on a Post on a Users' Wall would become "UserID_PostID_CommentID" which uniquely identifies the Comment belonging to the Post of that particular users' Wall.

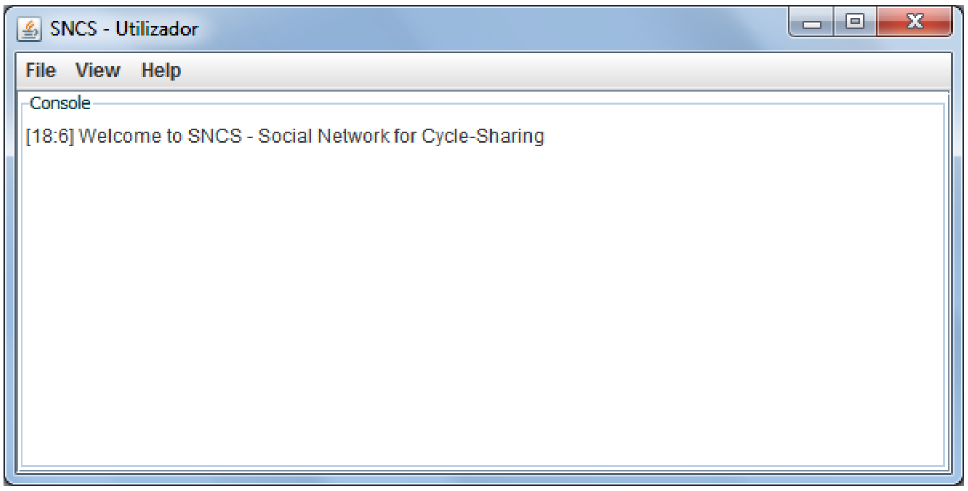
Moreover, Facebook gives the possibility for outside applications to authenticate a user by means of

[1]JSON: http://www.json.org accessed on 15/10/2010
[2]Facebook REST methods: http://developers.facebook.com/docs/reference/rest accessed on 15/10/2010
[3]Facebook Graph connections: http://developers.facebook.com/docs/reference/api accessed on 15/10/2010

their own Facebook Connect system, which is a Web page dedicated for the *Log in* process. Also, for the client application to gain access to Facebook pages, it has to be authorized by the users and given an *access token*, generated by the use of the *OAuth 2.0* protocol.[4] The authentication protocol is used only for Facebook applications that needs users' information to interact with Facebook, meaning the users' friends and groups. The users are also able to allow the Facebook applications to access their connections and objects (in Graph), by specifying which permissions[5] they can have, i.e. the *offline_access* permission enables a Facebook application to perform authorized requests on behalf of the user at any time, while also extending the tokens' expiration time.

For the purpose of displaying the Facebook Connect Web page, we make use of the *JDIC*[6] library. This enables us to display a Web site to the users in order for them to trust the Facebook application (in this case Tese_SNCS)[7] for the authentication process. The library gives us the ability to contact the Web site without any knowledge of how it is created (or in what language it was created), meaning that even if Facebook changes this Web site the client application can still acquire the necessary *access token*.

After a successful *Log in* process, Facebook returns a Web site (with the word "success" in its body) which SNCS can then parse its URL, that contains a set of parameters, including the *access token* and depending on its type, the tokens' expiration time.



[18:6] Welcome to SNCS - Social Network for Cycle-Sharing

Figure 4.1: SNCS Main Interface

As SNCS also needs to gather the information about the local resources of the users' computer, we make use of the *SIGAR* library.[8] This allows us to easily access a list of local resources each time it is called, such as CPUs, cores, memory. Also, it gives us the ability to know the current states of those resources, i.e. it can give us the available memory at the requesting time, or even the current idle time for each of the available cores or CPUs. This library is also useful for the fact that it can work in multiple environments, such as

[4]Facebook Authentication methods: `http://developers.facebook.com/docs/authentication` accessed on 27/08/2010
[5]Facebook Permissions: `http://developers.facebook.com/docs/authentication/permissions` accessed on 15/10/2010
[6]JDIC: `https://jdic.dev.java.net` accessed on 15/10/2010
[7]Tese_SNCS Application Facebook Page: `http://www.facebook.com/apps/application.php?id=123798840981469` accessed on 15/10/2010
[8]SIGAR library: `http://www.hyperic.com/products/sigar` accessed on 15/10/2010

Windows, Linux, among others, making possible the portability of SNCS to other systems. Meaning that it gives an Java object to operate on (gather the system information), while its components which are made for specific systems, are called from within the Java object.

4.2 Class Diagram

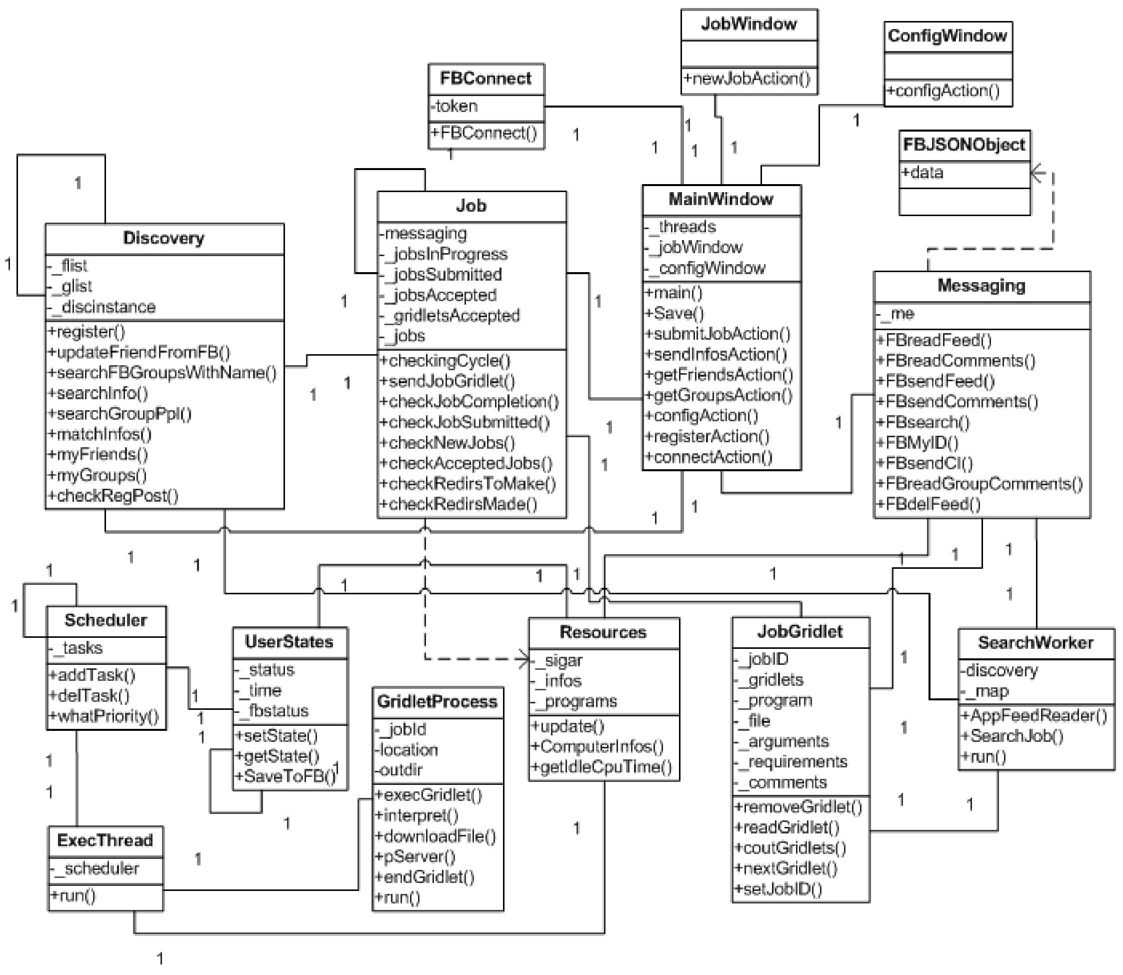


Figure 4.2: SNCS Class Diagram

The client application, as depicted in Fig. 4.2, has its main class named MainWindow, which uses the AWT library (the standard API for providing a graphical user interface in Java) to present the user with the GUI. Also, it is responsible to save information when the client application needs to exit. Furthermore, it contains the starting method for the Job submission, which begins the search for resources in Facebook and the completion of the Job.

This class needs the JobWindow and ConfigWindow classes, which display to the user windows in order for them to easily create a new Job or to configure some SNCS configurations. Also, it makes use of the FBConnect class, which displays the Facebook Connect Web site and retrieves the access token needed for future interactions.

Furthermore, it starts a thread that runs the "checking" cycle, that is located in the Job class, and the threads that are needed to execute the *Gridlets*.

The Job class contains the methods used to invoke the actions needed to complete a Job and to receive *Gridlets*. This class is singleton in order to have a one entry point to all the tasks that SNCS does. It makes use of the JobGridlet class which is the "Job" object created for a Job submission, meaning that it contains all the information for a Job (program, arguments, requirements among others).

The Discovery class has the responsibility to search for friends and groups, and when needed their computer information. It takes from the SearchWorker class the computers' information that it gathers from the Applications' Wall (FoFs method). Therefore, it is imperative that it remains the only instance available, meaning it is a singleton class.

The Messaging class contains the methods used to call Facebook functions (methods or objects), meaning that it makes use of the *RestFB* library to read and write from and to Facebook. This class is connected to the Java objects such as FBJSONObject, FBGroupComment, FBMember in order for the *RestFB* methods to map the incoming JSON objects to the corresponding Java objects.

The Resources class uses the SIGAR library to discover (and get) the information of the local resources, i.e. it has the methods used for getting the CPU idle times (in percentage) needed for other purposes (i.e. to determine when to stop the client application). Also, it is capable of gathering the computer information to send to Facebook. Meaning that, it creates a map of the resources and uses the CI class to map it into the Schemas used for Facebook, and vice versa.

The UserStates class gathers information from the resources and use it to assess if the client application should continue to run or stop its activities. Also, it reports its state to Facebook and it is a singleton class.

The Scheduler class is called to determine the *Gridlets* priorities, meaning that it contains the priority lists that the user configures in the ConfigWindow class, and also the list of the *Gridlets* that it has gathered to be processed. Also, this class is singleton in order to retain the lists for all the components that needs them. Moreover, the Job class, each time it gathers a *Gridlet* calls the Scheduler to queue them.

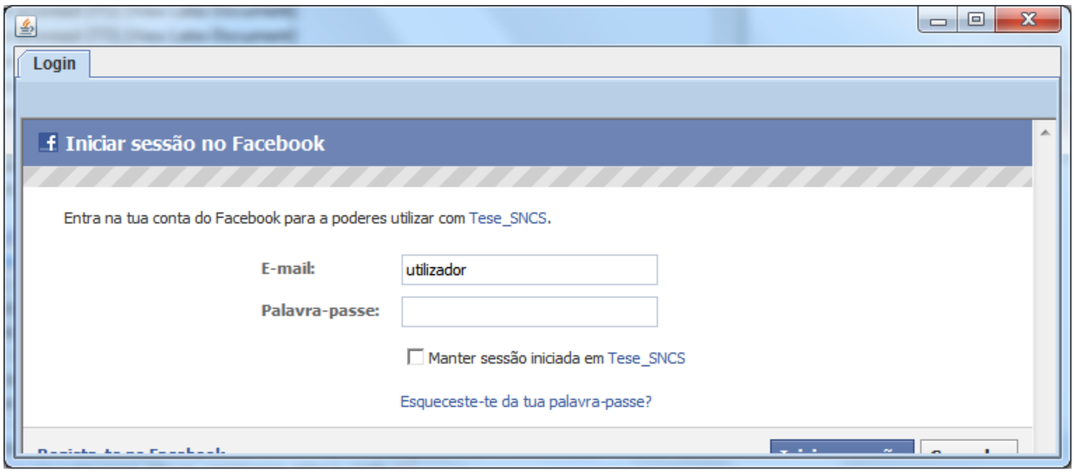
The ExecThread class is another thread that starts the processing of *Gridlets*, it calls the Scheduler class to obtain the next that should be processed.

The GridletProcess class contains a client and server methods for transferring the data files necessary for each *Gridlet* that uses the Application to Application transfer method. The class also has the methods to transfer the data files from a Web site. This class also interprets the arguments that are to be given to the processing program in order to know the files to be used, and to modify the paths to match the local ones. Moreover, it has the method to call other programs and wait for their results.

Furthermore, this class is responsible to finish the *Gridlet* process, meaning that it sends the completion message (calls the appropriate methods on the Job class) back to the originator of the *Gridlet*. Also, it determines if the completion message should contain the place where the originator can retrieve the resulting data, or if it should be an error message in order for the originator to be able to retry sending to someone else.

The Schemas for the messages are shared between the Job class and Messaging class, meaning that each one creates its own part of the message, i.e. sending a *Gridlet* message starts in a Job method (SendJobGridlet) that constructs a message with “JobID;GridletNumber;UserID;UserID;Program;File;Arguments;” and calls a method on messaging class which sends it to Facebook (which can be FBsendComment, FBsendGroupComment or FBsendFeed). This method then inserts into the message the head tag (SNCS) and the type tag (in this case JobGridlet), in order for the message, once read, be detected as from SNCS and what type of message it is.

4.3 User Interface



Iniciar sessão no Facebook

Entra na tua conta do Facebook para a poderes utilizar com Tese_SNCS.

E-mail: utilizador

Palavra-passe:

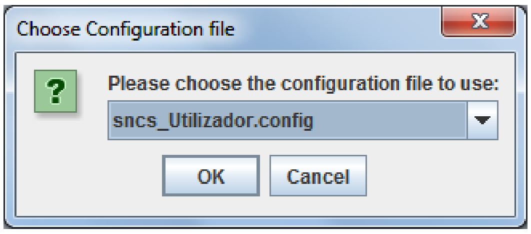
☐ Manter sessão iniciada em Tese_SNCS

Esqueceste-te da tua palavra-passe?

Figure 4.3: SNCS Log In Interface

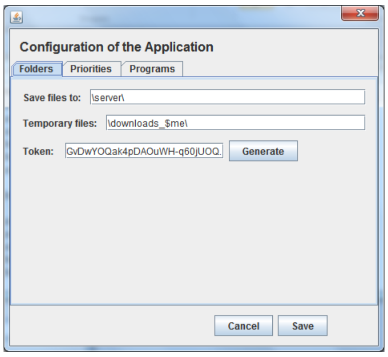
Social Networks for Cycle-Sharing has been designed to provide a simple Graphical User Interface (GUI) (Fig. 4.1) in order for any user to utilize it without much burden. For SNCS to function correctly a user needs to have an account on Facebook, and *Log in* into it via the client application, as depicted in Fig. 4.3, a user only needs to write his/her own account name and password on the screen presented and Facebook confirms or denies access, while on the background the client application gets the *access token* needed for future communications. As an add-on to SNCS after a user successfully *Logs in*, it has the opportunity to save his/hers *access token* for future interactions with the client application and skipping the *Log in* Web site (Fig. 4.4), although the user may have to generate a new *access token* after the last one expires.

The client application is also able to configure some of the aspects needed to better suit the users. As depicted in configuration images (Fig. 4.5), the user may be able to save the *access token* as explained above in the Folders tab (Fig. 4.5(a)), to prioritize the incoming *Gridlets* by specifying which friends have higher or low preferences, as depicted in the Priorities Tab (Fig. 4.5(b)), also the user may add other users that it wishes to give priorities. As a concern to the *Gridlets* priorities and for experimental results, no user can be blocked from executing his/her *Gridlets*, although this feature should be included as a precaution



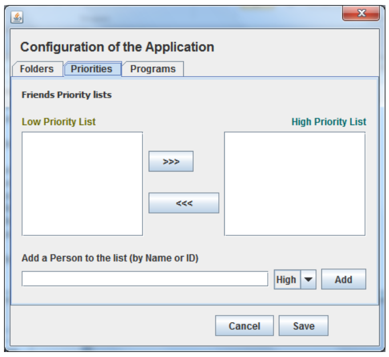
Please choose the configuration file to use: sncs_Utilizador.config

Figure 4.4: SNCS Choose Interface

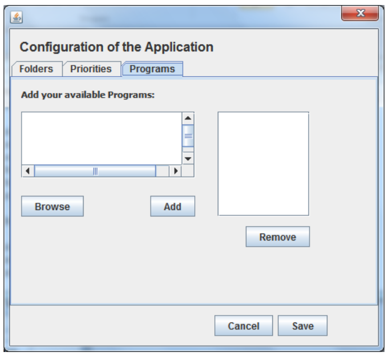


Save files to: \server\
Temporary files: \downloads_$me\
Token: GvDwYOQak4pDAOuWH-q60jUOQ

(a) SNCS Folder Tab Interface



(b) SNCS Priorities Tab Interface


Configuration of the Application
Folders
Priorities
Programs
Add your available Programs:
Browse
Add
Remove
Cancel
Save

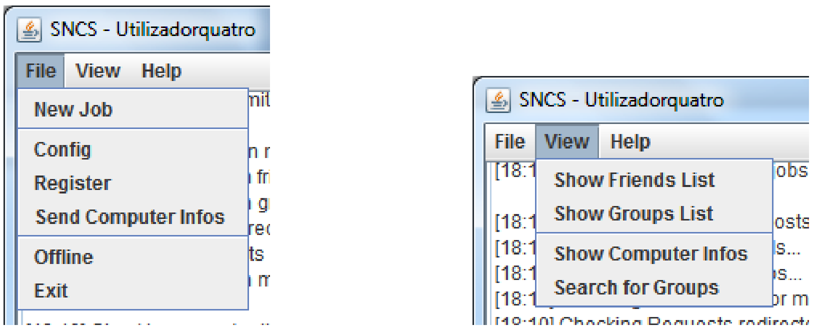
(c) SNCS Programs Tab Interface

Figure 4.5: SNCS Configuration Interface

for the user's abuse of the network. In the Programs tab (Fig. 4.5(c)) the user has the ability to manage its own programs that can be executed on the user's side, this feature allows SNCS to reject Jobs that cannot be executed on the users' computer.

After the *Log in* process, users have access to the main interface (Fig. 4.1), where they can choose from the menus interface (Fig. 4.6) some options, such as creating a new Job, registering on the Applications' Wall, sending their computers' informations to the users' Wall, or even to stop the client application from processing messages and *Gridlets* by pressing the *Offline* menu option.

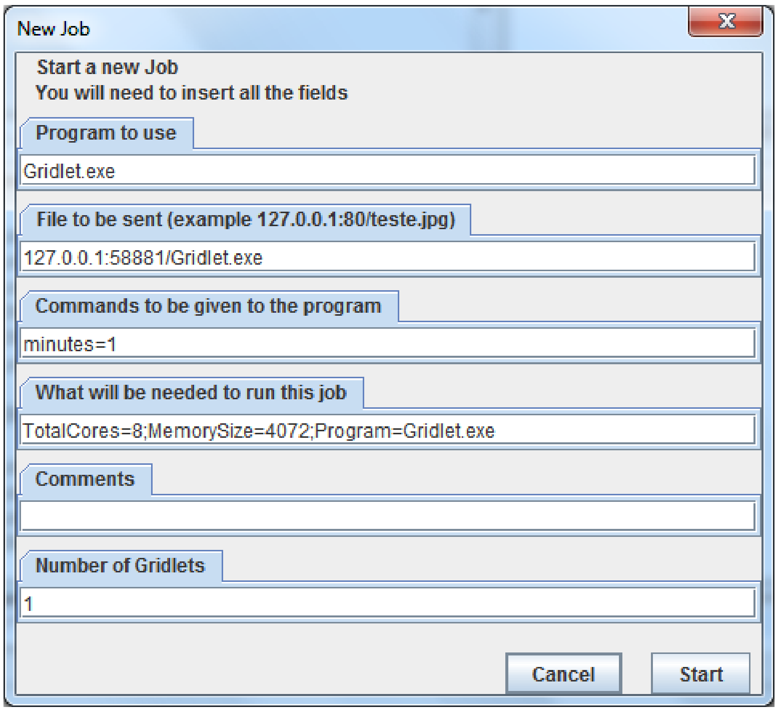
The new Job interface, depicted in Fig. 4.7, allows a user to easily start a new Job, by inserting all the necessary information in the fields presented. Afterwards, SNCS does the necessary steps to ensure the



(a) File Menu **(b)** View Menu

Figure 4.6: SNCS Menus Interface

completion of the Job.


New Job

Start a new Job
You will need to insert all the fields

Program to use
Gridlet.exe

File to be sent (example 127.0.0.1:80/teste.jpg)
127.0.0.1:58881/Gridlet.exe

Commands to be given to the program
minutes=1

What will be needed to run this job
TotalCores=8;MemorySize=4072;Program=Gridlet.exe

Comments

Number of Gridlets
1

Cancel Start

Figure 4.7: SNCS New Job Interface

4.4 Data structures and Message Schemas

The client application makes use of some internal data structures, that have the responsibility to hold the information required to interact with other client applications, and the Schemas that allow simple reading and understanding of the messages that can appear on the users' Wall. Thus, we have constructed the following data structures.

- **JobGridlet**: A data structure that contains the information submitted by the user, to start a Job. For this particular data structure, we do not generate any identification, it uses the unique identifier for the *Post* made in the users' Wall (which is preserved until the Job has been completed). And thus, if the client application is not successful in sending this message, the Job is aborted. Moreover, this structure contains the number of total *Gridlets* that have to be processed, and those that have already been completed so far.

- **User**: The user data structure is used to identify any type of user, meaning that it can be the user itself or a friend. It has the necessary information such as UID of the user or name.
- **Group**: This data structure is similar to the users, because the Graph API sends the users and groups data as being objects of the "same" type, and thus having the same (or almost) connections (or functions). For the client application, this structure contains the users for the particular group and its UID.
- **Computer Information**: The data structure comprising computer information, contains a subset of the systems' information, meaning that only the relevant data is saved. Such data can contain the number of processors, memory size and programs that can be used on the computer side to process *Gridlets*.
- **Priority Lists**: A data structure that is used to prioritize the *Gridlets*, such lists are only necessary when the users configure them. Although, these lists should be always generated in order to block unwelcome *Gridlets*, that could appear (making a Security risk).
- **Programs List**: A data structure that is used to contain all the programs that the user grants permission to be used on the computer. This list contains the name of the program and the location where the executable resides, in order for SNCS to run it when a *Gridlet* needs to be processed by the specified program.
- **Internal Jobs Containers**: These data structures serves the purpose of saving the tasks that are currently being executed, such as Jobs submitted, *Gridlets* in progress (on the requesting side), Jobs accepted (Jobs that have been accepted and are waiting for *Gridlets*) and *Gridlets* accepted (on the processing side). These data structures are internal to the client application and are used to monitor the tasks sent/received to/from other users using the Social network.

In SNCS we make use of the *RestFB* library, which gives us the flexibility of contacting Facebook without knowing JSON objects are being sent. Also, the library gives us a generic Java object that it uses to map the JSON objects to it. However, some Facebook objects cannot be mapped to a generic Java object, which requires us to create Java objects compatible with the JSON objects, in order to acquire the information. These Java objects are described as follows.

- *FBGroupComment*: is used to be mapped by the Comments on Posts that belongs to Groups. It is an extension from the generic Comment object (from *RestFB*), that contains the correct syntax used on the messages, such as "text" instead of "message", and "fromid" instead of "from".

```
//Search for someone to do a job..
SNCS;Job;MyId;<hardware/software required>;<Comments>

//Answer to a job / JobGridlet
SNCS;<Job/JobGridlet>;MyId;<Accepted/Denied/Completed>;<JobId>
//Special versions only on comments (App Wall Post) adds a JobId

//Give a GridletJob to someone
SNCS;JobGridlet;JobId;GridletNumberX;MyId;YourId;<Program>;<File>;<Commands
needed to execute>;<Other comments needed>;
//The file should be the place where the client application can download it,
//MyId and YourId should be here so that if someone else reads this
//they should skip it if its not for them, and security reasons

//GridletJob Status Update (completed)
SNCS;JobGridlet;MyId;Completed;<JobId>;<GridletNumber>;<where to download>;

//Redirect Messages (For the user to redirect to its friends)
SNCS;Redir;<PersonId to redirect to>;<PostId>;<Type>;<rest of the message>;

//Redirect Request (to be made in RedirToID)
SNCS;Redir;Request;<RedirToID>;<Type>;<UserId that started the redirection>;
//the last is optional

//Message Parts
SNCS;PartX-Y;<UserID>;<Type>;<Rest of the message>;
//X is the number of the part, Y is the total number of parts, UserID must be
placed because Facebook sometimes //forgets from whom the message belongs
```

Figure 4.8: Schemas for messages in Facebook

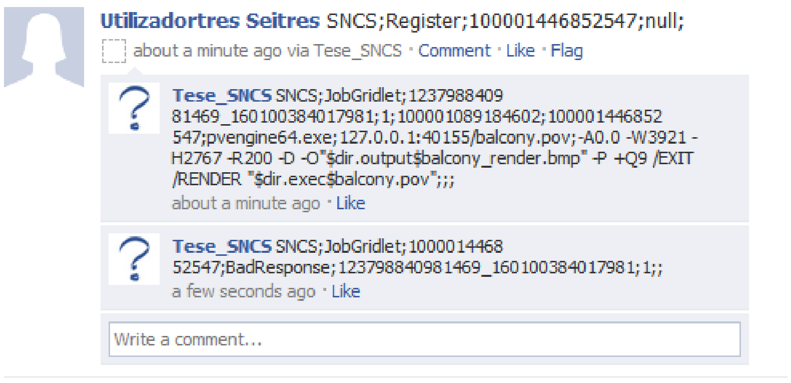
- *FBisFriend*: is used to be mapped when invoking a FQL[9] method, which is a way to directly ask for data to the databases. Note that these methods (from FQL) are predefined and not always reliable.
- *FBJsonObject*: is used to be mapped by a list of other objects, such as the friends' list or groups' list.
- *FBMember*: is used to be mapped by users belonging to a group, which have their own information. This happens because people in a group may not be friends of the user, meaning that some information about those users may not be available to the requesting user.

For the communication between the client applications using Facebook, we use our own Schemas. Much because Facebook does not allow some types of message schemas, such as XML based.

These Schemas make use of an ordinary separator of *Strings*, portrayed in Fig. 4.8. Regarding messages that can be longer than the limit imposed by Facebook, such as the Computer information messages (Fig. 3.5), they are split into various messages and an indicator of more messages alike is inserted in the schema ("PartX-Y"), which is read by the client application, informing it is not the only part that has to be fetched, and it needs to fetch Y messages. Also, in the new updates to Facebook API, the comments appear to have no originating user (thus having the name of the Facebook Application as depicted in Fig. 4.9), and as such we introduced a UserID in the schema.

These Schemas are very simple and human readable, in order for Facebook to allow them on their Web site, and not consider them as "Spam" or other type of blocked messages. Although, it is possible that in

[9]FQL: `http://developers.facebook.com/docs/reference/fql` accessed on 15/10/2010



Utilizadortres Seitres SNCS;Register;100001446852547;null;
about a minute ago via Tese_SNCS · Comment · Like · Flag

Tese_SNCS SNCS;JobGridlet;1237988409 81469_160100384017981;1;100001089184602;100001446852 547;pvengine64.exe;127.0.0.1:40155/balcony.pov;-A0.0 -W3921 -H2767 -R200 -D -O"$dir.output$balcony_render.bmp" -P +Q9 /EXIT /RENDER "$dir.exec$balcony.pov";;;
about a minute ago · Like

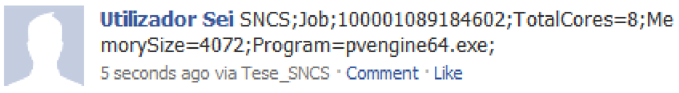
Tese_SNCS SNCS;JobGridlet;1000014468 52547;BadResponse;1237988409 81469_160100384017981;1;;
a few seconds ago · Like

Write a comment...

Figure 4.9: Completion message example in Applications' Wall (completed with error)

the future such messages might not be supported. Also, they are human readable to assure the users what information is being sent to other users.

Fig. 4.9 depicts the moment when a user sent a *Gridlet* message to the user that was going to process it, and an error occurred. This means that the originator can delete the *Gridlet* message from Facebook and retry to send the same *Gridlet* to another user (or possibly the same user).



Utilizador Sei SNCS;Job;100001089184602;TotalCores=8;MemorySize=4072;Program=pvengine64.exe;
5 seconds ago via Tese_SNCS · Comment · Like

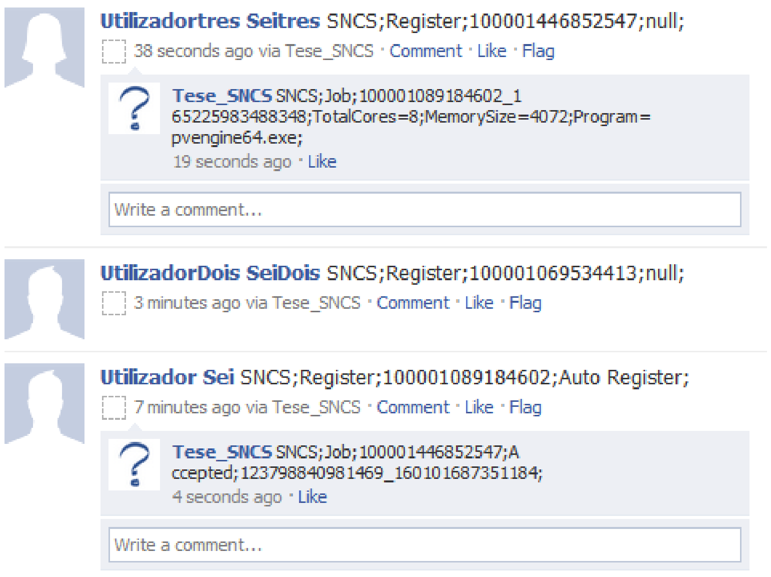
Figure 4.10: Job Search message on Users' Wall

The Schemas represented in Fig. 4.10 and Fig. 4.11 gives us the idea of how it appears to the users in their Walls and in the registration Post in the Applications' Wall (FoFs method) respectively. Moreover, these messages are comprised of the Jobs' requirements and the JobID in case of the Applications' Wall (which contains the UserID). Also, in Fig. 4.11 we can see that the user has responded to the Job Search with an accept message.

Fig. 4.12 represents a *Gridlet* message sent to a user containing the location of the file "balcony.pov" that the processing client application needs to *download*. Also, we can see that the *Gridlet* number is 3 (depicted in a red circle), meaning that it is the third *Gridlet* of the Job in question. Moreover, it displays the program and the arguments given to it to process the data file.

Fig. 4.13 depicts the message sent to a users' friend, in order for him to redirect to its own friends. This message contains the ID for which the FoFs has to send the message, and the type of message that it sends is also described (in this case "cpuInfos", meaning the FoFs computers' information).

SNCS uses Facebook to send and retrieve messages via the Facebook API. Meaning that, it reads Posts (messages that are contained in the users' Wall, groups' Wall) and Comments (messages contained within the Posts), and writes other messages on users' Wall (which is a space that contains messages) either as Posts or Comments.


Utilizadortres Seitres SNCS;Register;100001446852547;null;
38 seconds ago via Tese_SNCS · Comment · Like · Flag

Tese_SNCS SNCS;Job;100001089184602_1
65225983488348;TotalCores=8;MemorySize=4072;Program=
pvengine64.exe;
19 seconds ago · Like

Write a comment...

UtilizadorDois SeiDois SNCS;Register;100001069534413;null;
3 minutes ago via Tese_SNCS · Comment · Like · Flag

Utilizador Sei SNCS;Register;100001089184602;Auto Register;
7 minutes ago via Tese_SNCS · Comment · Like · Flag

Tese_SNCS SNCS;Job;100001446852547;A
ccepted;123798840981469_160101687351184;
4 seconds ago · Like

Write a comment...

Figure 4.11: Job Search and Acceptance messages in Applications' Wall

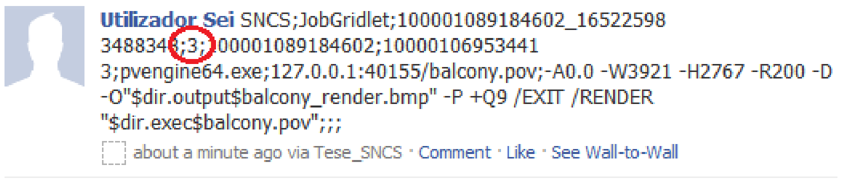

Utilizador Sei SNCS;JobGridlet;100001089184602_16522598
3488348;3;100001089184602;10000106953441
3;pvengine64.exe;127.0.0.1:40155/balcony.pov;-A0.0 -W3921 -H2767 -R200 -D
-O"$dir.output$balcony_render.bmp" -P +Q9 /EXIT /RENDER
"$dir.exec$balcony.pov";;;
about a minute ago via Tese_SNCS · Comment · Like · See Wall-to-Wall

Figure 4.12: Gridlet Message

Posts can only be used between users that are considered friends or in known groups, and therefore the people and groups that the user cannot directly contact, do so via the Applications' Wall by commenting on users' Posts (Registration Post). This Post is either created by the users or it can be created automatically by the client application when it needs to reach FoFs. This method is used to bypass the inability of contacting other people rather than just direct friends. This does not mean that other Social networks have the same restrictions and therefore the operations of registering and redirecting messages should be optional on SNCS.

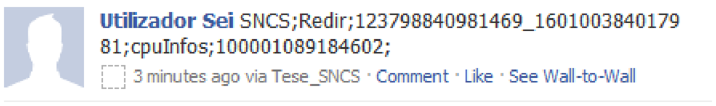

Utilizador Sei SNCS;Redir;123798840981469_1601003840179
81;cpuInfos;100001089184602;
3 minutes ago via Tese_SNCS · Comment · Like · See Wall-to-Wall

Figure 4.13: Message for the user to redirect to friends

4.5 SNCS Configuration

Prior to the execution of the client application there are some configurations that the user needs to make, and some of them cannot be optional. The main condition to use SNCS is that the user has to be registered in Facebook, or the Social network that is being used, meaning that the user needs to create a Facebook account[10].

As Facebook does not allow users to write Posts or Comments on the Applications' Wall as is, the user must "Like" the Facebook application, meaning that the user has to go to the Application Facebook page (in this case Tese_SNCS)[11] and click on the "Like" button that appears for them, in order to enable sending messages to the Applications' Wall. Afterwards, the user can "register" on the Applications' Wall, meaning that the client application sends a message to that Wall in order for the redirected messages (that were sent by others) to reach the user (in case of interactions with FoFs).

The use of "sending your computer informations" is optional, although it should be used as a part of the configuration. This happens because the client application tries to search for computer's information on the users' Walls instead of asking for them at first, meaning that it is a added overhead and latency if the client application needs to request it explicitly from all users. Thus, if the user has already sent its computer information to Facebook, the client application can quickly gather them.

The priority lists and programs lists can be specified in the Configuration Menu and are also optional to the users, however these options should not be ignored in order for a smoother usage of SNCS and a more efficient resource and application discovery.

Furthermore, the use of these lists are recommended in order for SNCS to gain perspective on what type of Jobs (depending on the programs the users allow) it can donate the computers' idle cycles and also the way *Gridlets* are executed, because the user may want to help processing them for the people that most helps the user.

4.6 SNCS Constraints

The decision of using Facebook as the Social network for interactions between people, has brought some constraints due to the limitations that Facebook enforces, either with the Use Terms[12] or their API.[13]

In order to interact between users, the client application normally uses the Posts method, which can not be guaranteed between users that are not friends. As such, we use the method of redirecting messages, by sending it to a friends' Wall, so that the users' client application can direct it to the proper Wall, meaning its friends (the FoFs method). In order to generate fewer "spam" the last client application does not redirect messages to users that are already friends of the first. Furthermore, as the route of messaging can be expanded "indefinitely" we only consider sending messages until FoFs.

In the case of sending messages, Facebook has limited the size of the messages that can be sent by outside applications (in the order of 420 characters), and the method used to circumvent it was to split messages in smaller ones, making the client application verify from all the Posts their message type and from whom they belong to.

As it as been said before, the Graph protocol is still in development, and as such some functionalities

[10]Facebook: `http://facebook.com` accessed on 15/10/2010
[11]Tese_SNCS page: `http://www.facebook.com/apps/application.php?id=123798840981469` accessed on 15/10/2010
[12]Facebook Use Terms: `http://www.facebook.com/terms.php` access on 26/08/2010
[13]Facebook Graph API: `http://developers.facebook.com/docs/api` accessed on 15/10/2010

do not work as expected, therefore the reading and writing of Comments on groups' Walls is dealt with the *REST* protocol, which should be changed when Facebook decides to implement a better way.

The most important constraint is that Facebook also limits the number of requests that can be sent by the client application each day per user.

This limit can be changed by Facebook, and is based on the affinity users show for the Facebook application's use of Facebook Platform through their interactions,[14] also *"values will change over time depending on how users interact with your application"*. Therefore Facebook created a system of Buckets, that have a limit number of requests, i.e. Tese_SNCS (name of the client application on Facebook) is on 9 of 14 Buckets which has 26 requests per user per day. As the current method to overcome this limitation, SNCS should encourage users to only have 1 Job per day. However, we cannot consider that every Social network has the same limits, and therefore a specialized limitation would have to be reviewed for each case.

[14]Facebook Allocations: `http://www.facebook.com/business/insights/app.php?id=123798840981469&tab=allocations` accessed on 27/08/2010 (can only be accessed by Facebook applications' Administrators)

5 Evaluation

Contents

In this chapter we present the evaluation of SNCS regarding its performance, stability and viability for using a Social network to achieve public-resource sharing. Our focus is to know the achievement of resource and service discovery, by recruiting as many users as possible to execute *Gridlets*. While also, integrating with the normal usage of the Social network, which in the user's point of view would be the amount of information perceived in the Social network, which should be kept minimal. Finally, SNCS should provide idle cycles to be used to process the upcoming *Gridlets*, as such the client application was tested in a "more realistic" environment as described in this chapter.

In order to perform all the tests we constructed several scenarios, where the environment for each would change as follows.

In the first scenario, to make sure that two friends can achieve public-resource sharing with each other and as the main task for the client application, we have 1 *Gridlet* to be completed with 1 or 5 minutes of processing time. As this is the starting point of our tests, it was imperative that the client application, which would execute the *Gridlet*, had available resources.

In the following scenarios we changed the number of users involved and also their roles. Beginning by adding a new friend and increasing the number of *Gridlets*. Afterwards we changed the role of the friend to be a Friend of Friend (FoF), so we can expand the number of users that the Job can reach. As these scenarios are not sufficient for a large scale test, we replaced the FoF with a user in a group. And consequently introduced a FoF in the scenario, while also adding a friend to the group, in order to increase the number of *Gridlets*.

In scenario six, as depicted in Fig. 5.11, we considered 2 friends, 2 FoF and a group with 3 users, where one of them was a FoF, also we increased the number of *Gridlets* to 7 and the processing time to 5 minutes only. In scenario 7, as depicted in Fig. 5.14, is considered as a "more realistic" scenario, were there are Friends, FoF and other people connected by a Group, while also increasing the number of Jobs to two, and the total number of *Gridlets* to 15, with the same processing time as before.

For all the scenarios above, we assumed that the number of *Gridlets* were suitable to complete the Job in question and that they would take exactly the time spent (1 or 5 minutes), however realistic processing times for *Gridlets* can take much more (in hours) as it happens with other cycle-sharing systems, such as SETI@Home; that all users would have their client applications running prior to the start of any Job, meaning that they would be in any part of the "checking" cycle (as referred in the Jobs Manager module description); that every user could only execute 1 *Gridlet* at a time; and also that all computers would have the capabilities for processing the *Gridlets*, meaning that the search would match their resources exactly and therefore the resources would be considered as the ones needed. Also, for the purpose of simulating the processing time of each *Gridlet*, we made use of a timed count down program for each one and that each client application would have to *download* it from a Web site.

For the last scenario, we considered to execute *Gridlets* in a real program (Pov-Ray) to render a image, to understand exactly the consequences of the added overheads of SNCS to the overall process.

5.1 Scenarios 1 and 2

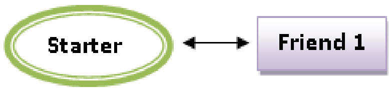


Figure 5.1: Scenario 1 View

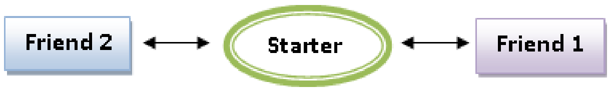
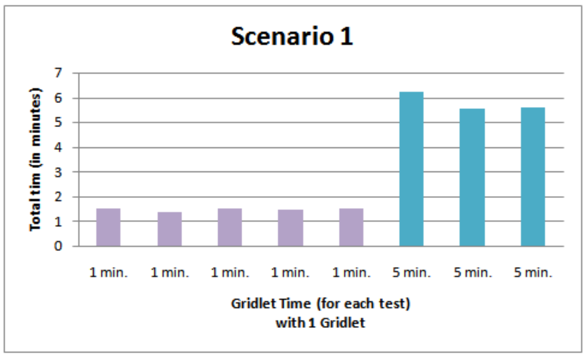


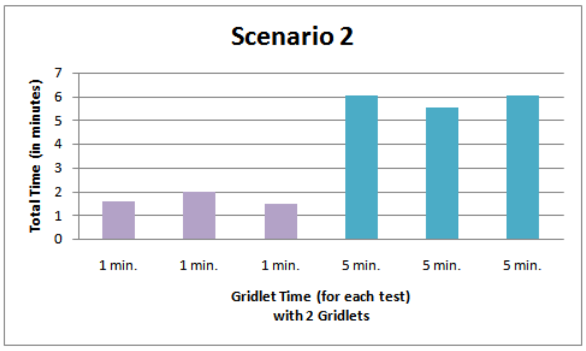
Figure 5.2: Scenario 2 View

Scenarios one and two were simple tests to make sure that we could achieve our works' goals and are described as follows.

In Scenario 1, as depicted in Fig. 5.1, we have two users connected as Friends and one of them has to process 1 *Gridlet* that has 1 or 5 minutes of processing time in each test. Moreover, the Friend that processes the *Gridlet* has the necessary resources available that it requires to be executed.



(a) Total times for Scenario 4



(b) Total times for Scenario 5

Figure 5.3: Total times for Scenario 1 and 2

In scenario 2, as depicted in Fig. 5.2, we augment the tests with a new Friend connection and with two

Gridlets to be completed by the Friends.

In both scenarios we assume that each client application has to *download* a timed count down program to be executed in the processing client application and that the client application would be running prior to the start of the tests, meaning that it could be in any given point of the "checking" cycle.

The messages (as seen in Fig. 4.8) that are sent and retrieved by the client applications are made using only the starter's Wall and therefore the use of other Walls is not necessary for these tests.

The *Gridlet* message contains the place where the client application should *download* the data files and also the arguments that are used to execute the processing program on the user's computer. In these cases the arguments given are "minutes=1" or "minutes=5". Also, as it was expected, the requirements of the *Gridlets* would match the computers' information, in this case the requirements are "TotalCores=8; MemorySize=4072; Program=Gridlet.exe", meaning that it needs 8 CPUs, 4 GB of memory and the program Gridlet.exe.

5.1.1 Results

The results for scenarios 1 and 2, as depicted in Fig. 5.3, can lead us to believe that using a Social network to discover resources and services can be achieved.

When considering the tests where the *Gridlet* takes 1 minute, we can see that the overhead of the client application (only considering friends), is around 50 seconds, these times includes sending and retrieving messages from the Social network, meaning that the Social network usage is accounted for. Note that Facebook introduces variable latency in Wall update that SNCS cannot circumvent.

Moreover, when we increased the processing times to 5 minutes the overheads remained around the same, which can lead us to a speedup in larger scale environments.

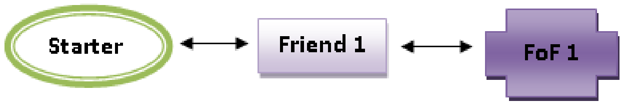


Figure 5.4: Scenario 3 View

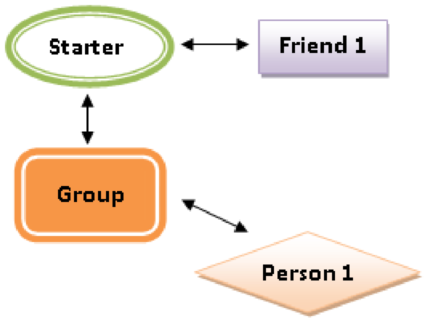
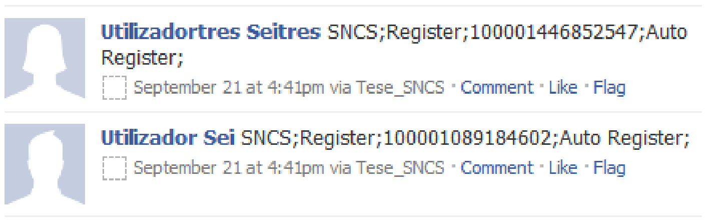


Figure 5.5: Scenario 4 View

5.2 Scenarios 3 and 4

In the following scenarios we expand the range of communication by adding a Friend of Friend (FoF) in scenario 3 (Fig. 5.4) and by adding a person in a group (without counting the starter) in scenario 4 (Fig. 5.5).

In order to reach as many users as possible to evaluate larger networks we need to guarantee that we can include these types of users and that the SNCS overheads introduced do not affect the overall performance.

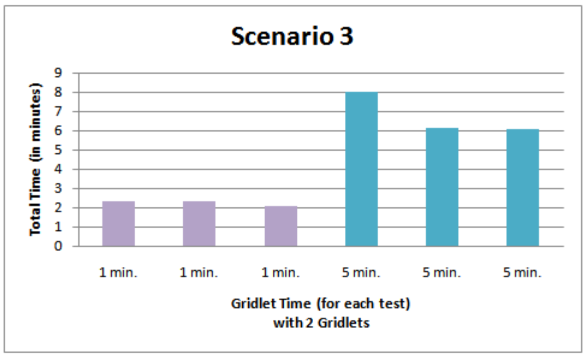


Utilizadortres Seitres SNCS;Register;100001446852547;Auto Register;
September 21 at 4:41pm via Tese_SNCS · Comment · Like · Flag

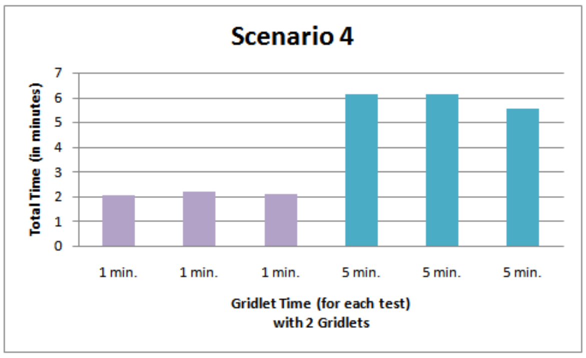
Utilizador Sei SNCS;Register;100001089184602;Auto Register;
September 21 at 4:41pm via Tese_SNCS · Comment · Like · Flag

Figure 5.6: View of the Registration Post

In case of scenario 3, the Applications' Wall is needed for the communication between FoF and the starter user, meaning that both users have "registered" their client applications and a Post was created in the Applications' Wall containing their UserIDs, as depicted in Fig. 5.6



(a) Total times for Scenario 3



(b) Total times for Scenario 4

Figure 5.7: Total times for Scenario 3 and 4

For both scenarios we can assume that each user has the resources to process the *Gridlets*, that each has their client application running prior to the start of the Job and that they would *download* the timed count down program from a Web site. Moreover, the two *Gridlets* and their consuming times are the same for these tests.

The messages for these tests are also sent and retrieved through the starter' Wall, however a Job request is also sent to the starter' group Wall in order for the group users to be able to retrieve it. And also as discussed in previous chapters, the communication between the starter and FoF uses the Applications' Wall, and thus the messages sent there additionally contains the UserID and the JobID, also described in the Schemas (Fig. 4.8). Moreover, the *Gridlet* messages are of the same type as before.

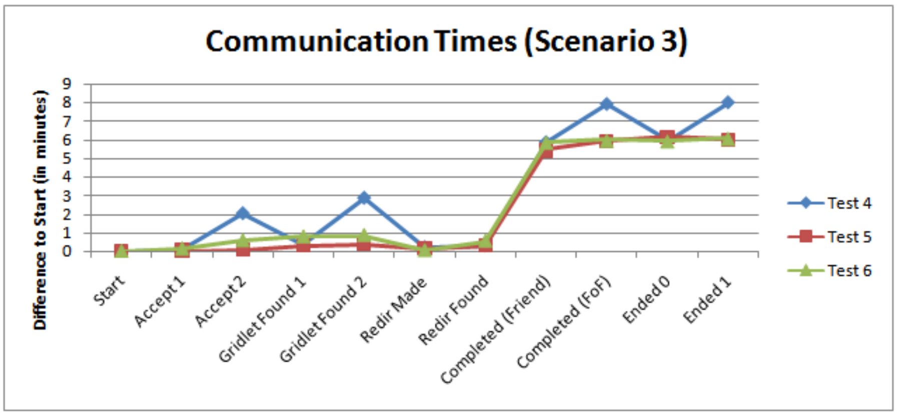


Figure 5.8: Communication Times for Scenario 3

5.2.1 Results

The results for scenarios 3 and 4, as depicted in Fig. 5.7, changes the overheads of SNCS. Because we introduced either a FoF (in scenario 3) or a group (in scenario 4) and it is explained by the fact that the messages are sent to other Walls instead of just only the starter's Wall.

In the case of FoF, we must consider the fact that the redirected message takes a longer time than sending a message directly. Although, in our tests the added time is still less than the processing time for a *Gridlet* and in case of test 4 in scenario 3 (first with 5 minutes), the time to discover the *Gridlet* message suggests that the Social network had an increased traffic load, making the FoF wait a longer time to retrieve it, as depicted in Fig. 5.8. This figure represents the time between events and its difference to the start of the process, meaning that each event marked, such as Accept 1 (accepting a Job), Completed (FoF) (*Gridlet* completed by FoF) happened some minutes after the Job Search started for each test represented.

5.3 Scenario 5

In scenario 5, as depicted in Fig. 5.9, we merge the previous scenarios and begin to construct a larger network. By having a Friend, a FoF and two persons in a group (without counting the starter), where one of

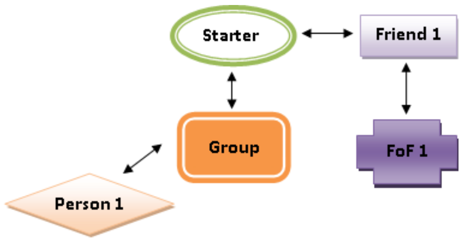


Figure 5.9: Scenario 5 View

them is the Friend. While also increasing the number of *Gridlets* to four and still maintaining the processing times of 1 and 5 minutes.

For these tests we assume that each of the participants of the group have already been accepted to the group, making it possible for the client applications to send and retrieve Posts and Comments from it. We also assume that SNCS has "registered" the users in the Applications' Wall and is already executing in each user's computer.

The messages for the Job request are sent to the starter's Wall, to the group Wall and upon receiving the FoFs' computer information it begins the Job request using the Applications' Wall. Moreover, the *Gridlets* do not change in these tests, and therefore the previous description of them still applies.

5.3.1 Results

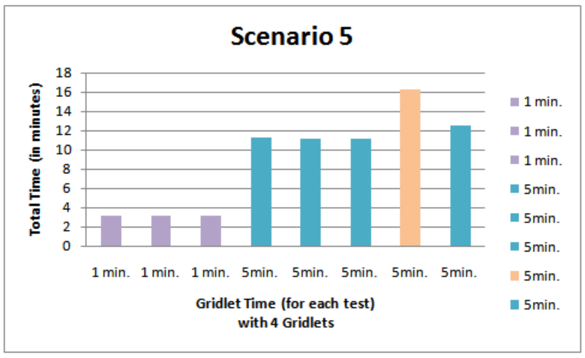


Figure 5.10: Total times for Scenario 5

The results for scenario 5, as depicted in Fig. 5.10, are encouraging us to pursue the idea of each user to process a *Gridlet* at a time. Although in Test 7 and 8 (the last two tests in Fig. 5.10), we see an increase in the Social network traffic load, which increases our total times and hinders the speedup that we could achieve. Moreover, we can say that using both methods (FoFs and Group messaging) becomes acceptable with longer running *Gridlet* executions.

5.4 Scenario 6

Scenario 6 bring us a larger view of a Social network, where the starter user is connected to two Friends, which are connected to one FoF each and a group with three people, where one of them is a FoF (without

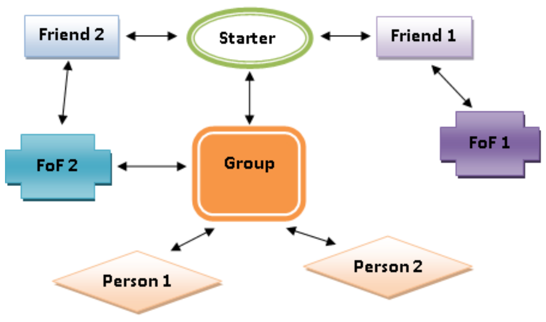


Figure 5.11: Scenario 6 View

counting the starter). Also the number of *Gridlets* is increased to seven and their processing times are of 5 minutes only.

For this scenario we assume that the client applications has "registered" in the Applications' Wall, that each of the group members have already accepted their membership and that the client application is already running in the users' computers. Moreover, we assume that the time to process a realistic *Gridlet* can be more then 5 minutes and therefore the time spent is sufficient to determine the viability of using the Social network to achieve our works' goals and also the processing time of a *Gridlet* data does not change the inherited overheads of SNCS.

The messages for this scenario continues to be of the same type as before, although the requirements of the *Gridlet* are changed to accommodate the computers that were used. In this case the requirements are changed to "TotalCores=2;MemorySize=4078;Program=Gridlet.exe" and each SNCS that processes the *Gridlet* is able to accept its conditions (however they still need to assess if they have idle cycles to spare).

5.4.1 Results

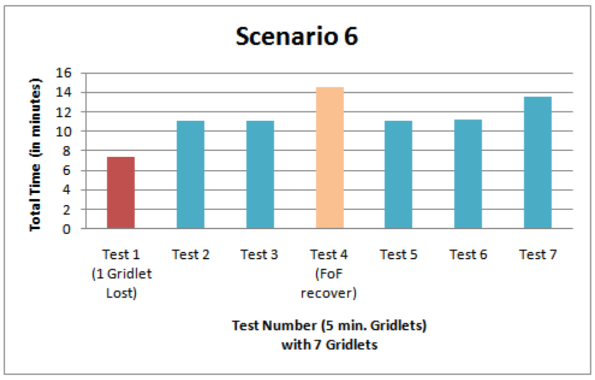


Figure 5.12: Total times for Scenario 6

The resulting times from this scenario, as depicted in Fig. 5.12, are consistent with the previous ones. Meaning that in each test, the times to complete a Job were in the order of 11 minutes. Although, in Test 1 the FoF2 did not receive the last *Gridlet* as it was supposed to, and in Test 4 the FoF2 crashed and recovered the last *Gridlet* in time to complete it. These situations proved that the total times, will be hindered by the fact that people are not always in a *Online* state and also by giving more than one *Gridlet* to the same user the Job will have longer total times. However, we cannot always expect to find as many users as the number of *Gridlets* needed to complete a Job.

We can also see that the overhead of SNCS is minimal considering the processing time of the *Gridlets*, which makes it possible to have speedups on data processing. However, we cannot estimate the exact added time of the Social network usage, meaning that these times can vary with the Social networks' traffic load at the time of use.

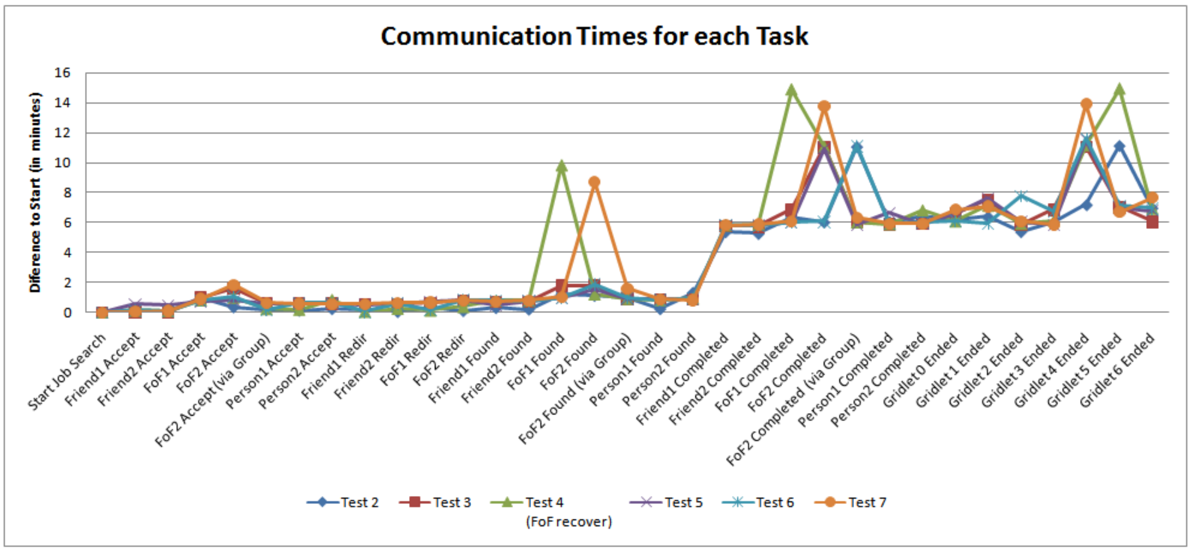


Figure 5.13: Communication Times for Scenario 6

Fig. 5.13 explains in detail how much time each task takes in relation with the starting point, i.e. it can take less than 1 minute for users' client applications to find and accept new Jobs, and that the higher spikes are caused by the fact that the client application only found the *Gridlet* some minutes later due to its *Offline* state and between the found tasks and completed tasks (for each user) we can see the processing time of the *Gridlet* (5 minutes).

5.5 Scenario 7

Scenario 7 is an attempt to test SNCS in a more realistic environment, having a more complex network of users. As depicted in Fig. 5.14, we have two users who start a Job (User 1 and 6), where User 1 has three Friends (User 2, 7 and 15), User 6 has two Friends (User 8 and 15). User 2 and 15 have each a FoF not connected to anyone else. Also, we created a group with six people (User 1, 2, 4, 5, 6 and 7). The layout of this network is made in a attempt to maximize the diversity of the users' roles, making it possible for a Job request to reach any of the users.

In this scenario, User 1 and 6 start a Job each, that contain 8 and 7 *Gridlets* respectively, making a total of 15 *Gridlets* to be processed by any of the users in this network. Furthermore, the client application does not restrain itself to gather only one *Gridlet* for each Job, however it only accepts a Job request per user for

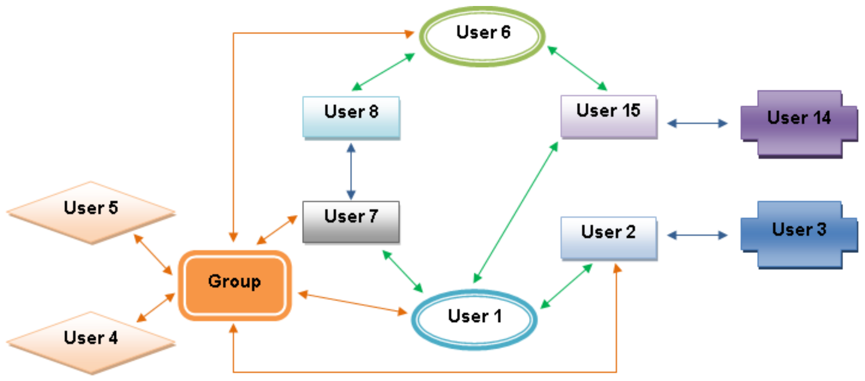


Figure 5.14: Scenario 7 View

each "channel" (Group, Wall, Applications' Wall) that the Job request appears in, i.e. User 7 can accept Jobs from the Group it is connected to, from its friend (User 1), and its friend (User 8), where the latter connection is of FoF to User 6, meaning that (in this network) it could acquire four *Gridlets*.

For this scenario we assume that each Job has less *Gridlets* than users that can be connected to the starter user, in order to simulate a larger network where the user could have potentially hundreds of connections, that could either accept or deny the request. Also, each client application would already be "registered" in the Applications' Wall, the group members already established and the client application would be running prior to the Jobs submissions. Furthermore, the two Jobs are started roughly around the same time in order for the client applications to retrieve the *Gridlets* in any given order.

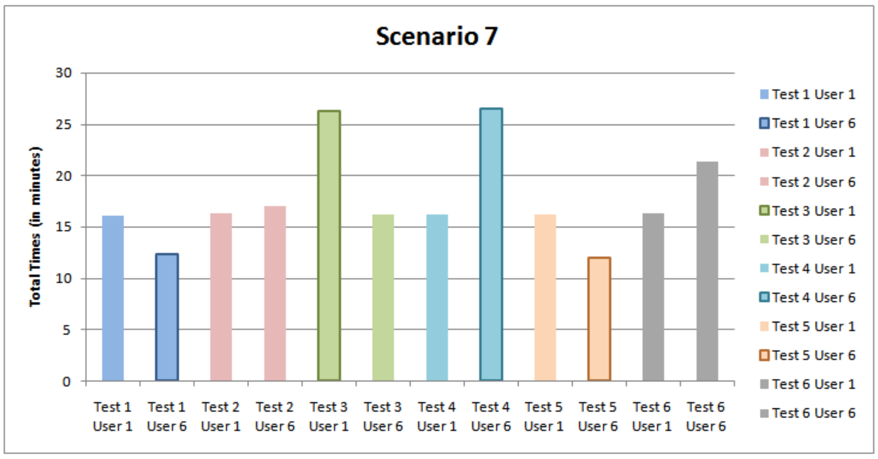


Figure 5.15: Total times for Scenario 7

The messages are, as before, sent and retrieved either by the starters' Walls, group Wall or Applications' Wall.

In order to facilitate the reading of each Job request, the comments field contains the User number, how many *Gridlets* the Job has and the test number, i.e. "Comments=job6.7.1" is introduced in the Job description, where 6 is the User number, 7 is the *Gridlet* number and 1 is the test number. Also, the *Gridlets* requirements and processing times are unchanged from the previous scenario.

5.5.1 Results

The results for scenario 7, as depicted in Fig. 5.15, brings us closer to understand how SNCS performs in a realistic environment. In this scenario we can see that the total times can vary depending on factors such as number of *Gridlets*, users states (*Offline* versus *Online*), number of users/groups involved, Social network latency and use of concurrent *Gridlets* (or Gridlet queue).

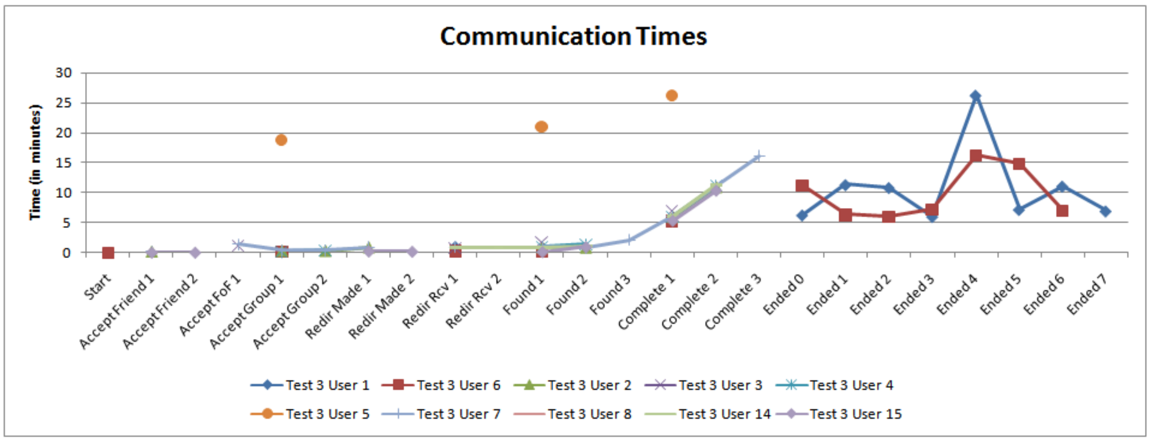


Figure 5.16: Communication Times for Scenario 7 Test 3

The times on this scenario are around 16 minutes to complete both Jobs, however we can see that in Test 1 and 5 the Job initiated by User 6 was completed 5 minutes earlier than in the other tests, this is due to the fact that the *Gridlets* were evenly distributed through the available users.

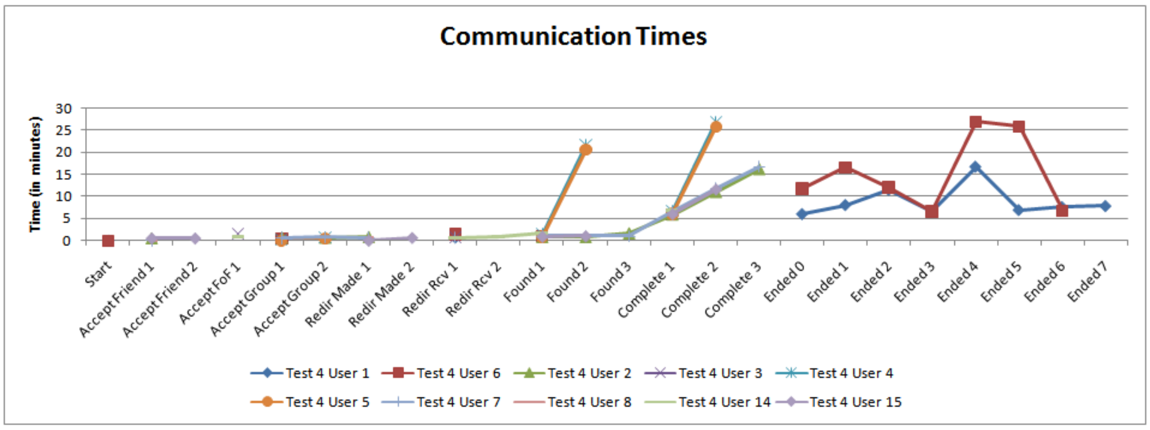


Figure 5.17: Communication Times for Scenario 7 Test 4

In Test 3, as depicted in Fig. 5.16 we can see what happens in the situation where a user goes *Offline* and all the remaining users are already occupied, making the total time much higher.

In Test 4, as depicted in Fig. 5.17 we can see the added time of the Social network latency, where two of the *Gridlets* were retrieved only after all other *Gridlets* were already processed, and thus hindering SNCS performance.

We can also see in Test 6, depicted in Fig. 5.18, that the total time is hindered by the uneven amount of

Gridlets that a user can take, in this case User 7 gathered 4 *Gridlets* to be processed when the computer had idle cycles to spare. However, we can say that the total time still remains with a speedup compared to how much it would take in the users' computer. This would be more so with longer running *Gridlets*.

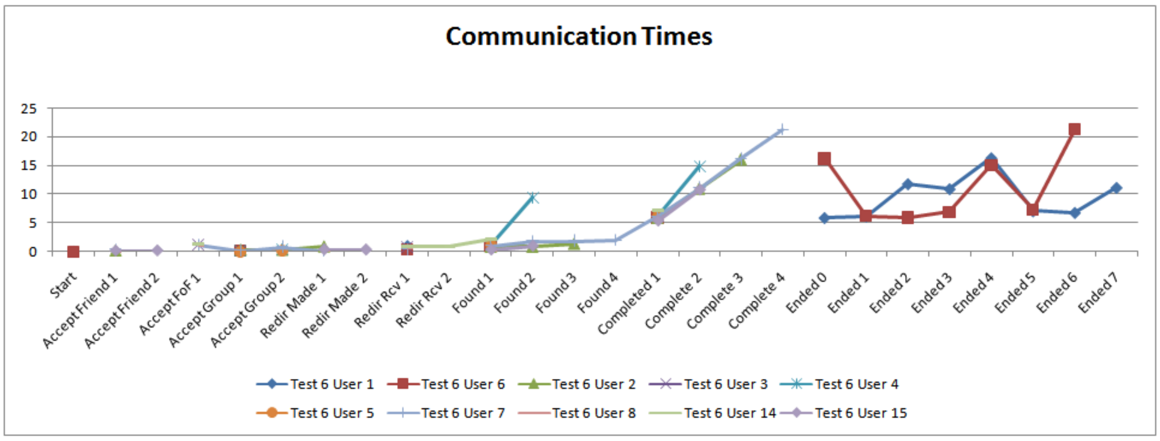


Figure 5.18: Communication Times for Scenario 7 Test 6

5.6 Scenario 8

Scenario 8 was made in order to evaluate the performance with a real program that renders images. In this scenario, as depicted in Fig. 5.19 we have one friend, one FoF and two users in a group (not counting with the starter) where one of them is the friend. The goal of this scenario is to know if SNCS can function with a real processing program, such as Pov-Ray,[1] which is used in the tests.

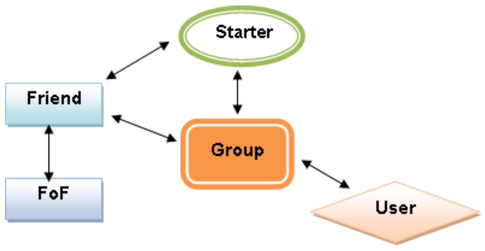


Figure 5.19: Scenario 8 View

For each test the number of *Gridlets* to be completed is four and their consuming times in the processing computers are undefined, meaning that it depends on the computers' hardware states and capabilities. However, the first data file (for Test 1) is smaller than the second one (used in Test 2) and Test 3 uses the same file as the second test, but with different rendering options. Furthermore, we use an Application to Application transfer method to retrieve the data files in both ways (starter - user and vice versa) for each test. Also, we assume that the client applications are running prior to the start of the Job.

Test 1 is initiated with the property arguments as being:

"-A0.3 -W1280 -H720 -D -O"$dir.output$abyss_rend.bmp" -P +Q9 /EXIT /RENDER "$dir.exec$abyss.pov""

[1]Pov-Ray: `http://www.povray.org` accessed on 15/10/2010

and the program property as “pvengine64.exe”, which in every SNCS is defined (by the user) in the “Programs list”.

For Test 2, the arguments property is altered to become:
“-A0.3 -W1280 -H720 -D -O”$dir.output$balcony_rend.bmp” -P +Q9 +R5 /EXIT /RENDER ”$dir.exec$balcony.pov””
and with the same program parameter as the latter test.

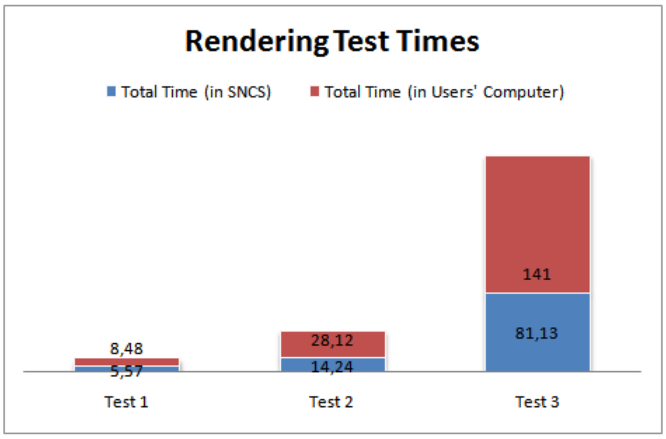


Figure 5.20: Rendering Test Times for Scenario 8

In Test 3 we modify the arguments property to be:
“-A0.0 -W3921 -H2767 -R200 -D -O”$dir.output$balcony_render.bmp” -P +Q9 /EXIT /RENDER ”$dir.exec$balcony.pov””, which modifies the images properties, such as anti-aliases, resolution and how many rays POV-Ray will supersample with when it is anti-aliasing, in order to have a longer running *Gridlet*, and also the program property still remains the same.

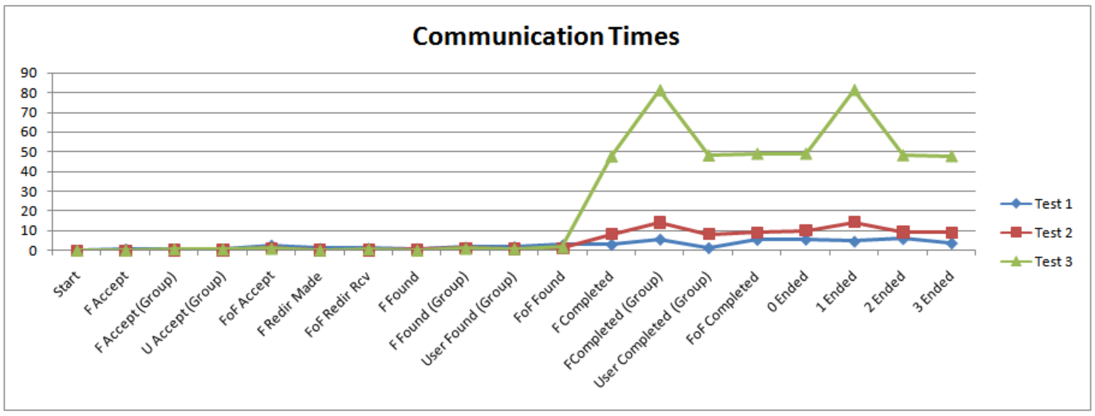


Figure 5.21: Communication Times for Scenario 8

5.6.1 Results

The results for scenario 8 confirmed that SNCS can gain speedups against local execution, as depicted in Fig. 5.20. Where we have the total times in Test 1 around 6 minutes, Test 2 around 14 minutes and Test 3 with 81 minutes.

Furthermore, in all the tests the friend user processes 2 *Gridlets*, meaning that it queues one to be processed when it has idle cycles to spare. We can also see in Fig. 5.21 that although each task can take some time to execute, the average performance can be acceptable for *Gridlets* that have higher processing times.

Also, we see that in Test 2 the *Gridlets* take around 5 minutes more to be processed than in Test 1, meaning that we can define that the processing time for the second data file is greater. Moreover, the first test suffers from communication latency, i.e. the task FoF Accept (accepting the Job from the Applications' Wall) takes more time to happen than in the second test.

Test 3 demonstrates that with longer running *Gridlets* the variables that hinder the overall performance, can be amortized by the difference that it would take to process all the data in the user's computer.

5.7 Discussion

When comparing with traditional processing, SNCS decreased the total processing time of what it would have consumed in the users' computers, meaning that SNCS achieves overall speedups on Jobs.

We can also state that the overhead that SNCS imposes on the overall process can be minimal compared to the time it takes to process a *Gridlet*, which in realistic terms it can be more than 1 hour. However, times can be hindered from the fact that searching for resources may not return positive results, or that the total resources available are less than the number of *Gridlets* to be processed, or even that latency of Facebook servers may vary with their global traffic load.

We can also conclude that the number of messages varies with the number of users (friends, FoFs and groups) that comes in contact with the Job, while varying with the number of *Gridlets* comprising the Job, i.e. in scenario 6 the number of messages in total were 41, considering that between friends and groups there are 5 messages for each user that accepts a Job, and that for FoFs there are 8 messages for each.

We can state that the number of messages sent to Facebook are proportionally increased by the number of users in the network, meaning that a Job may receive as many accepts and denies messages as users in the network. Although, the user may not be aware of this in the long run, because those messages are erased when they are not needed, making a clean environment in Facebook, meaning that we can accomplish our goal of making SNCS viable to use Facebook without hindering the usage of the Social network.

We can also conclude that the method used to contact FoFs hinders the total times, although in our tests the delays were not significant as compared to the overall process.

Moreover, we can confirm that the users can donate their resources (CPU time) for other users' consumption and for users' groups that would have interest in acquiring more processing power. Also, takes advantage of other users' resources with the same interests (or in the same groups) to further speedup their own programs.

In conclusion, even with the latency variables and excess messages introduced by the interaction between users, using a Social network, SNCS can definitely use the dispersed and idle resources available on these networks to speedup application execution, that would take more time in the users' computer.

6 Conclusions

Contents

In this dissertation we presented a new method of resource and service discovery through the use of a Social network. It is also considered that by making use of a Social network already established, we can involve more people to donate their computers' idle cycles. Also, we analyzed Peer-to-Peer networks and Grids to understand the related problems like efficient resource discovery, while also analyzing Social networks and user interactions to understand how we can achieve our works' goals.

The first projects in distributed computing, such as Distributed.net[1] and SETI@home [ACK$^+$02] opened the possibility of using millions of house hold computers for computer intensive problems that could take much more time than in supercomputers. These projects use a client-server base platform to enable home computers to gather information to be processed on their side and ultimately send the results back.

The idea of distributed computing has enabled other projects (such as XtremWeb [FGNC01]) to create environments to execute common applications used by desktop computer users. Meaning that anyone can join or create its own network to share and receive idle cycles for its own usage. Although, this may not be practical for common users, because they might not have the resources or capabilities to gather enough users (or computers) for their problems. Their idea creates small networks within communities, or enterprises in order to gather idle cycles for common applications' execution.

The Ginger Middleware [SFV10, VRF07, RRV10] aims to solve this problem, in order for any user to be able to consume idle resources (however it lacks the supporting infrastructure for users to connect with each other). Moreover, to leverage the process of sharing, they introduced an application and programming model based on the *Gridlet* concept, which is described as containing chunks of data and the operations to be performed on it, and also other information, such as the estimated cost to process the data.

Regarding resource discovery, some approaches emerged from P2P networks, in order to locate files and contents within peers more efficiently. An early method for such was *Flooding* [PFA$^+$05], which sends a message to anyone in the network until it reaches the required resource (or a limit is reached), which proves to be inefficient with the fact that not every peer may have the capabilities to reply to all the requests.

Other blind methods were considered, such as Random walks and multiple random walks [TR03], which randomly crawl the network to find the specific location of the resource, however they still lack the capabilities to direct their searches to the right peers.

Also, other methods combining the previous were considered, such as direct searches [LCC$^+$02], or forwarding indices [CGM02], which follows a not-so-random walks through the peers to locate the resources. However, for these methods to function correctly it is necessary that the resource information is available on the network.

Social networks were a step forward for user interactions in the Internet, meaning that Web sites, such as Facebook and MySpace are used for personal or business interactions at any given time, i.e. friends interactions or advertising.

Studies done to these networks demonstrate that they follow some properties of Small-World networks. Meaning that on these networks a user can reach another with just a few links, and also there is a small

[1]Distributed.net Web site: `http://www.distributed.net` accessed on 05/01/2010

group of users with many links (to others) and a larger group with fewer links. We can also see this in a P2P perspective, where users with many links are super-peers connected by users with fewer links (peers). Which made us believe that we could utilize these networks for other purposes, much like using P2P networks for global distributed computing.

Social networks have attracted the attention of other projects to utilize those networks for other purposes rather than the ones they were initially intended for, because of the potential of reaching millions of users without having to create their own networks. Also, because these networks export their own APIs to interact with their databases, allowing outside applications to gain faster population (or to be known by more people).

These Social networks use a *REST*-like interface or also a Graph interface (in case of Facebook), which is an added layer to the HTTP protocol, that communicates with HTML, XML or JSON objects to retrieve and send information, such as Users' name, friends, groups, among others.

Other projects have surfaced on the Internet that share the same properties as the previous distributed projects, meaning that their goal is to utilize the idle resources scattered on the Internet (users' computers) to do cycle-stealing. However, these new projects use the users' Browsers to achieve their goals, such as Collaborative Map-Reduce[2] and Plura Processing.[3]

Our work describes Social Networks for Cycle-Sharing (SNCS), a Web-enabled platform, which is designed to use Facebook, to search for potential idle resources available on this type of network, also enabling public-resource sharing within a Social network.

The main approach for SNCS was to have a client application split into two parts. One that interacts with the Social network using *REST* or *Graph* protocols. And another to interact with the users' computers for local resource discovery and the Ginger Middleware for creation and aggregation of *Gridlets*.

SNCS main concern was to actually achieve resource and application discovery while being able to do resource sharing, meaning that our works' concerns were to utilize users' computers in a way that would help common users to share their resources (when not needed) and to use others' resources to gain computational cycles for their own applications.

Also, we described, in Section 3.2, the SNCS architecture which is comprised of a set of components with determined functions. Moreover, some of the SNCS components depend on libraries such as *RestFB*[4] (which eases the interactions with Facebook, concealing the use of other objects in other languages), or SIGAR[5] (which gives a faster way to access computers' information).

Moreover, the users would only need to submit a Job, which is comprised of *Gridlets*, to SNCS in order for the SNCS client to interact with other client applications to distribute these *Gridlets*. Also, we defined *Gridlets* as containing data (partial data from the Job), the specified program and its arguments to process that data, while also having the requirements for which SNCS would be searching for, within the Social

[2]Collaborative Map-Reduce in the browser: `http://www.igvita.com/2009/03/03/collaborative-map-reduce-in-the-browser` accessed on 05/01/2010

[3]Plura Processing: `http://www.pluraprocessing.com` accessed on 05/01/2010

[4]RestFb Web site: `http://restfb.com` accessed on 24/08/2010

[5]SIGAR library: `http://www.hyperic.com/products/sigar` accessed on 15/10/2010

networks' available resources.

Furthermore, we established that interactions between SNCS clients and Facebook would be needed. As such, we demonstrated, in Section 3.3, that a flow of communication is created when a user submits a Job onto SNCS. Meaning that, after a successful submission of a Job, SNCS requesting client starts a search for resources that would meet the requirements of that particular Job. Moreover, it sends a Job Search message to the users' friends and to groups which would have the capabilities to process the Job, and also the users' friends of friends (FoFs).

In the prototypical example, in Section 3.6, we explained in more detail how the interactions are performed, starting with the users' Job submission to its conclusion. And also, how the search for resources would be accomplished, meaning that, by using Facebook API to send and retrieve messages we needed to deal with some facts, such as sending messages to FoFs through the user's friends' Walls.

On Chapter 4, we explained the SNCS implementation, detailing some of the more complex expects of SNCS, meaning the technologies used to ease the interactions between the client applications, the users' computers, and also the Social network used. Furthermore, we give an overview of how the client application was developed and some of the Java classes created for that purpose, in Section 4.2.

We also described, in Section 4.4, the internal representation of the data structures and message schemas used on the client application, that have the responsibility of holding information needed to complete a Job, and also help SNCS organize incoming *Gridlets*.

Furthermore, we created message schemas to send (and retrieve) them to (from) Facebook, because the Social Network does not allow some types of messages, such as XML based. As such, we used a separator of *Strings*, in order for a simple usage and human readable way to be allowed on the users' Walls, while also comforting the user as what SNCS would send to others.

However, the choice of using Facebook as the Social network for resource discovery and cycle-sharing, brought some constraints to SNCS. As described in Section 4.6, Facebook enforces some limitations either with their Use Terms or their API. For example, Facebook limits the size of the messages that can be sent by outside applications (Facebook applications), which was dealt with a simple splitting method to enable large messages to be passed to other users. And also, as Facebook is still developing the Graph protocol, some functionalities do not work as expected, therefore we used the discontinued *REST* protocol for those methods.

In the end, Chapter 5, we evaluated SNCS with scenarios that would give us the idea of how it would manage in such environments. Meaning that, several scenarios were created in order to test SNCS regarding its performance, stability and viability for using a Social network to achieve cycle-sharing, and resource and application discovery.

The first scenarios were created in order to evaluate SNCS viability, either by connecting the starter user with a friend, or also linking with people in a group. The results, for these scenarios, gave us insight on some variables that hindered the performance of SNCS, such as Facebook latency. But also, it gives us the idea that it is viable to use a Social network to do resource discovery and cycle-sharing.

The last scenarios, such as scenario 7 in Section 5.5, describes a more realistic environment with a more complex network of users. Meaning that, we created its layout in an attempt to maximize the diversity of users' roles and for the two Jobs to reach more users. Moreover, in scenario 8, in Section 5.6, we tested SNCS with a processing program that renders images (Pov-Ray)[6] in order to evaluate SNCS with realistic applications.

With the obtained results we can conclude that while the total times for processing a Job gained speedups against what it would have consumed in the users' computers, it can be hindered by some variables such as latency of Facebook servers, and the fact that searching for resources among Social networks users may not return positive results, or that the total number of available resources are less than the number of *Gridlets* that comprises a Job.

However, with functionality and quantitative evaluation, we can conclude that the results are encouraging despite the overheads introduced by the variable Facebook latency, and the intermediate messaging among FoFs. In fact, with SNCS, Jobs are completed faster than in the user's computer releasing it for other tasks. Also, the performance gains would increase with longer running *Gridlets* (more realistically about 1 hour) by amortizing overheads attributable to Facebook and communication.

Moreover, we can conclude that our works' goals have been successfully met. Meaning that, it is possible to utilize a Social network to do resource and service discovery, and also global distributed computing.

Furthermore, by introducing the concept of global resource sharing to Social network users, we believe that any common user can utilize SNCS to make use of idle resources scattered across the World to further advance process parallelization and continue decrease in processing waiting times. We also hope that this dissertation may contribute to the study and advancements made to novel cycle-sharing models.

6.1 Future work

In the future, we plan to augment the results to address the issues of having a realistic environment, completing it with results of real peoples' usage and longer running *Gridlets*. Also, to extend the use of processing programs that could include more common applications, such as video enconding, among others.

Moreover, we believe that Jobs completion and the search for resources would benefit with requirements' semantics, increasing the chance to direct *Gridlets* to peoples' computers that would satisfy the requirements, instead of having to be exact.

Also, the use of topic ontologies would greatly alter the number of users that may be able to help in a Job while also focusing on users that have more interest in such topics. Meaning that, we could search for specific groups using the Jobs' topics as a point of reference to obtain the groups that would be more favorable to that particular Job.

Furthermore, we could extend the parameters of cycle-sharing to make an advance scheduling, meaning that SNCS would request resources before starting a Job in order to avoid the lack of resources, and decrease the overheads attributable to resource discovery in the Job search requests.

[6]Pov-Ray: `http://www.povray.org` accessed on 13-10-2010

Moreover, to continue further development of SNCS and study on our works' goals, we plan to support other Social networks, to do cycle-sharing between the users.

Also, we plan to substitute the need of having a stand-alone application, by embedding the SNCS client with the Browser, in order to gather resources and process *Gridlets* while the users are navigating through the Social network or the Internet.

Bibliography

[ACK$^+$02] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. Communications of the ACM, 45(11):61, 2002.

[AFG$^+$09] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. 2009.

[AHK$^+$07] Y.Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In Proceedings of the 16th international conference on World Wide Web, page 844. ACM, 2007.

[And04] D.P. Anderson. BOINC: A system for public-resource computing and storage. In proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, page 10. IEEE Computer Society, 2004.

[ASBS00] LAN Amaral, A. Scala, M. Barthelemy, and HE Stanley. Classes of small-world networks. Proceedings of the National Academy of Sciences of the United States of America, 97(21):11149, 2000.

[ATS04] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys (CSUR), 36(4):371, 2004.

[Bac06] Lars Backstrom. Group formation in large social networks: membership, growth, and evolution. In In KDD 06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 44–54. ACM Press, 2006.

[BPSM$^+$00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. W3C recommendation, 6, 2000.

[BTM07] F. Boldrin, C. Taddia, and G. Mazzini. Distributed Computing Through Web Browser. In 2007 IEEE 66th Vehicular Technology Conference, 2007. VTC-2007 Fall, pages 2020–2024, 2007.

[CCRB10] K. Chard, S. Caton, O. Rana, and K. Bubendorfer. Social Cloud: Cloud Computing in Social

Networks. In 2010 IEEE 3rd International Conference on Cloud Computing, pages 99–106. IEEE, 2010.

[CGM02] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In International Conference on Distributed Computing Systems, volume 22, pages 23–34. IEEE Computer Society; 1999, 2002.

[CH97] A. Caglayan and C. Harrison. Agent sourcebook. John Wiley & Sons, Inc. New York, NY, USA, 1997.

[CPJ05] D. Crane, E. Pascarello, and D. James. Ajax in action. Manning Publications Co. Greenwich, CT, USA, 2005.

[DG04] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. To appear in OSDI, page 1, 2004.

[Dun01] C.R. Dunne. Using mobile agents for network resource discovery in peer-to-peer networks. ACM SIGecom Exchanges, 2(3):1–9, 2001.

[FBJW08] R. Figueiredo, O. Boykin, P.S. Juste, and D. Wolinsky. Social VPNs: Integrating overlay and social networks for seamless P2P networking. In Workshop on Collaborative Peer-to-Peer Systems (COPS), Rome, Italy, 2008.

[FGNC01] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb: A generic global computing system. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID 01), 2001.

[FKNT02] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. Computer, pages 37–46, 2002.

[FST02] Roy T. Fielding, Day Software, and Richard N. Taylor. Principled design of the modern web architecture. ACM Transactions on Internet Technology, 2:115–150, 2002.

[GABF06] A. Ganguly, A. Agrawal, P.O. Boykin, and R. Figueiredo. IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing. Arxiv preprint cs0603087, 2006.

[GDY06] L. Gao, Y. Ding, and H. Ying. An adaptive social network-inspired approach to resource discovery for the complex grid systems. International Journal of General Systems, 35(3):347–360, 2006.

[Goo98] D. Goodman. Dynamic HTML: the definitive reference. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1998.

[HHK04] F. Heine, M. Hovestadt, and O. Kao. Towards ontology-driven P2P grid resource discovery. In Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pages 76–83. IEEE Computer Society Washington, DC, USA, 2004.

[KLXH04] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In IEEE INFOCOM, volume 1, pages 96–107. Citeseer, 2004.

[LAM07] L. Liu, N. Antonopoulos, and S. Mackin. Social peer-to-peer for resource discovery. In Parallel, Distributed and Network-Based Processing, 2007. PDP'07. 15th EUROMICRO International Conference on, pages 459–466. IEEE, 2007.

[LCC$^+$02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th international conference on Supercomputing, pages 84–95. ACM New York, NY, USA, 2002.

[LKR04] J. Liang, R. Kumar, and K.W. Ross. Understanding kazaa. Manuscript, Polytechnic Univ, 2004.

[LSS$^+$09] Stefan M. Larson, Christopher D. Snow, Michael Shirts, Vijay S. P, and Vijay S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology, 2009.

[LZZ$^+$04] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the internet. In The 3rd International Workshop on Peer-to-Peer Systems. Springer, 2004.

[Man03] G.S. Manku. Routing networks for distributed hash tables. In Proceedings of the 22nd annual symposium on Principles of distributed computing, pages 133–142. ACM New York, NY, USA, 2003.

[MBAS06] M. Mowbray, F. Brasileiro, N. Andrade, and J. Santana. A reciprocation-based economy for multiple services in peer-to-peer grids. In Peer-to-Peer Computing, 2006. P2P 2006. 6th IEEE International Conference on, pages 193–202. IEEE, 2006.

[MGD06] A. Mislove, K.P. Gummadi, and P. Druschel. Exploiting social networks for internet search. BURNING, page 79, 2006.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. Proceedings of IPTPS02, Cambridge, USA, 1:2–2, 2002.

[MMG$^+$07] A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, page 42. ACM, 2007.

[MRPM08] E. Meshkova, J. Riihijrvi, M. Petrova, and P. Mhnen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. Computer Networks, 52(11):2097–2128, 2008.

[OBBO04] A. O'Connor, C. Brady, P. Byrne, and A. Olivr. Characterising the eDonkey Peer-to-Peer File Sharing Network. Computer Science Department, Trinity College Dublin, Ireland, Tech. Rep, 2004.

[PFA+05] C. Papadakis, P. Fragopoulou, E. Athanasopoulos, M. Dikaiakos, A. Labrinidis, and E. Markatos. A feedback-based approach to reduce duplicate messages in unstructured Peer-to-Peer networks. In Integrated Workshop on Grid Research. Springer, 2005.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001, pages 329–350. Springer, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, page 172. ACM, 2001.

[RRV10] P.D. Rodrigues, C. Ribeiro, and L. Veiga. Incentive mechanisms in peer-to-peer networks. In Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, pages 1–8. IEEE, 2010.

[Sch97] RR Schaller. Moore's law: past, present and future. IEEE spectrum, 34(6):52–59, 1997.

[Sco88] J. Scott. Social network analysis. Sociology, 22(1):109, 1988.

[SFV10] JN Silva, P. Ferreira, and L. Veiga. Service and resource discovery in cycle-sharing environments with a utility algebra. In Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, pages 1–11. IEEE, 2010.

[Sho98] S. Shostak. Sharing the universe- Perspectives on extraterrestrial life. Berkeley, CA: Berkeley Hills Books, 1998., 1998.

[SKM+02] D. Stainforth, J. Kettleborough, A. Martin, A. Simpson, R. Gillis, A. Akkas, R. Gault, M. Collins, D. Gavaghan, and M. Allen. Climateprediction. net: Design principles for public-resource modeling research. In Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing Systems, 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, page 160. ACM, 2001.

[SVF08] João Nuno Silva, Luís Veiga, and Paulo Ferreira. nuboinc: Boinc extensions for community cycle sharing. In SASO Workshops, pages 248–253. IEEE Computer Society, 2008.

[TAA$^+$03] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.C. Hugly, E. Pouyoul, and B. Yeager. Project JXTA 2.0 super-peer virtual network. Sun Microsystem White Paper. Available at www. jxta. org/project/www/docs, 2003.

[TR03] D. Tsoumakos and N. Roussopoulos. A comparison of peer-to-peer search methods. In Proceedings of the 6th International Workshop on the Web and Databases. Citeseer, 2003.

[TTP$^+$07] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-Peer resource discovery in Grids: Models and systems. Future Generation Computer Systems, 23(7):864–878, 2007.

[VRF07] L. Veiga, R. Rodrigues, and P. Ferreira. GiGi: An Ocean of Gridlets on a ”Grid-for-the-Masses”. In Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, pages 783–788. IEEE Computer Society, 2007.

[WBS$^+$09] C. Wilson, B. Boe, A. Sala, K.P.N. Puttaswamy, and B.Y. Zhao. User interactions in social networks and their implications. In Proceedings of the 4th ACM European conference on Computer systems, pages 205–218. ACM, 2009.

[ZL04] D. Zhou and V. Lo. Cluster Computing on the Fly: resource discovery in a cycle sharing peer-to-peer system. In Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid, pages 66–73. IEEE Computer Society, 2004.