Enhancing Online Communities with Cycle-Sharing for Social Networks

Nuno Apolónia, Paulo Ferreira and Luís Veiga

Abstract The Internet has made it possible to exchange information more rapidly on a global scale. A natural succeeding step was the creation of Social networks where anyone in the world can share their experiences, content and current information, using only their Internet enabled personal computer or mobile devices. Under this scope there are many Social Networks such as Facebook, Orkut or Youtube each one exporting their own APIs to interact with their users and groups databases. Studies done on Social Networks show that they follow some properties like the Small-World property. Meaning that traversing friendship relations, vast numbers of other users could be reached from each single user (e.g., Friends of Friends), even though users usually only interact (on a daily basis) with a restrict group of friends. 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 online virtual communities. Resources from these types of networks can be used to further advance studies in other areas which may be too computational intensive for using a single computer or a cluster, e.g. to process data mined from the vary Social Networks. We describe the design, the development and resulting evaluation of a web-enabled platform, called CSSN: Cycle-Sharing in Social Networks. The platform leverages a Social Network (Facebook) to perform discovery of computational resources, thus giving the possibility for any user to submit his own Jobs for remote processing. Walls, messages and comments in Facebook are used as the underlying transport for CSSN protocol messages, achieving full portability with existing Social Networks. Globally, CSSN gives the chance for common users to unleash the untapped computing power hidden in Social Networks, and exploit it using the cycle-sharing paradigm to speedup their own (or common) applications' execution.

Nuno Apolónia, Paulo Ferreira and Luís Veiga
INESC ID Lisboa / Technical University of Lisbon, Rua Alves Redol 9, 1000-029 Lisboa, Portugal, e-mail: nuno.apolonia@ist.utl.pt, paulo.ferreira@inesc-id.pt, luis.veiga@inesc-id.pt

1 Introduction

The computing power has been significantly increased in the past few years, however there are still many computational problems that need an enormous and increasing 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 such as CPU, memory or data storage, and all of them can be found in the millions of desktop computers all around the World. In other words, the needed resources can be gathered from every house hold or from offices and even from our daily devices, such as notebooks or mobile phones.

The idea to use idle cycles for distributed computing was proposed in 1978 by the WORM computing 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 available 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. They employ coordinated resource-sharing and problem solving environments, that made possible to make distributed processing of large computational (and scientific) problems.

With the Internet, the available resources for such projects were extended. Projects like SETI@Home [ACK+02], Folding@Home (folding.stanford.edu), Distributed.net (www.distributed.net) 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 (Plura Processing response to the Digsby Controversy in Wordpress.com).

Motivation: The Internet has also enabled information and content sharing by using Peer-to-Peer (P2P) networks [TTP+07]. These networks are usually formed by interconnected home desktop computers. They can be categorized in terms of their formation as being structured or unstructured. Unstructured P2P systems are characterized by having an underlying topology unrelated with the placement of the contents, as opposed to Structured P2P systems where it is attempted to place the contents in locations related with the content identification. Furthermore, optimizations were done to leverage the performance for locating contents and their scalability (in terms of traffic load). The resulting systems are generally called Hybrid P2P systems that highlight two types of users. The users that provide more bandwidth are called super-peers and those with low bandwidths are called peers; the last ones are connected to the super-peers [TTP+07].

These networks raise 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 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 resource.

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 (facebook.com/mobile). Under this scope there are many Social networks such as Facebook (facebook.com), Orkut (orkut.com) or Youtube (youtube.com) each one exporting their own *APIs* to interact with their users and groups data bases, e.g. Facebook API[1] and OpenSocial[2]. Also, these networks have great potential for financial benefits, such as Advertising.

Furthermore, studies show that the Social Networks have 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.

Background: In our work, we developed and evaluated a Web-enabled platform, called Cycle-Sharing in Social Networks (CSSN), that interacts with a Social Network (Facebook) to be able to locate and search for idle resources among its users. We leverage an existing middleware, Ginger [VRF07, SFV10] for task (called *Gridlets*) creation and aggregation.

CSSN 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 which Facebook provides. This interface gives us access to users' information, such as their friends, groups and Walls. 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.

We designed a communication protocol to allow interaction among the CSSN clients and execute Jobs (sets of tasks) successfully. This means that we use Face-

[1] Facebook Developers: `developers.facebook.com` accessed on 05/01/2010
[2] OpenSocial Web site: `code.google.com/apis/opensocial` accessed on 05/01/2010

book Walls (users/groups/Application Walls) to send and retrieve messages sent by other CSSN clients. CSSN starts the discovery 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 which may be willing to help (or have the Jobs' requirements). Furthermore, we extend the reach of gathering resources to contain friends of friends (FoFs), by contacting FoFs CSSN clients.

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

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

In the last scenarios we augment the network size used, to become more realistic in terms of users' roles (friends, FoFs, group members). In addition, we create *Gridlets* representing tasks to be performed by a known program (Pov-Ray)[3] to render an image. These scenarios were made, in order to conclude that CSSN can gain speedups against local execution; however it is demonstrated that CSSN can be hindered by variables such as Facebook latency, and searching for resources may not return positive results, or even that the number of *Gridlets* may surpass the number of donating users.

Current Shortcomings: The public-resource sharing and cycle-sharing systems that are widely used today, are not concerned with the common users' needs. They are mostly used for intensive computational 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 flexible 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 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 defined by predesignated users and targets occasional users (which may not be aware of the system).

[3] Pov-Ray web site: `povray.org` accessed on 13/10/2010

Contribution: Our main contribution is the CSSN platform with its architecture, messaging protocol, system monitor and client application. It performs resource and service discovery on top of a Social Network already established. Furthermore, CSSN needs to be able to gather idle cycles from users' computers and communities that would be willing, and capable of executing a Job, in order to achieve cycle-sharing on a Social Network. It can also allow 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, CSSN can use an already established network as in case of Social Networks (Facebook, MySpace, among others) to interact between users to share their idle cycles.

2 Related Work

This section offers a review on relevant works and technologies more related to our focus, addressing: i) Social Networks, ii) peer-to-peer networks, Grids and Distributed Computing, and iii) deployment mechanisms and code execution via the web.

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 (List of Social Networks on Wikipedia.org). 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 with whom the interaction may be daily. In the Social Networks, the basic (real life) behaviors or interaction patterns still apply. By grouping people in the same areas or topics, it is easier to exploit those interactions, because people understand better what the distributed tasks will accomplish and are willing to participate. 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 enable ad-hoc VPNs [FBJW08].

Small-World networks can be described by the following properties: the local neighborhood is preserved; and the diameter of the network, quantified by the average shortest distance between two vertices, increases logarithmically with the number of vertices. It is then possible to connect any two vertices in the network through just a few links [ASBS00]. Growing of such 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.

Many Social Networks also have ways of connecting users without being linked as friends; such connections are called groups or communities, where knowledge is exchanged within a specific topic of interest. The creation of such groups and their subsequently taking shape and evolution over time is inherent in the structure of society; this means that people have the tendency of coming together to share knowledge on a particular theme [Bac06]. We give special focus on **Facebook** and **OpenSocial**, explained by their size and possibility of access to user databases, by means of the APIs they export. 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.

The Facebook API and the OpenSocial API enables Web applications to interact with the server using a *REST*-like interface[4] or in case of Facebook also a *Graph* interface.[5] 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**.[6] 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. **PeerSpective**[7] is a Social Network-based Web Search. It tries to merge the Social Networking with search engines, to improve their ranking system, by understanding how people perform searches. They employ the search results, made by friends, into the ranking system. 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]. **Social Cloud** [CCRB10] is described as being a model that integrates Social Networking, cloud computing [AFG+10] and *volunteer computing*. In this model, users can acquire the resources (in this project the only resource considered is disk space) by exchanging virtual credits, making a virtual economy over the social cloud computing. Users can gather resources from their friends (either by virtual compensation, payment, or with a reciprocal credit model [MBAS06]), which allows this model to approach the objectives of public-resource sharing. 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[8] in Social Networks such as Facebook.

---

[4] Representational State Transfer: `tinyurl.com/6x9ya` accessed on 05/01/2010
[5] OpenGraph Protocol: `opengraphprotocol.org` accessed on 23/08/2010
[6] Progress Thru Processors: `facebook.com/progressthruprocessors` accessed on 05/01/2010
[7] PeerSpective: `peerspective.mpi-sws.org` accessed on 05/01/2010
[8] How Facebook could make cloud computing better: `tinyurl.com/237ddem` accessed on 15/10/2010

Peer-to-Peer networks, Grids and Distributed Computing: Peer-to-Peer (P2P) networks and Grids are the most common types of sharing systems. They evolved from different communities to serve different purposes [TTP+07]. The Grid systems interconnect clusters of supercomputers and storage systems. Normally they are centralized and hierarchically administrated, each with its own set of rules regarding resource availability. Resources can be dynamic and thus may vary in amount and availability during time, and have to be known beforehand among the network. Grid systems 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.

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 vary in time with more intensity than with Grids. P2P 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 Multi-player Online games using P2P [KLXH04] to alleviate server load, distributing tasks as SETI@Home [ACK+02], data streaming for watching TV[9]). The nodes (or peers) are composed by anonymous or unknown users unlike in Grids, which raises its own problems with security or even with forged results [TTP+07].

Both Grid and P2P systems have been converging by relaxing rules from Grid systems and opening P2P applications for more computation, and not simply storage. 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.

In **Unstructured P2P systems**, the placement of contents (Files) is completely unrelated to the overlay topology and they must be located (or searched). These systems, such as Gnutella[10] 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* [PFM+05], *Dynamic Querying*[11], *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]. **Structured P2P 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], 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]) mean-

[9] PPStream: `ppstream.com` accessed on 05/01/2010
[10] Gnutella Protocol: `tinyurl.com/yaz95ep` accessed on 07/01/2010
[11] Dynamic Querying Protocol: `tinyurl.com/6jh958q` accessed on 05/01/2010

ing 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.

There are also **hybrid approaches** created to make up for the lacks that each approach has and still retain their benefits. A common optimization to unstructured systems is to have 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. Thus, flooding of messages are only passed through the super-peers and does not cause problems with peers which cannot handle too many search requests [TTP+07]. **Kademlia** [MM02] uses a group of peers (that are near each other) known as buckets to locate files, avoiding some flooding problems. Also peers may have the ability to change their positioning, 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]).

SETI@Home [ACK+02] aims at using globally distributed resources to analyze radio wave signals that come from outer space, hoping to find radio signals originated from other planets on our galaxy.

For this project, having more computing power means it is possible to cover a greater range of frequencies, instead of using supercomputers which owners 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 are divided in small units of fixed size to be able to distribute among the BOINC clients (that would be located in any user computer operating as a *screen saver* when there are idle cycles). Then, each client computes the results in its spare time and sends it to the central server asking for more work to do. In this process, clients only need to be able to communicate with the server when they finish computations (or for asking more data). The client (application) is platform independent, in order to reach as many Internet users as possible. A ranking system allowed users to compete against other users, to motivate them to use this system. Thus, 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. **BOINC** (Berkeley Open Infrastructure for Network Computing) [And04] is a platform for distributed computing through volunteer computers; it emerged from the SETI@Home project and became useful to other projects.[12] 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.

Folding@Home [LSSV09] is an example of a BOINC system that studies protein folding, determining whether 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. 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.

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

Another example of a BOINC system is 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 to improve understanding of how sensitive climate models are to small changes, e.g., in carbon dioxide and sulphur cycles. The project has many similarities with other BOINC systems, but the computational tasks are different. **Distributed Computing Projects** embody another approach to leverage spare cycles across the Internet. The first relevant projects to distributed computing were distributed.net and GIMPS. **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. The 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**[13] 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 is $2^{43,112,609} - 1$, which has about 12.8 million digits. Both projects use its own Client and Server applications, following the same idea as the BOINC projects.

There are many other projects for distributed computing (List of Distributed Computing projects on Wikipedia.org). 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 research topic. With BOINC Extensions for Community Cycle Sharing (**nuBOINC** [SVF08]), users without programming expertise may address the frequent difficulties in setting up the required infrastructures for BOINC systems and subsequently gather enough computer cycles for their own project. The nuBOINC extension is a customization of the BOINC system, that allows users to create and submit tasks for distributed computing using available commodity applications. They try to bring global distributed computing to home users, using a public-resource sharing approach.

The main concept of **Ginger** (Grid Infrastructure for Non-Grid Environments) [SFV10, VRF07, RRV10] is that any home user may take advantage of idle cycles from other computers, much like SETI@Home. However, by donating idle cycles to other users to speedup their applications, 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*. 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

[13] The Great Internet Mersenne Prime Search: `mersenne.org` accessed on 05/01/2010

a P2P model to provide a large-scale deployment in a self-organized way. **Social-P2P** [LAM07] is a social-like P2P algorithm for resource discovery. It mimics the way humans interact in Social Networks. Knowledge is passed on among people in these networks as a means of sharing information; moreover, people recall information in memory to find the right persons to interact with, when searching for a given resource. However, in most circumstances, people recall something because they had similar knowledge, or in the same context of the requesting resource, instead of actually knowing about it. A person may be recalled solely 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. This algorithm serves as a demonstration that human interaction strategies are successful for resource discovery in P2P networks. Nevertheless, in their simulations, a dynamic environment with only probabilistic request structure and file sharing was considered.

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, Chrome 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, browsers and running applications contact servers by using a standard protocol named HTTP (Hypertext Transfer Protocol).[14] This protocol is used for retrieving interlinked resources, called hypertext documents. This protocol follows a request-response sequence of messages, where the basic request methods (or verbs) are *GET, POST, PUT* and DELETE to, respectively, request a representation of the specified resource, submit data to be processed to the identified resource (this may result in the creation of a new resource or its update), submit a document to be stored in the server, and delete a document stored within the server.

Other languages can also be used either on the client or on the server, to generate HTML dynamic content [Goo98], e.g. Asynchronous JavaScript and XML (AJAX)[15] being client side, Hypertext Preprocessor (PHP)[16] being server side.

AJAX [CPJ05] is an 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. It is able 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*) [FT02] is a style of software architecture for distributed Hypermedia (including graphics, audio, video, plain text and hyper links) systems. The main concept is that existing resources can be referenced with a global identifier (e.g. URI in HTTP), and also the exchange of a representation of

[14] HTTP 1.0 specification: `www.ietf.org/rfc/rfc1945.txt` accessed on 05/01/2010
[15] AJAX article: `adaptivepath.com/ideas/e000385` accessed on 05/01/2010
[16] PHP Web site: `php.net` accessed on 05/01/2010

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 that consists of a document format with structural markers, XML generally used to represent arbitrary data structures, for example in Web services, and JSON (JavaScript Object Notation) as a lightweight data-interchange format, made in order to ease the computational parsing of data. The last one 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 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. Any Web site can use this technology to organize information in a structured way, similar to Facebook pages. Also, it is built on standards (RDFa)[17] 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;[18] 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. Furthermore, their concern was only to apply the Map-Reduce algorithm [DG08] on the data collected from the server. 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,[19] which is a proprietary executable code made to enable idle cycle-stealing. Its main idea is that everyone that browses 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. It is claimed that users can sacrifice their CPU time, even without their knowledge, to benefit computationally intensive projects (much like SETI@Home). However, this approach may not be best to suite the users, because they need to understand the tasks' relevance (Plura Processing response to the Digsby Controversy in Wordpress.com). Moreover, they use simple Web pages and games (Adobe Flash based) to embed their processing code to execute the needed tasks.

Analysis: Social Networks, are popular infrastructures for communication, interaction and information sharing on the Internet. A user only needs his/her Internet enabled device (e.g., desktop computers, notebooks, mobile phones) to access Web sites, such as Facebook, MySpace, Orkut, LinkedIn, and many others, to be able to send or receive personal or business information.

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, such as cycle-sharing.

---

[17] RDFa standard: `www.w3.org/TR/rdfa-in-html` access on 23/08/2010
[18] Collaborative Map-Reduce in the browser: `tinyurl.com/ad248t` accessed on 05/01/2010
[19] Plura Processing: `pluraprocessing.com` accessed on 05/01/2010

Some global distributed computing projects make use of distributed computing technologies, to solve their computer resource shortage (CPU time), by using the millions of Internet enabled users' computers all over the world. On all these projects, we can say that they do not have the flexibility to change their own research topic (the goal of their data processing), and also only used to further advance their own research.

While there are other systems that give the ability of cycle-sharing to common users, each of them employ their own platforms to enable common applications to be executed on peers, not Social Networks. Moreover, some systems allow interaction only on P2P networks, meaning that their networks have to be created by the users.

3 Architecture

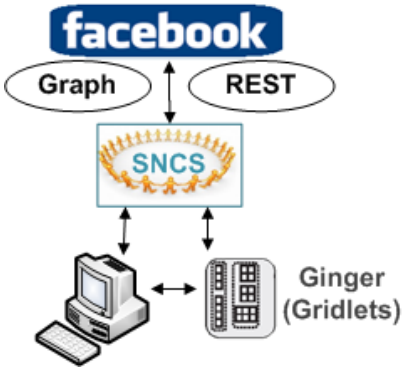

Fig. 1 Cycle-Sharing in Social Networks global overview.

This work uses a Social Network (Facebook) to discover resources for the execution of Jobs (which are composed of *Gridlets* [VRF07]) submitted by the users, and to discover computer full capabilities (e.g. processor information) and users' profiles, such as the groups which they belong to and their friends.

Users should be able to *install* CSSN, which is a Web-enabled platform, (Fig. 1) into their computers. Then, the user has the ability to *log in* into their Facebook account, by means of the *Facebook Connect*, 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. The client application also gives the user the ability to initiate a Job, by using CSSN user interface.

To actually locate resources through the Social Network, CSSN has the ability of searching local resources, by means of the *SIGAR* library, that gives information, such as processors status, memory available, among others. 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 access to it. Note that this information may contain the programs that can be executed by a computer in the network to process *Gridlets*.

Cycle-Sharing in Social Networks must also have access to friends and groups through the Social Network API. It advertises users' availability to others, sending messages and scheduling tasks (i.e. search for information, *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).

The main approach for CSSN is to have a client application split into two parts: one that interacts with the Social Network; and another to interact with the users and the Ginger Middleware (in order to create and regroup *Gridlets*, which is out of the scope of this work [VRF07]).

Design Requirements: 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.[20]

Another requirement for CSSN 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, in order for CSSN to not interfere with the users normal usage of their computer or Facebook page, the CSSN client schedules *Gridlets* according to user preferences, meaning that friends have priorities for executing their *Gridlets*. Also, to prevent overuse of the computer, while the user is in an *Online* state, CSSN is able to stop its activities, i.e., the processing of requests and *Gridlets* only happens when there are idle cycles to spare. The CSSN client also removes any unnecessary posts that could prevent the normal usage of the Facebook page.

CSSN Architecture: The CSSN architecture, depicted in Fig. 1, relies on an 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 aggregation; and also the user's operating system to acquire the information and hardware states that are needed.

Jobs are considered to be tasks initiated by the users, and containing more than one *Gridlet* to be processed in someone else's computer; all Jobs state what they require in order to execute those *Gridlets*, so that the client application can search for specific users or groups.

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

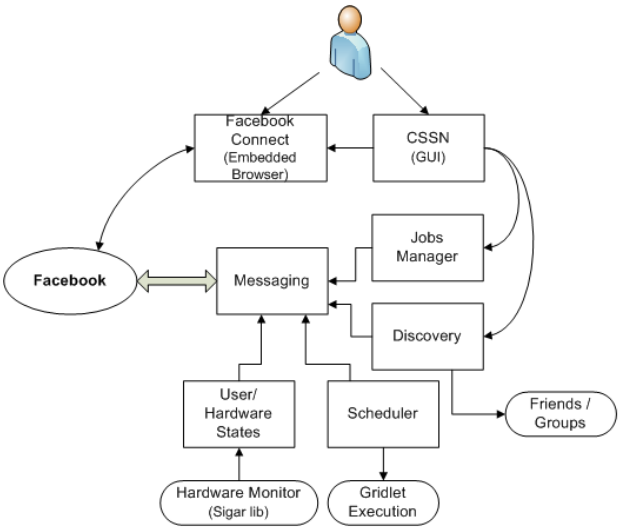

Fig. 2 Cycle-Sharing in Social Networks module view.

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

The architecture for CSSN is comprised of a set of components depicted in Fig. 2, and described as follows.

CSSN (GUI): this module performs the main interaction with the user via a graphic interface. 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 GUI interface, which is responsible for starting the chain of events for processing that Job (search for users, acceptance and execution of *Gridlets*).

Facebook Connect (Embedded browser): this component authenticates the user to Facebook (it displays the Web page given by Facebook for that purpose). Then, 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[21] to display the Web site for the user's authentication.

Messaging: this 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. This module also contains the

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

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 applied to the messages sent and retrieved, to specify what actions should be taken.

Jobs Manager: this is the module that runs a cycle of the following tasks (named "checking" cycle).

Verifying submitted Jobs that the user has in progress, and it assembles *Gridlets* to send 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 re-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 CSSN to answer on the Registration Post (in the Applications' Wall), that was made prior by the requesting user. This adds 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 CSSN has acquired a *Gridlet* message, it hands it to the scheduler module (described later) for ulterior execution. Moreover, this module has the task to remove all the Posts that are no longer necessary. This module can be stopped if the computer is in a *Online* state, in order to not interfere with the normal computer's usage.

Discovery: this module 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.

This module 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 execution to a later time, when the processor has idle cycles to spare. In addition, it sends the state of the CSSN client (*Online*, *Offline*, *Idle*) to the Social Network in order to inform other users of its state. The state *Online* should be active when the user has decided that the client application should run. The *Idle* takes place 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 CSSN to be running.

This module uses a submodule, named Hardware Monitor depicted in Fig. 2, 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, while also stopping its 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.

This module starts a submodule that is responsible for processing the *Gridlet*; it performs data transfer, executes the program that processes the data and upon completion 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).

CSSN Communications: CSSN 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, which can be comprised of several *Gridlets*, is initiated by the user, by submitting the Job on the CSSN 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*.

A search for resources is specified by the *Gridlets* requirements, e.g., a Job that consists of generating a image on POV-Ray, which needs 4 processors and 2048Mb of memory. The client application uses this information in order to gather users that have such resources (including the processing application) available.

After the user submits a Job, CSSN starts to perform the actions to complete it, 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 (as illustrated in Fig. 3).

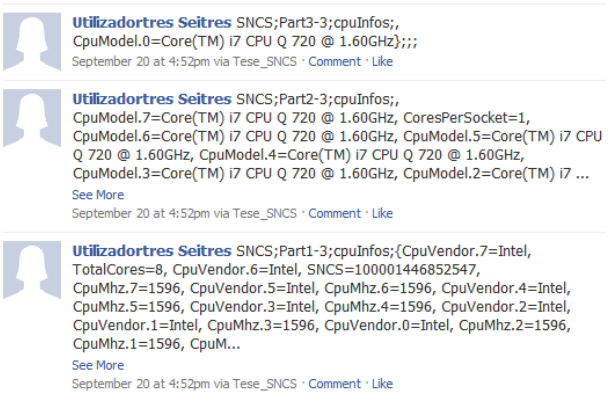

Fig. 3 Example of the Computer Information message on Facebook

There is an impossibility of directly contacting people and groups that are not in the friends' domain, such as Friends of Friends (FoFs). To handle this, CSSN client routes 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, i.e. 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.[22] As such, CSSN only goes as further as FoFs (2nd degree). The client application only contacts the users' groups that are able to help for the specific Job. It searches for computer information in order to determine the ones capable of processing the Job.

The discovery mechanism of CSSN tries to gather as much computer information as possible, and sends messages to the corresponding users and groups. Meanwhile the Job part stays alert for incoming messages on the users' Wall that may carry requests or stating availability.

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 application (client application), which enables the process of searching for people outside the scope of friendship.

Their client applications then tries 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.

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

The transfer of *Gridlet's* data occurs after a client application has retrieved the *Gridlet* message, and determined that it has idle cycles to execute it. The transfer can employ a direct connection between the CSSN clients (acting as peers). This may also be carried out 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 CSSN client determines that the execution of a *Gridlet* has failed, due to the processing program returning 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, CSSN can still reacquire the *Gridlet* from the users' Wall, to repeat it, 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.

Prototypical example: We describe a more detailed example of a Job submission and the steps CSSN takes to process it, as depicted in Fig. 4.

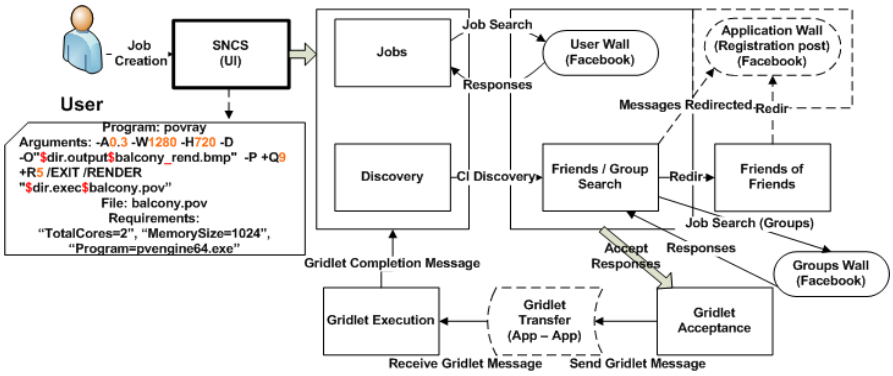

Fig. 4 CSSN Prototypical example

A user submits a Job, using the CSSN GUI, with the following properties.

The 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 needs, such as "TotalCores=2", "MemorySize=1024", "Program=pvengine64.exe" and the number of *Gridlets* that comprises this Job.

CSSN then starts the search for resources using the specified requirements; the requirements should be as accurate as possible to search for specific groups and users. To locate the resources, a client starts by sending a Job search message to the users' Wall. Meanwhile, it also requests the computers information (CI discovery)

from the people that are in the user's groups, in order to know which of the groups would be willing to accept the Job. Also, it tries to send a message to the user's 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, i.e. the Post ID for which it should be sent, and the type of message that the user should send (in this case their computer information).

Afterwards, CSSN 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 accept the Job, CSSN sends a *Gridlet* message; until all the *Gridlets* have been sent, this message is then received by the processing client application.

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 uses a direct 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 contains the necessary information in order for this client application to locate and retrieve the necessary data to be processed.

The CSSN client receiving a request, before it downloads 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 client application receiving the request, 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

The implementation of CSSN aims for a simple use by the end-users. Also, the different types of operating systems lead us to favor portability; therefore, we used Java as the main language. We chose to use Facebook over other alternatives, such as OpenSocial-based networks, because Facebook has a higher number of registered users than any other Social Networks. Another consideration was the fact that Facebook exports its own API and many libraries (such as *RestFB*) have been created to facilitate the usage of the API.

This section gives an insight on how the technologies were used, such as *Graph* and *REST* protocols. It also explains the schemas used for the messages sent/received to/from the Social Network chosen. The section ends with a view of some of the constraints that CSSN suffered from using Facebook as the Social Network for users' interactions.

Technology Employed: 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.[23] However, the functions (using REST) or connections (using 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.

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 external client applications to authenticate a user by means of their own Facebook Connect system, which is a Web page dedicated for the *Log in* process. Also, for the client applications 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.[24]

For the purpose of displaying the Facebook Connect Web page, we make use of the *JDIC* library. This enables us to display a Web site[25] to the users for the authentication process.

As CSSN also needs to gather the information about the local resources of the users' computer, we make use of the *SIGAR* library.[26] 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 Windows, Linux, among others, making possible the portability of CSSN to other systems.

[23] JSON: `json.org` accessed on 15/10/2010
[24] Facebook Authentication methods: `tinyurl.com/24edrkg` accessed on 27/08/2010
[25] Tese_CSSN Application Facebook Page: `tinyurl.com/6duzlmc` accessed on 15/10/2010
[26] SIGAR library: `hyperic.com/products/sigar` accessed on 15/10/2010

Message Schemas: CSSN uses Facebook to send and retrieve messages via the Facebook API. 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.

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.

In CSSN 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 sent from Facebook.

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.

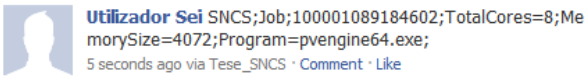

Fig. 5 Example of Job Search message on Users' Wall

These Schemas make use of an ordinary separator of *Strings*, as depicted in Fig. 5. Regarding messages that can be longer than the limit imposed by Facebook, such as the Computer information messages (Fig. 3), 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.

These Schemas are very simple and human readable, in order for Facebook to allow them on the Web site, and not consider them as "Spam" or other type of blocked messages. They are also human readable to assure the users what information is being sent to other users.

The Schemas represented in Fig. 5 and Fig. 6 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. 6 we can see that the user has responded to the Job Search with an accept message.

Fig. 6 Example of Job Search and Acceptance messages in Applications' Wall

CSSN 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 or their API.

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). To reduce traffic "spam" we avoid loops in message exchanges among users who are friends.

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.

5 Evaluation

In this section we present the evaluation of CSSN regarding its performance, stability and viability for using a Social Network to achieve public-resource sharing. Our focus is demonstrating the practicality of resource and service discovery, by recruiting as many users as possible to execute *Gridlets*. We also evaluate integration 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, as CSSN is designed to provide idle cycles to be used to process

the upcoming *Gridlets* with real world applications, the client application was tested with ray-tracing jobs in a "more realistic" environment as described in this section. In order to perform all the tests we constructed several scenarios, where the environment for each would change. In these scenarios we changed the number of users involved and also their roles, i.e. in scenario 1, as depicted in Fig. 7, we considered 2 friends, 2 FoF and a group with 3 users, where one of them was a FoF. The number of *Gridlets* as 7 and the processing time is only 5 minutes for each *Gridlet*.

In scenario 2, as depicted in Fig. 10, a "more realistic" scenario is considered, where there are Friends, FoF and other people connected by a Group, while also increasing the number of Jobs to 2, and the total number of *Gridlets* to 15, with the same processing time for each *Gridlet* as before.

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

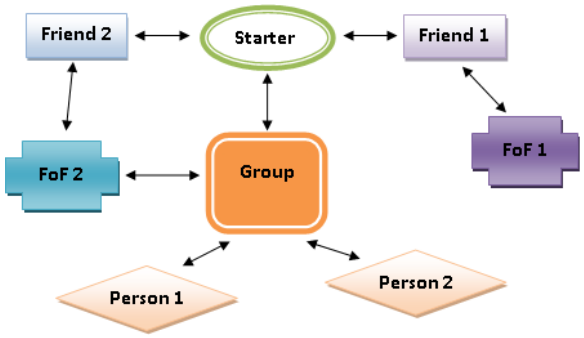

Fig. 7 CSSN Scenario 1 View

Scenario 1: Scenario 1 bring us a view of a Social Network, where the user Starter is connected to two Friends, which are connected to one FoF each and a group with three other people, where one of them is a FoF. The number of *Gridlets* is 7 and their processing time is of only 5 minutes.

For this scenario we assume that the client applications have "registered" in the Applications' Wall, that each of the group members has already accepted group 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 than 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 CSSN. In this scenario the requirements to process a *Gridlet* are "TotalCores=2; MemorySize=4078; Program=Gridlet.exe", and each CSSN client that processes the *Gridlet* is able to accept its conditions (however they still need to assess if they have idle cycles to spare). In the results for this scenario, as depicted in Fig. 8, we can see that the times to complete a Job were in the order of 11 minutes. Although, in Test 1 the user FoF2

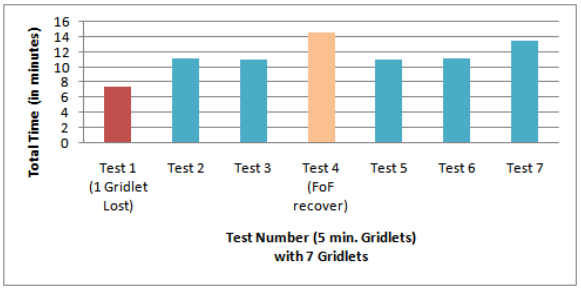

Fig. 8 Total times for Scenario 1

did not receive the last *Gridlet* as it was supposed to, and in Test 4 the user FoF2 crashed and recovered the last *Gridlet* in time to repeat its execution and complete it. These situations show that the total times will be hindered by the fact that people are not always in a *Away* state and also by giving more than one *Gridlet* to the same user, the Job will have longer completion times. However, we cannot always expect to find as many users as the number of *Gridlets* needed to complete a Job.

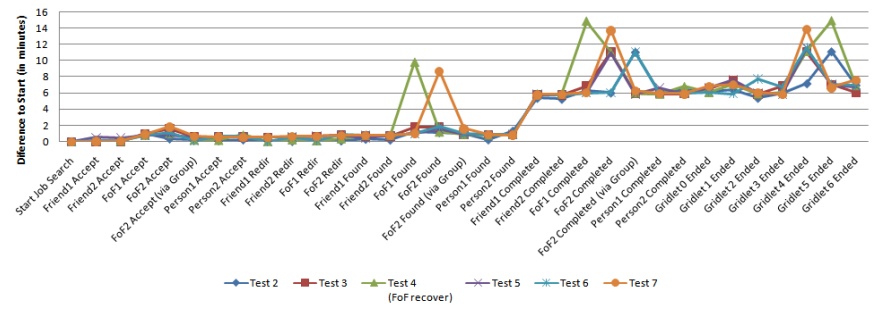

Fig. 9 Communication Times for Scenario 1

We can also see that the overhead of CSSN 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, since these times can vary with the Social Networks' traffic load at the time of use.

Figure 9 explains in detail how much time each task takes in relation with the starting point. It can take less than 1 minute for users' client applications to find and accept new Jobs. 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).

Scenario 2: Scenario 2 is an attempt to test CSSN in a more realistic environment, having a more complex network of users. As depicted in Fig. 10, we have two

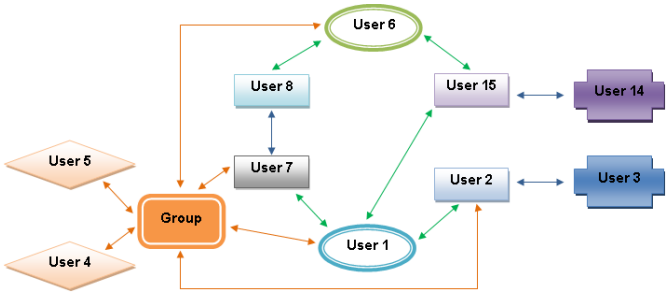

Fig. 10 CSSN Scenario 2 View

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 kind of users.

In this scenario, User 1 and 6 start a Job each, that contains 8 and 7 *Gridlets* respectively, making a total of 15 *Gridlets* to be processed by any of the users in this network. 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 each Wall (Group, Wall, Applications' Wall) that the Job request appears in. This means that, for example, 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; thus, in this network it could acquire four *Gridlets*.

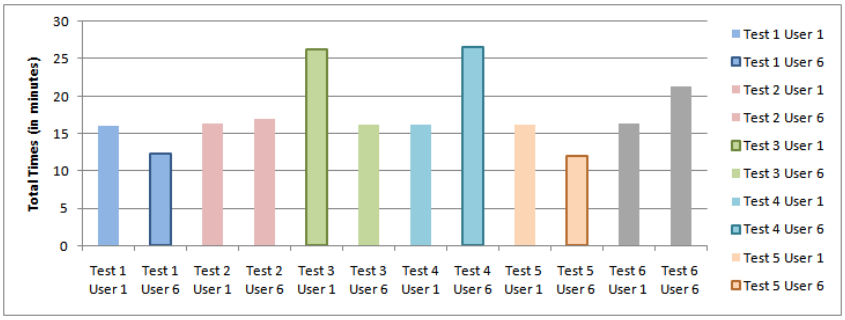

Fig. 11 Total times for Scenario 2

For this scenario we assume that each Job has less *Gridlets* than users that can be connected to a user submitting requests (e.g., User1 and User6), in order to simulate a larger network where the user could have potentially hundreds of connections, that could either accept or deny the request. Each client application must already be "registered" in the Applications' Wall, the group members already established and

the client application 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.

The results for scenario 2, as depicted in Fig. 11, bring us closer to understand how CSSN 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).

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 among the available users.

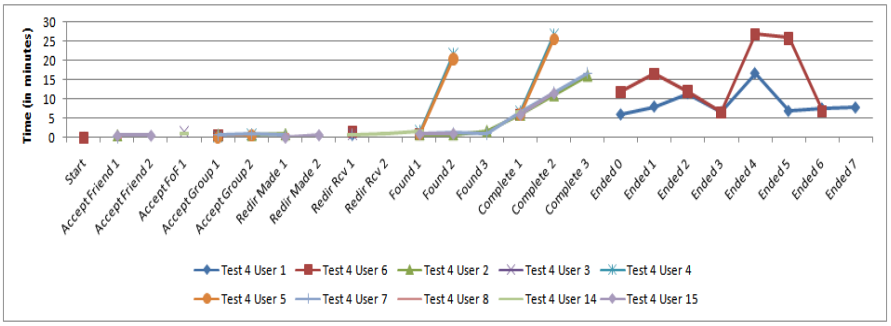

Fig. 12 Communication Times for Scenario 2 Test 4

In Test 4, as depicted in Fig. 12 we can see the added time due to the Social Network latency, where two of the *Gridlets* were retrieved only after all other *Gridlets* were already processed, and thus hindering CSSN performance.

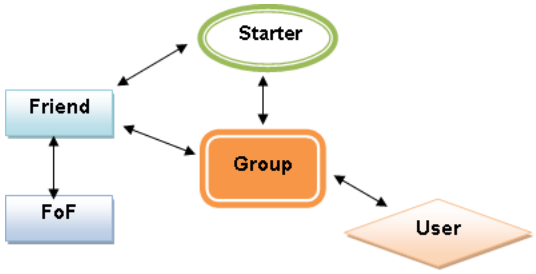

Fig. 13 CSSN Scenario 3 View

Scenario 3: Scenario 3 was designed in order to evaluate the performance with a real program that renders images. In this scenario, as depicted in Fig. 13 we have

one friend, one FoF and two users in a group (not counting with the user Starter) where one of them is the friend. The goal of this scenario is to know if CSSN can function with a real processing program, such as Pov-Ray, which is used in the tests. For each test the number of *Gridlets* to be completed is 4 and their execution times in the processing computers are undefined, as they depend 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 a direct transfer method to retrieve the data files in both ways (Starter to User and vice versa) for each test. Also, we assume that the client applications are running prior to the start of the Job.

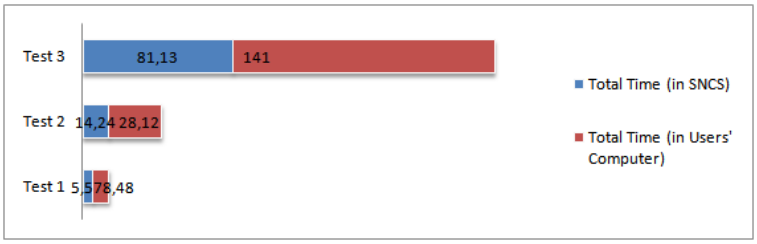

Fig. 14 Rendering Test Times for Scenario 3

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' "
and the program property as "pvengine64.exe", which in every CSSN 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.

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.

The results for scenario 3 confirmed that CSSN can gain speedups against local execution, as depicted in Fig. 14, where we have the total times of 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. 15 that although each task can take some time to execute, the average performance can be acceptable for *Gridlets* that have higher processing times.

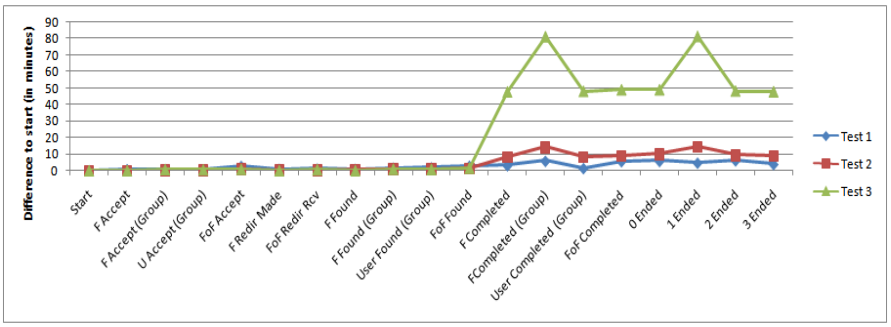

Fig. 15 Communication Times for Scenario 3

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 execute 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.

Discussion: When comparing with local execution, CSSN decreased the total processing time, compared to what it would have consumed in the users' computers, meaning that CSSN achieves overall speedups on Jobs.

We can also state that the overhead that CSSN imposes on the overall process is 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.

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 no longer needed, making a clean environment in Facebook, meaning that we can accomplish our goal of making CSSN 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 ac-

quiring 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, CSSN 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 Conclusion

In this project 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 idea of distributed computing has enabled other projects to create environments to execute common applications used by desktop computer users. Anyone can join or create its own network to share and receive idle cycles for its own usage. However, 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. This idea suites small networks within communities, or enterprises in order to gather idle cycles for common applications' execution.

Social Networks were a step forward for user interactions in the Internet, since 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. On these networks a user can reach another with just a few links; more, there is a small 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). This lead us to believe that we could utilize these networks for other purposes, other than messaging and interaction, much like using P2P networks for global distributed computing.

Our work describes Cycle-Sharing in Social Networks (CSSN), 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 CSSN is 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*.

CSSN main concern is to actually achieve resource and application discovery, while being able to perform resource sharing; thus, our works' primary 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. After a successful submission of a Job, the CSSN requesting client starts a search for resources that could meet the requirements of that particular Job. Moreover, it sends a Job Search message to the users' friends, to groups which could have the capabilities to process the Job, and also to the users' friends of friends (FoFs).

We evaluated CSSN with scenarios to determine how it would manage in such environments. Several scenarios were created in order to test CSSN regarding its performance, stability and viability for using a Social Network to achieve cycle-sharing, and resource and application discovery. These ranged from a simple one to derive speed-up and latency measurements, to more sophisticated ones with more diverse user's roles, and employing real applications such as **Pov-Ray**.

With the obtained results, we can conclude that while the total times for processing a Job gained speedups against local execution in the users' computers, this can be hindered by some variables: latency of Facebook servers, the fact that searching for resources among Social Networks users may not return positive results, and that the total number of available resources is less than the number of *Gridlets* that comprises a Job.

However, with functional 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 CSSN, Jobs are completed faster than in the user's computer, also releasing it for other tasks. The performance gains would increase with longer running *Gridlets* (more realistically about 1 hour) by amortizing overheads attributable to Facebook and communication.

We can conclude that our works' goals have been successfully met. It is possible to utilize a Social Network to perform 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 CSSN 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 project may contribute to the study and advancements made to novel cycle-sharing models.

Future work: In the future, we plan to augment the testing scenarios to address the issues of having a realistic environment, completing it with results of real peoples' usage and longer running *Gridlets*. We intend to extend the use of processing programs to include more common applications, such as video encoding, 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.

Also, the use of topic ontologies would greatly help in determining the number of users that may be able to help in a Job, while also focusing on those users that have more interest in such topics. Thus, CSSN could search for specific groups using the

Jobs' topics as a point of reference, in order to obtain the groups that would be more favorable to that particular Job.

Furthermore, we could extend the parameters of cycle-sharing to perform a form of advance scheduling: CSSN 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.

Moreover, to continue further development of CSSN and study on our works' goals, we plan to support other Social Networks, to perform cycle-sharing between the users. Also, we plan to substitute the need of having a stand-alone application, by embedding the CSSN client with the Browser, in order to gather resources and process *Gridlets* while the users are navigating through the Social Network or the Internet.

Acknowledgements: This work was supported by national funds through FCT Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2011 and project PTDC/EIA-EIA/102250/2008.

References

[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):56–61, 2002.

[AFG+10] 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. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[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*, pages 4–10. IEEE Computer Society, 2004.

[ASBS00] L.A.N. 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):11149–11152, 2000.

[ATS04] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, 2004.

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

[BTM07] F. Boldrin, C. Taddia, and G. Mazzini. Distributed Computing Through Web Browser. In *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.

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

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[FBJW08] R. J. Figueiredo, P. Boykin, P. Juste, and D. Wolinsky. Integrating overlay and social networks for seamless p2p networking. In *Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2008. WETICE'08. IEEE 17th*, pages 93–98. IEEE, 2008.

[FKNT02] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer, 35 (6)*, pages 37–46, 2002.

[FT02] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

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

[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. IEEE, 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.

[LSSV09] S. M. Larson, D. Snow, M. Shirts, and S. Pande Vijay. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Arxiv preprint arXiv:0901.0866*, 2009.

[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. *Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets-V), Irvine, CA*, pages 79–84, 2006.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proceedings of IPTPS02, Cambridge, USA*, 1:53–65, 2002.

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

[PFM+05] C. Papadakis, P. Fragopoulou, E. Markatos, E. Athanasopoulos, M. Dikaiakos, and A. Labrinidis. A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks. *Integrated Workshop on GRID Research*, pages 103–118, 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*, pages 161–172. ACM, 2001.

[RRV10] P.D. Rodrigues, C. Ribeiro, and L. Veiga. Incentive mechanisms in peer-to-peer networks. In *15th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS), 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2010)*. IEEE, 2010.

[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.

[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*, pages 32–38. ACTA Press, Calgary, 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*, pages 149–160. ACM, 2001.

[SVF08] J. Silva, L. Veiga, and P. Ferreira. nuboinc: Boinc extensions for community cycle sharing. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, SASOW '08, pages 248–253, Washington, DC, USA, 2008. IEEE Computer Society.

[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*, pages 61–66. 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.