Towards a Social Cloud Framework for Collaborative eResearch

by

Ashfag M. Thaufeeg

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Computer Science.

Victoria University of Wellington
2012

Abstract

Collaboration has always been an important aspect of scientific research. The coming of internet opened the doors for greater levels of collaboration for the research community, first enabled by email and then by web 2.0 based online portals called VREs. A new force, social networks, are bringing a paradigm shift to online research communities. Social networks could foster a more vibrant research environment powered by social activities such as sharing, community creation, tagging and community groups.

This thesis explores the idea of using the power of social networks to create a social cloud to contribute and share computing resources. The prototype implementaion, called the Social Collaborative Cloud (SoCC), uses facebook as the underlying social network. The prototype was evaluated using simulations of both real and synthetic datasets, as well as real world tests.

Acknowledgments

First I would like to thank my supervisor, Dr Kris Bubendorfer, for his guidance throughout the thesis.

I would also like to thank Dr. Andy Linton for his help in configuring out the test machines used for the implementation, and my colleague Koshy John and Kyle Chard for their helpful suggestions.

Last but not least, I would also like to thank Julia Harrison of Victoria International Office for her support throughout my stay in New Zealand.

Contents

Chapter 1

Introduction

Sharing and collaboration has always been important aspects of scientific research. Whereas in the past the only opportunity for productive communication exists when scientists visits other institutions or meet in conferences, the internet has brought a paradigm shift in this area. Electronic communication and services such as email and online file sharing services has allowed scientists to conduct collaborative research spanning institutional and national boundaries.

We are now at the point of another paradigm shift in scientific collaboration, brought about by the advent of social networks. The use of social models is promoting rapid sharing and collaboration at a level not seen before ([33]), leading to productivity. Online research communities such as myExperiment ([21]) and nanoHUB.org ([30]) has allowed sharing of datasets and documentation, tagging and commenting on research artefacts and collaborative research. These websites attempts to serves as a portal where scientists can conduct research. In other words, the provide a Virtual Research Environment (VRE) for scientists.

Although most of these VREs has brought with it greater sharing and collaboration, they still do not go far enough to harness the real capabilities possible with a social model. Commercial social networks such as facebook and LinkedIn, although not specifically scientific research communi-

ties, have fostered a more vibrant community of people from all fields and age groups. It therefore provokes the consideration that these commercial networks could be utilised as a VRE.

Another important development is the cloud computing model. High Performance Computing (HPC) traditionally has been provided by institutional grids. The internet has allowed the online delivery of grid services such as TeraGrid ([20]) and commercial cloud services such as Amazon EC2 ([2]). HPC computing capabilities are still a scarce resource. This has given rise to volunteer computing efforts such as BOINC ([14]), which enables the general public to volunteer their spare computing capacity to scientific research.

Some VREs, such as nanoHUB.org, utilise these cloud computing services to deliver specific capabilities directly within the VRE environment, such as running simulations and modelling. More recently, the Social Cloud concept ([18]]) attempts to address this problem by enabling computing resources to be shared using social relationships found in a social network. This model utilises the social relationships between parties as a basis and motivating force to encourage the sharing of computing resources.

The goal of this thesis is to utilise the capabilities of social networks to implement a socially motivated, computing resource sharing framework. In particular, the VRE capabilities of social networks will be used to integrate a social cloud with a virtualised resource sharing framework, called the Social Collabortive Cloud or SoCC. The result of the thesis would be a working prototype of SoCC integrated to a commercial social network.

1.1 Contributions

The main body of work for this thesis is the design and implementation of a prototype Social Cloud framework integrated into a social network. Specifically, this thesis makes the following research contributions:

- Design and build a prototype Social Collaborative Cloud. The prototype should satisfy the following requirements:

  - *Integrate seamlessly with the social network environment.* In order to facilitate ease of use and acceptability by users, the prototype must integrate well with a social network. This means that information such as usernames and email addresses must not be duplicated (it has to use what the user has provided to the social network), the users privacy settings must be respected, and should not duplicate any social network functionality, such as builtin user groups and chat boards.

  - *Implement a IaaS/PaaS cloud model.* The SoCC prototype is aimed at scientific researchers, whose primary need is access to raw computing resources. SoCC provides a combined Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) cloud computing model to serve this need. It provides the basic resource fabric that can be customised to run a set of Virtual Machines (VMs) of a required specification such as CPU, Memory and storage (IaaS model). SoCC also provides the option to launch a pre-configured VM with a selection of Operating Systems and software packages and use it run computations (PaaS model).

  - *Use Virtual Machines (VMs) for resource sharing.* VMs provide a number of advantages over traditional grids, which is investigate in more detail in section 4.6. The prototype must implement an easy and straightforward mechanism for users to instantiate and use VMs, while preserving the advantages of using VMs.

  - *Implement a scheduling framework that encourages sharing.* The main purpose of the Social Collaborative Cloud is to encourage collaboration among scientists and sharing of computing resources. To this end, the SoCC prototype must implement a

way that would encourage participating users to contribute resources, while giving them more in return.

  - *Be open.* Openness is viewed as a basic requirement of any VRE ([21]). To this end the SoCC prototype must use open protocols and standards as much as possible. It should also provide a way to extend its functionality to cater advanced users or users with very specific requirements.

- Evaluate the performance of the SoCC prototype. Two sets of tests were done. The first set is composed of the real world tests intended to evaluate the workings of the prototype, and to investigate the impact of the use of a social network has on the implementation. The second set of tests evaluate the performance of the scheduling algorithms via a simulation. The simulation uses a real world grid dataset adapted to be used in the SoCC environment, as well as a synthetic trace to evaluate extreme cases.

1.2 Thesis Organisation

The thesis is organised as follows: Chapter 2 explores some background knowledge that would form the foundation of this work. Chapter 3 looks at related work, and chapter 4 details the architecture and implementation of the SoCC prototype. Chapter 5 presents the real world and simulation tests to evaluate the performance of the SoCC prototype. Chapter 6 concludes this thesis and suggests areas for future research.

1.3 Related publications

The following publication outlines the initial work done on this thesis. I was the first author for this paper, and is co-authored with my supervisor Kris Bunendorfer, and Kyle Chard from University of Chicago.

- Ashfag M. Thaufeeg K. Bubendorfer K.Chard. Collaborative eResearch in a Social Cloud, 7th IEEE International Conference on e-Science in Stockholm, Sweden, December 2011.

Chapter 2

Background

This chapter looks at the background knowledge and concepts that serves as the foundation for this thesis.

2.1 Service Oriented Computing

2.1.1 Overview

Traditional computing approaches consists of each organisation implementing and maintaining their own computing infrastructure. In the early days of computing before the explosion of the Internet, most organisations that require a computing capability invested heavily in their own data centres with servers and networking equipment, technical staff and in many cases their own software developers as well.

However, with the connectivity and opportunities that came with the Internet, computing infrastructures, and IT in general, has evolved into specialised services [38]. As a result, it is common to find computing capabilities being delivered over the Internet, more generally referred to as Service Oriented Computing (SOC).

Due to the wide range of computing services that are possible, from raw infrastructure capabilities to software, it is possible to partition SOC

into three major layers, also known as the cloud stack ([29]), as shown in figure 2.1.

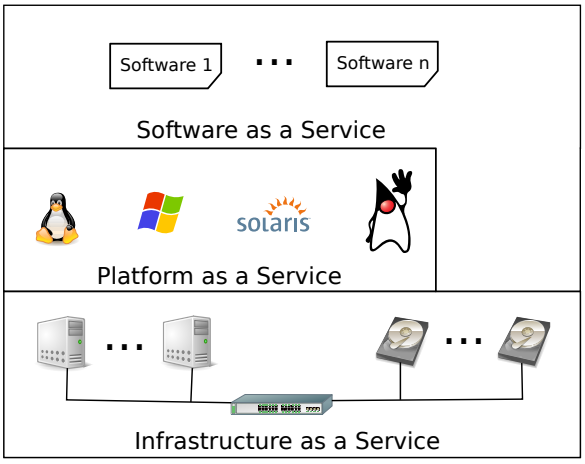


Figure 2.1: Service Oriented Architecture Layers.

Some definitions only refer to the lower stacks as the cloud [15], mostly due to the fact that the software stack has been around for some time (referred to as Software as a Service), while the lower stacks are a more recent innovation.

2.1.2 Infrastructure as a Service

The lowest layer of the service stack is the infrastructure layer. At this layer the capabilities of the computing hardware, generalised into networking, storage and computing, is exposed to the client in the form of Infrastructure as a Service (IaaS). It is important to note that it is very uncommon for the raw hardware to be exposed to the client. Rather, the current trend is to expose a uniform view of the infrastructure layer to the client by using resource virtualisation [31]. This form of virtualisation allows the service provider to hide the heterogeneity of their infrastructure (eg: different server models) and present the client with a standard and uniform view.

2.1.3 Platform as a Service

The platform layer is an intermediate layer in figure 2.1. This layer consists of software implementations, so there is a certain overlap of this layer and the layer above (the software layer). However, the common view of this layer is that any software that is used to host other software or serves as an execution environment to other software [32]. With this definition, web hosting software such as Apache and execution environments such as Java Virtual Machine can be viewed as a Platform as a Service (PaaS). However, more common to be served as PaaS are Operating Systems (eg: Ubuntu, Microsoft Windows) which serves as a platform for virtually any other software.

2.1.4 Software as a Service

This is the top most layer of the service stack, and its common definition is any software that is directly consumed by the client[32]. Note that depending on the client user, the definitions of PaaS and SaaS can be different. For example, a web developer may view the Apache web hosting software as in the SaaS layer, while a user running a blogging software most likely would view Apache in the PaaS layer. Most current cloud services fits into this layer, such as web based email, online photo galleries and video sharing.

2.2 Social Networks

2.2.1 Overview

A Social Network, as the name suggests, is an online community of users linked to each other by social relationships. Social relationships here refers real world relationships such as friendship, kinship, professional acquaintance etc. In this sense, the Social Network represents a graph with the

users being the nodes and the edges being their relationships.

There is no rule as to what a node in a Social Network can be. It could be an individual user, an organisation or even a country. For brevity, in this section we would only refer to nodes as individuals, but it should be remembered that node and relationship heterogeneity still applies.

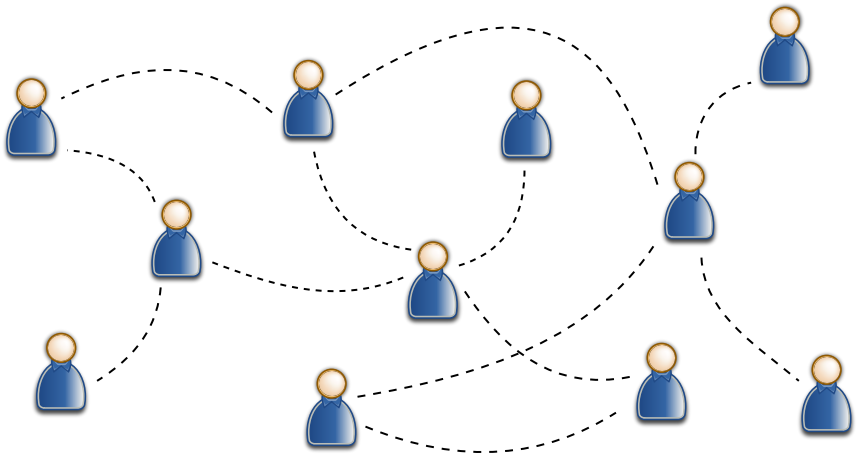


Figure 2.2: Social Network - Individuals and Relationships.

2.2.2 Basic Concepts and Features of Social Networks

1. **Relationships**
   A relationship in a Social Network is a connection between two individuals, or in another words, an edge in the social graph. This connection could be of many natures. For example, family relationship, friendships, a common interest or common professional fields are all possible relationships between individuals. In general a social network would represent all these relationships with just one type of an abstract link between two individuals, which is referred to in most Social Networks as a friend connection.

   Although its not possible to differentiate between different relation-

ship types (eg: professional relationship or family relationship) in general, some Social Networks does allow a user to organise individuals relationships into groups (eg: friend circles in Google+), thus allowing the links to be differentiated as far as that individual is concerned. It should be noted that these relationship differentiations makes sense only in the context of a particular individual, as different individuals may organise the same relationship differently.

It could argued that relationships between individuals are the fundamental feature of Social Networks, as relationships gives rise to the various other properties of Social Networks.

2. **Personal Social Network**
   An individuals Personal Social Network comprises of all the direct relationships of that individual (figure 2.3). This network forms a sphere of influence for that individual - i.e he or she can influence, via the direct relationships, other individuals in the Personal Social Network and vice versa.

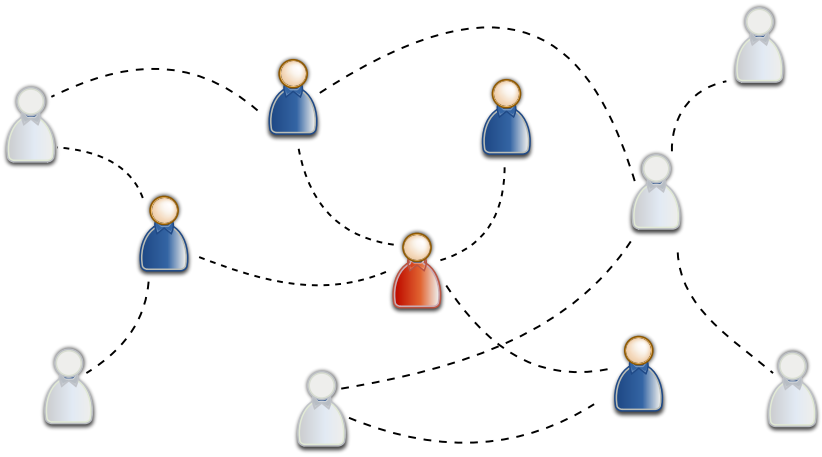


Figure 2.3: An individuals Personal Social Network.

Personal Social Networks are important as they allows relevant in-

formation to spread among individuals that has something in common. For instance, a piece of information shared by a scientist about some research work would be received by members in that scientists Personal Social Network, who are likely to contain other scientists with similar interests. The information could then get shared further through those scientists Personal Social Networks. The same mechanism can also work in the opposite direction. For example, if enough members in an individuals Personal Social Network joins a group, then that individual might be pressured to join as well.

3. **Social Incentives**
   Social Incentives refers to the forces that motivate individuals to share or contribute in a Social Network. The nature of the incentive would depend on such factors as what is being shared, the nature of the relationship or trust between individuals etc. For example, an individual might share a photo gallery with family members only for personal satisfaction without expecting anything in return. Another individual might share a photo gallery with the whole community, expecting others to commend his work and as a result raise his social standing.

   Often, the required incentive is related to the level of trust in a Social Network[35]. Trust is an abstract term referring to how willing an individual is to share something with another individual. Varying degrees if trust, or levels, might exists in a Social Network as shown in figure 2.4.

   An individual might not require more than personal satisfaction or moral responsibility to contribute or share something in his or her Personal Social Network. If sharing is to happen outside the Personal Social Network, then a more elaborate incentive might be required, such as a monetary reward or a mutual benefit.

   Levels of trust might also vary within the Personal Social Network.

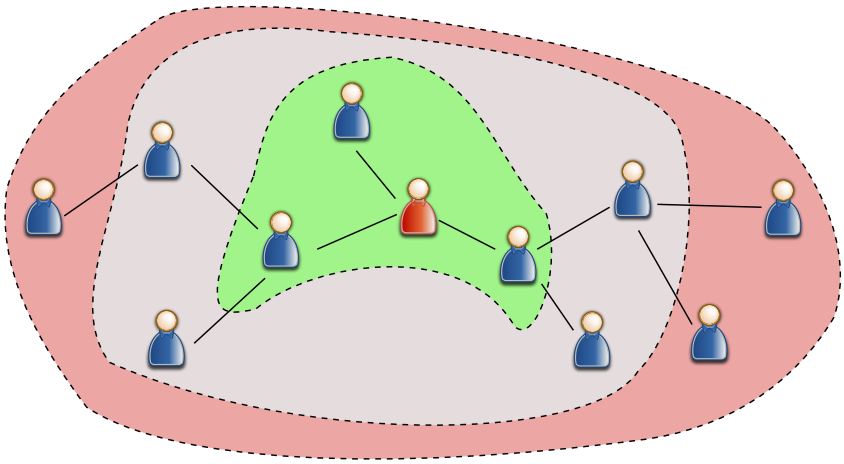


Figure 2.4: Levels of trust in a Social Network. An individual is more likely to trust another individual who is closer, while far away individuals are not as trustworthy.

For example, an individual might trust family members more than friends, and may trust friends more than professional acquaintances.

2.2.3 Development APIs

An important feature in today's commercial Social Networks is the built in developer support. Almost all Social Networks provides some form of an API (Application Programming Interface) to allow individuals to extend the functions of the Social Network in some manner. This feature also serves as a magnet to attract more users to the Social Network, by leveraging the skills of third-party developers to integrate interesting applications, such as games. However, the most useful aspect of the development APIs of Social Networks is that it can be used to integrate Social Network functions into other applications.

For reasons detailed in section 4.2, facebook was selected as the test bed Social Network for this thesis. Therefore, we will look at the face-

book developer API and dvelopment model in this section, although the fundamentals are the same for all Social Networks.

Figure 2.5 shows the high level architecture of the facebook application model. A developer can signup to the facebook developer website and get a unique key for each application they want to develop.

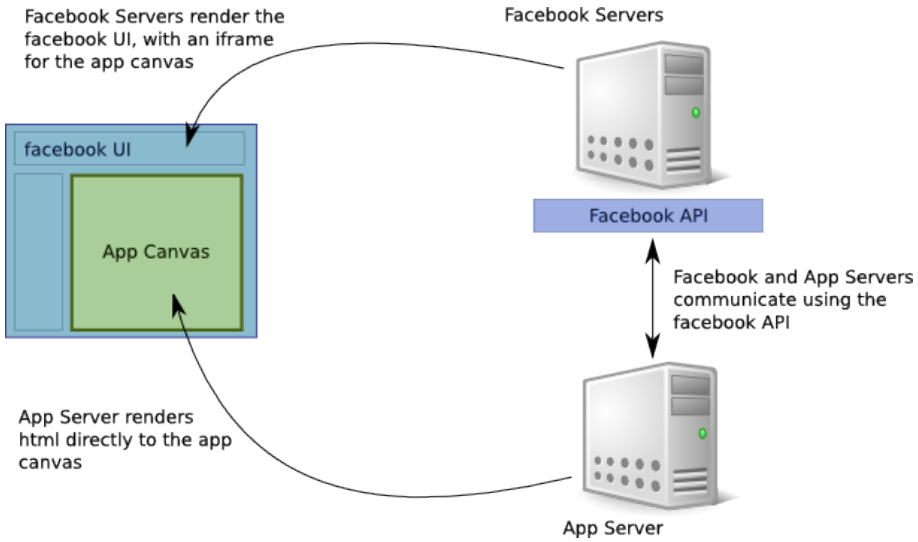


Figure 2.5: Facebook application model.

As shown in the figure, the application is nothing more than a piece of standard html (a web page) rendered inside an iframe within the facebook UI. The app will require a backend server (App Server) to generate the html. When the developer registers his app, facebook will save the URL for this App Server which would form that applications link inside the facebook UI. When a user clicks this link, the app is opened inside the iframe.

In order for the developer to have access to the users private information, such as the users email address, the user would first need to authorise the app for this information. This happens the first time the user opens a facebook app. Thus, an app would not have any information about a user

that has not been authorised by that user for that app.

2.3 Virtual Research Environments

2.3.1 Overview

Traditional means of collaboration between researchers (prior to web 2.0) consists mainly of email exchanges and face to face interaction in conferences. With the popularity of web 2.0, new modes of collaboration that takes advantage of web 2.0 features such as social networks, online data sharing and real-time messaging has become possible.

In fact, web 2.0 technologies has advanced to a level that it is now possible to provide a complete web based environment that allows efficient project management, communication, sharing and other services amongst researchers. A Virtual Research Environment (VRE), as its name suggests, is a web based interface that provides integrated access to these services.

2.3.2 Functions of a VRE

The main functions of a VRE, according to definitions from [21, 32] can be described as follows:

1. **Management of research artifacts**
   Research artifacts refer to documents, datasets, computer programs and any other computer stored object that is related to the research. A VRE must have the facilities to create, update, share and search such artifacts.

2. **Provide a social model for interaction between researchers**
   A VRE must encourage cooperation and sharing. It should implement features found in Social Networks such as community feedback and discussions, providing credit to researchers who shares the most, establish relationships with other researchers etc.

3. **Provide an open environment**
   A VRE must implement standardised APIs and development models. Make it easy for developers to extend the capabilities of the environment.

4. **Make research artifacts actionable**
   This means that research artifacts such as datasets and experiment descriptions arent merely stored or archived in the VRE. Rather, they can be executed by invocating remote services. In other words, research can be conducted directly from the VRE.

Chapter 3

Related Work

3.1 Fire Dynamics Simulation Demonstrator

This is a facebook interface for the Fire Dynamics Simulator (FDS) simulation package ([19]). Its interest here is because of its use of facebook to provide users access to a backend grid configured with the FDS software package. The demonstrator is built as a facebook applicattion (figure 3.1), allows users to submit simulation jobs to the backend grid and download results once the computations are completed. Being built into facebook, users are able to share datasets and results, participate in others experiments and discuss reasearch. This is a very specific, focused facebook application that allows users to submit FDS jobs to a specific grid. Both the grid and FDS software packages are pre-configured by the developers.

3.2 myExperiment

myExperiment([21]), shown in figure 3.2 is a Virtual Research Environment for scientists, focused on sharing scientific workflows and related materials. The VRE implements a social model, which allows users to upload content and share with others, enables scientists to form groups for

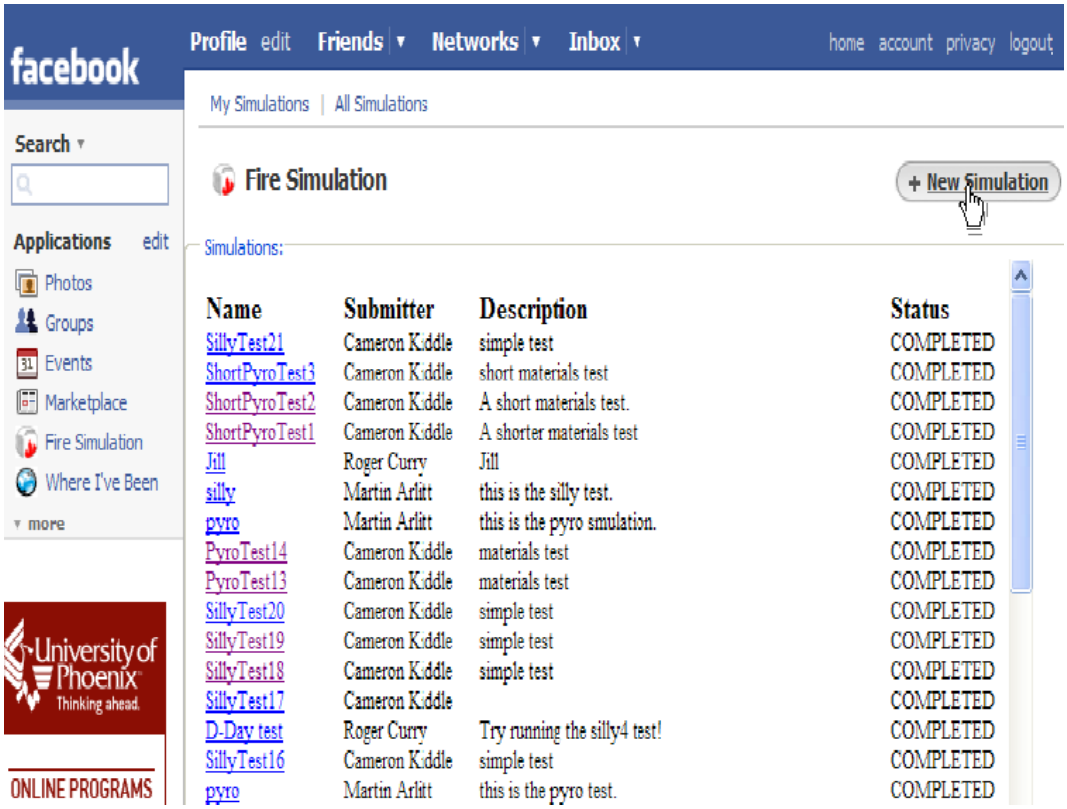


Figure 3.1: Fire Dynamics Simulator facebook interface.

conducting and collaborating on research etc. It also allows users to define *actionable reserach objects*, which are research materials such as datasets and workflows which can be submitted to external services for processing right from the VRE interface. myExperiment has built in support for some workflow management packages, and also provides a public API for developers to extend its functionality.

3.3 nanoHUB.org

nanoHUB.org ([30]) is an online portal for the nanotechnology reserach community. It provides a repository of software tools, research and teach-

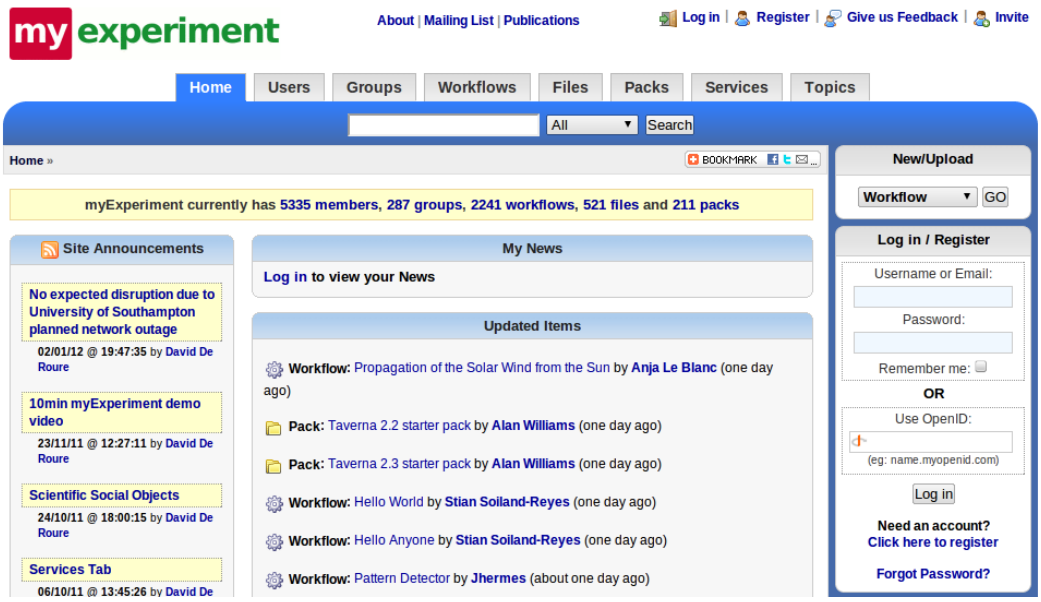


Figure 3.2: myExperiment VRE.

ing materials focused on the nanotechnology field. nanoHUB supports basic community focused fuctions such as user uploaded tutorials and reserch materials, tagging and ratings.

The most interesting feature of nanoHUB.org is archive of browser based simulation tools. These tools allow users to run simulation software inside a browser window, with the actual simulations powered by public grids. nanoHUB.org currently has a very active user base, which as of 2010 stands at 167,196 users([26]).

3.4 Social Cloud Computing

3.4.1 Overview

From [17], the definition of a Social Cloud is as follows:

> *A Social Cloud is a resource and service sharing framework util-*

ising relationships established between members of a social network.

Therefore, the fundamental feature that distinguishes a Social Cloud from other forms of cloud computing is the social relationships between members of the cloud. These relationships comes from the social network from which the Social Cloud is overlaid, as shown in figure 3.3.

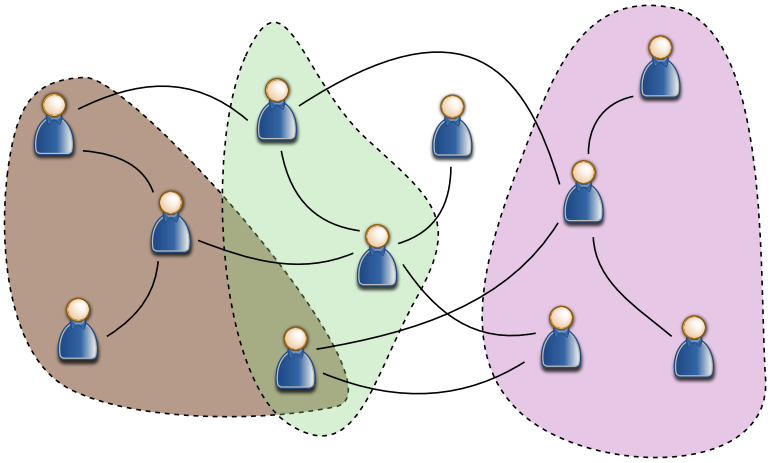


Figure 3.3: Multiple Social Cloud overlays over a Social Network.

Social Clouds are similar to volunteer computing models. In volunteer computing, users donate their idle computing capacity to public projects, such as SETI@Home [13], without expecting a monetary return. The motivations for such contributions can be either a sense of community good will, or the high status resulting from contributing more than others.

Social Clouds takes advantage of the relationships in the social network to form the necessary trust and motivation for contribution. Trust provides the primary motivation for users to risk allowing other members in their personal social network to access their resources. Social standing, such as recognition in the community as well as mutual benefit are also motivations for contribution.

3.4.2 Social Market

At the centre of the Social Cloud is the the Social Market. The Social Market regulates and manages the resource tradings that occur in the Social Cloud. It consists of the protocols and economic models selected to enforce a particular market behaviour.

The specific protocols and economic models of the Social Market depends on the nature of the resource been traded, the type of relationships between members and the level of trust between them. Taking account these factors, several models are possible. The following lists some socially motivated market models ([18]):

1. **Volunteer.** In this model, users selfishly share their resources with other members. The motivation behind this model is an idelistic one, such as charity. This is a high risk model, where users are generally not held responsible for bad behaviour.

2. **Trophy.** In this model, users contribute in return for a special status in the community. This can take the form of a badge in the users profile, or a ranking in the community to recognise the users contribution. The driving force in this model is fame, where users expect a higher social status in the community in return for their efforts.

3. **Reciprocation.** In this model, users are rewarded with higher priority when requesting resources depending on how much they themselves have contributed. Users who contribute more are favoured over those that contributes relatively less. The motivation behind this model is the need for more resources due to the scarcity of the resource among individual users possession. The market ensures that if the user contributes, then that user would be prioritised when requesting resources for themselves.

4. **Reputation.** This is a model that depends on individual users reputations when allocating resources. Reputation is a social status a user

earns from the community. Users who do not cheat, interact with the community more or otherwise shows good social behaviour would have a higher reputation. Users with higher reputation are favoured by the Social Market when resources are allocated.

3.5 Social Storage Cloud

The Social Storage Cloud (figure 3.4) is a prototype storage service implemented on facebook ([18, 17]). The service is implemented as a facebook application, and uses a virtual currency based social market to trade storage services (figure 3.4). The service supports both posted price and auction mechanisms. When users sign-up to the application, they are provided with a Globus Monitoring and Discovery System (MDS) identifier which they can use to configure their own storage services. The storage service can then register with the Social Storage Cloud and sell storage space.

3.6 The Social Cloud for Public eResearch

This is a volunteer computing framework built on top of facebook ([23]). It uses the BOINC [4] project as the underlying volunteer computing service. A facebook application connects the BOINC servers, facebook, users and projects (figure 3.5).

The framework uses the facebook relationships to promote volunteer computing projects among a users friends. It uses a number of social incentives, such as Project Champions and Compute Magnates ([23]) to encourage users to contribute.

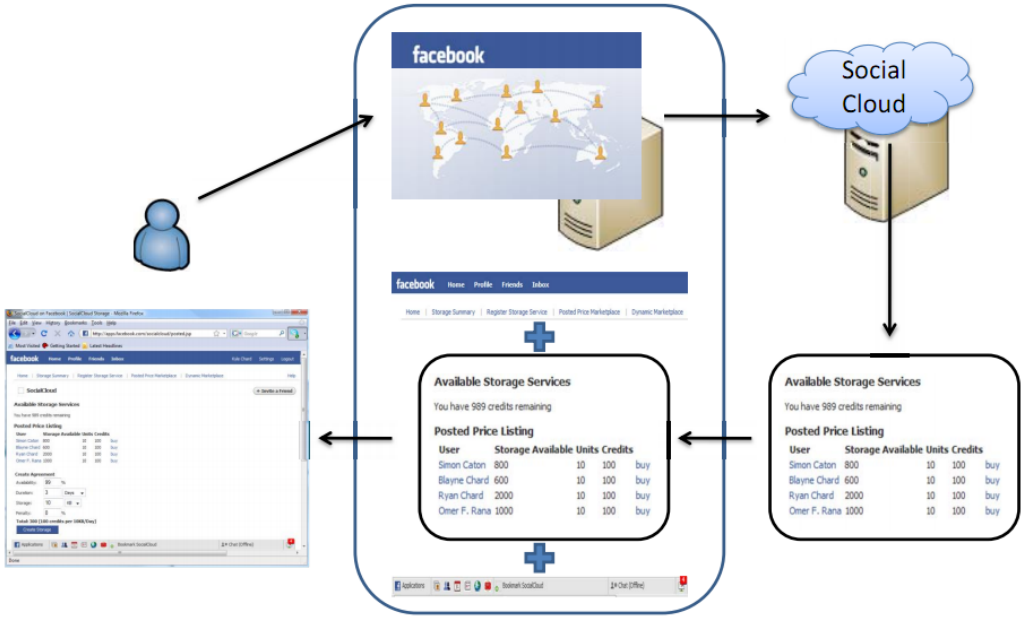


Figure 3.4: Social Storage Cloud.

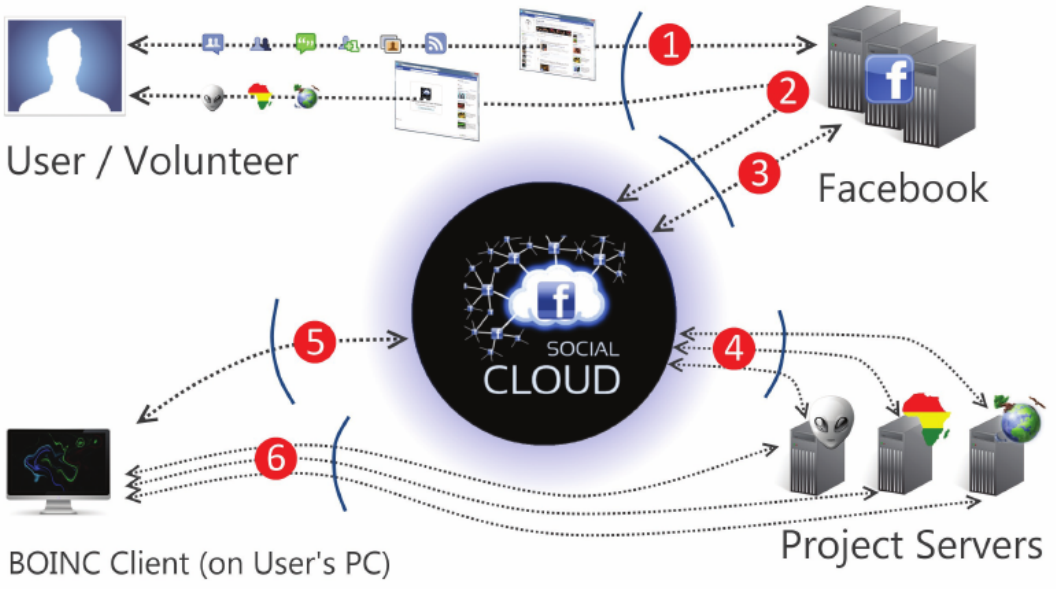


Figure 3.5: Social Cloud for Public eResearch.

Chapter 4

SoCC Architecture

This chapter details the architecture and implementation details of the prototype Social Collaborative Cloud. We first give an overview of the design followed by some of the design decisions behind choosing facebook for the underlying Social Network and use of Virtual Machines for resource sharing. We then look at how a VRE can be implemented using the facebook interface, and how SoCC can be implemented as a extension to the basic VRE. The rest of the chapter details the implementation of SoCC.

4.1 Overview

SoCC is intended to fuse the worlds of social networks and cloud computing. The goal is to harness the power of social networks, such as its inherent friend relationships and collaboration capabilities, to provide a VRE environment that would allow computing resource sharing.

To this end, a primary design goal of SoCC is to not break the underlying social network model in the process. This is achieved by storing as much information in the social network, and only storing the information related to the computing resources and accounting is stored in the SoCC application server. No existing functionality in the social network is duplicated – integration is prioritised when ever possible. Users would use

the social networks features to carry out essential activities such as document sharing, group discussions and forming relationships. Users would also the social network for authentication. Only the social cloud feature is implemented by SoCC.

For example, SoCC would use the social networks group feature to implement Virtual Organisations. This would allow users to form VOs by creating groups in the social network using a familiar interface. Any changes that is made to the social networks group would be reflected in the corresponding SoCC VO, such changes to member permissions. This model allows SoCC to feel as part of the social network, not as an external application with a interface inside the social network.

Another consideration is that in this SoCC prototype we are targeting scientific researchers. Their needs are such that low level computing resources are preferred in order to run CPU intensive algorithms and process large datasets. SoCC therefore provides both IaaS and PaaS models of cloud computing. The PaaS model would allow scientists to quickly deploy a pre–configured VM image to run their computations, while the IaaS model would allow the VM to be customised with the required CPU power, memory and storage to run whatever software tools they require.

4.2 Design Decisions

4.2.1 Use of facebook for the prototype

When deciding on a Social Network to build SoCC, the two major Social Networks, LinkedIn and facebook was considered. LinkedIn appeared attractive because of its orientation for catering towards professional users and thus appeared more suited for the target audience of SoCC [9, 5]. However, facebook had the better developer support as a result of its success, and already boasts millions of applications [3,4] and a very active developer forum. Also some of the related work has been already done

on facebook [12, 17]. For these reasons facebook was selected to build the SoCC prototype.

4.2.2 Use of Virtual Machines for resource sharing

The advantages of Virtual Machines for resource sharing will be discussed in detail on section 4.6. Coupled with the fact that major cloud providers are already using this model ([2, 6, 7]) and already many scientific work are being done on VMs ([36, 24]) it was decided to use Virtual Machines as the resource sharing model for SoCC.

4.3 Implementing a VRE Over Facebook

Facebook already serves well the extensible and developer support requirements of a VRE as discussed in section 2.3. It has a very extensive developer support network and provides very intuitive APIs to make available the user account and social information to third party applications. Figure 4.1 shows the main interface of facebook with important areas in the interface labelled.

When viewed as a VRE, the labelled areas in figure 4.1 implements the following functions.

- **Area 1:** This is the main menu of the facebook interface and provides access to social functions, such as searching for friends and organising events. This menu serves the same functions as a VRE.

- **Area 2:** This menu is the extension point to add functionalities to the VRE. It shows the apps currently added by the user. SoCC would be built as a facebook app and would be just another addition to this menu. More VRE functions, such as data analysis tools and document sharing can be built as facebook apps and added here.

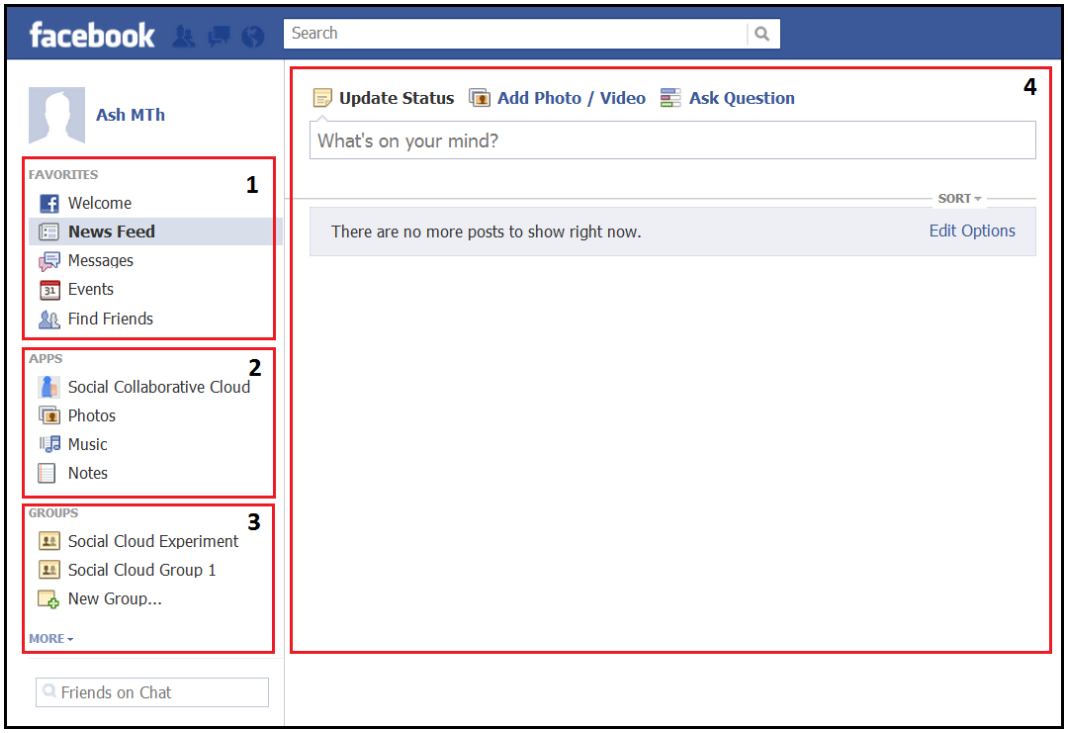


Figure 4.1: Facebook main interface.

- **Area 3:** This menu shows the groups the user is a member of. This feature would be used to implement Virtual Organisation functions of the VRE and is discussed in more detail in section 4.5.3.

- **Area 4:** This is the main content area of the facebook interface. Apps (implementing VRE functions) can be configured to open in this area giving the user a sense of integration with the Social Network.

4.4 Architectural Components of SoCC

The logical components of the SoCC is shown in figure 4.2.

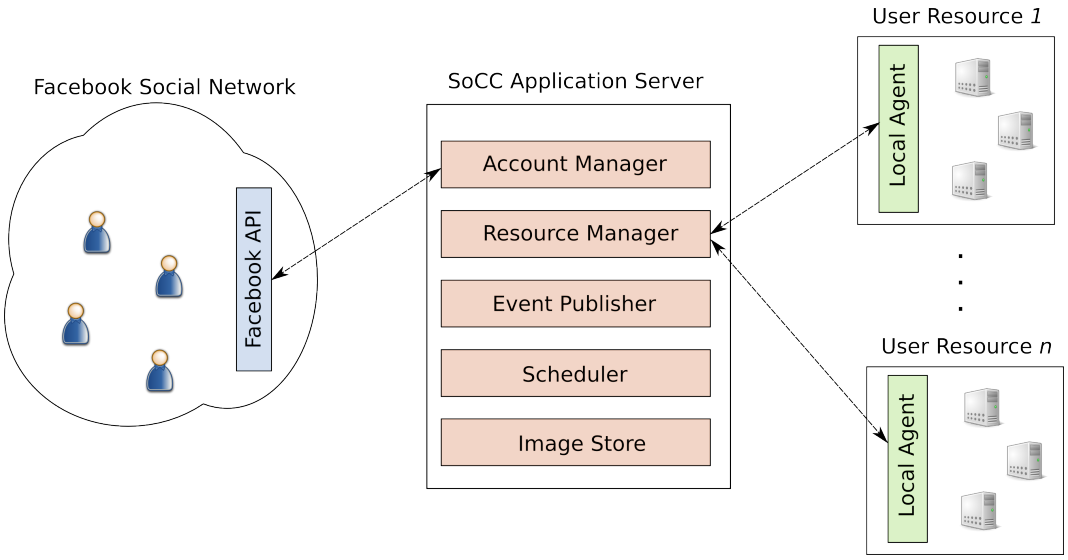


Figure 4.2: Architectural Components of SoCC.

4.4.1 SoCC Application Server

This is the server that hosts the facebook application code and the Account Manager, Resource Manager and Scheduler components.

4.4.2 Account Manager

This component interacts with the facebook public developer API. It handles the user creation and VO creation functions. It's address is provided as the end-point when registering the SoCC app in facebook. The account manager stores locally only minimal information about the user, namely the facebook user id and email address, so as to not duplicate any information.

4.4.3 Resource Manager

This component does all the interactions with the user resource. It's address is exposed to the User Agent hosted on the user resource. It handles the submitting of compute requests and receiving status updates from the User Agent about the resource and hosted computes. It exposes a REST interface to the Local Agent hosted on the user resource to push this information. The Resource Manager stores information about the user resource, such as the address of the Local Agent, available capacity, currently hosted computes and their status etc.

4.4.4 User Resource

The user resource is the computer cluster (here on referred to simply as user resource) registered by the user. It contains standard server or computers capable of hosting compute instances. The Local Agent located on the resource handles all interactions with the SoCC Resource Manager, and is required to implement a REST interface to create, update and terminate compute instances, as well as to push status information to the Resource Manager.

The Local Agent is a very light weight REST interface. The user only have to implement REST calls for receiving compute requests, resource availability and status queries. In most cases the user would only need to write a plugin for whatever Virtualisation Management Middleware that is being user.

Users do not need a large cluster or high-end machines to contribute resource. The SoCC prototype uses a cluster of 3 machines, with the Local Agent hosted on one of the machine. This cluster was created using the OpenNebula Virtualisation Management Middleware ([11]), with a couple of custom ruby scripts to implement the Local Agent.

It is also possible to make the Local Agent a proxy for a third party cloud. For example, Amazon EC2 provides an API for users to program-

matically deploy and monitor instances on its infrastucture. The Local Agent could be written to deploy instances on Amazon EC2 and forward the IP addresses and status updates to SoCC Resource Manager. In the future, such a Local Agent can also be made available for as a downloadable tool for SoCC users, to ease the process. This would be an alternative option for users that do not have the infrastructure themselves to contribute resources.

SoCC does not place too much restrictions on the specification of the hosts in the user resource. To handle the heterogeneity, the Local Agent is required to report the resource capacity in terms of compute units. This is similar to how Amazon EC2 [12] exposes its resources. In the SoCC prototype, this is achieved by a simple benchmarking using linpack to measure the processor performance. Using a compute instance of 1 virtual CPU and 1GB RAM hosted on a Intel Quad Core i5 host, the linpack reported a score of 25 GFlops. For the prototype this score is defined as one compute units performance. However, standardising the compute performance across user resources is a difficult exercise. Amazon EC2 itself, which has been providing cloud services for a long time, recommends users do their own benchmarking to measure the expected performance from their instances [1].

To simplify requesting and hosting compute instance, SoCC defines compute types similar to Amazon EC2. A compute type represents a specification (eg: 1 Virtual CPU, 1 GB RAM, 5 GB storage) of a compute, and is labelled small, medium and large depending on its resource requirements. When submitting a computes request, a user selects the required compute type from a dropdown menu instead of manually specifying the resource requirements.

4.4.5 Scheduler

This component handles scheduling of compute requests. It decides when to accept or reject requests. The SoCC prototype implements a fair share scheduler, detailed in section 4.7.

4.4.6 EventPublisher

The Event Publisher provides an extension point to the SoCC framework. It publishes events related to the compute instances running in the cloud, such as when an instances is booting, running or terminated. When users make a compute request, they have the option of supplying a REST endpoint that can receive the event notifications from this component. The real world evaluation tests in section 5.1.2 shows how to make use of this feature to automate creation of a virtual cluster in SoCC.

4.4.7 Image Store

The Image Store contains the standard set of images for computes. These images are meant to provide an OS images customised for common scenarios such as running simulations or hosting a web server. Using these images minimise the time to deploy an instance, as these would already be cached in the user resources so there is not image transfer overhead. Users also have the option to user their own custom image as well, but with the tradeoff of requiring additional time to transfer their image over the network.

4.5 Interactions in the SoCC

This section details the interactions that occurs between the various components of SoCC whan carrying out common use cases .

4.5.1 User Registration

Figure 4.3 shows the interactions that occur during user registration.

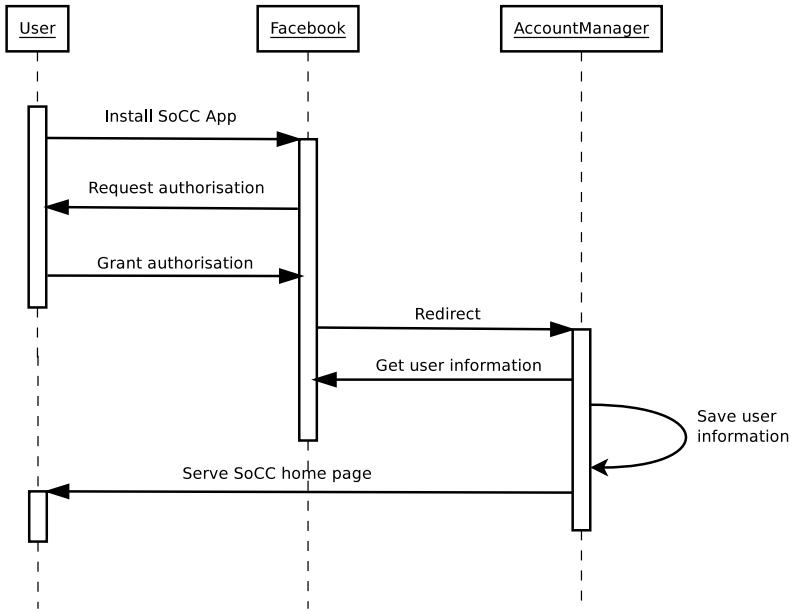


Figure 4.3: First time user registration.

The user uses the standard facebook mechanism to search for the SoCC application and selects to install it. SoCC requires users to expose their facebook id and facebook group information. Facebook Id is required to identify the user uniquely. The group information is required to handle the VO management. After the user authorises this information to SoCC, facebook forwards the request to the SoCC Account Manager which persists the user id in its local database.

4.5.2 Resource Registration

Figure 4.4 shows the interactions that occur during user resource registration.

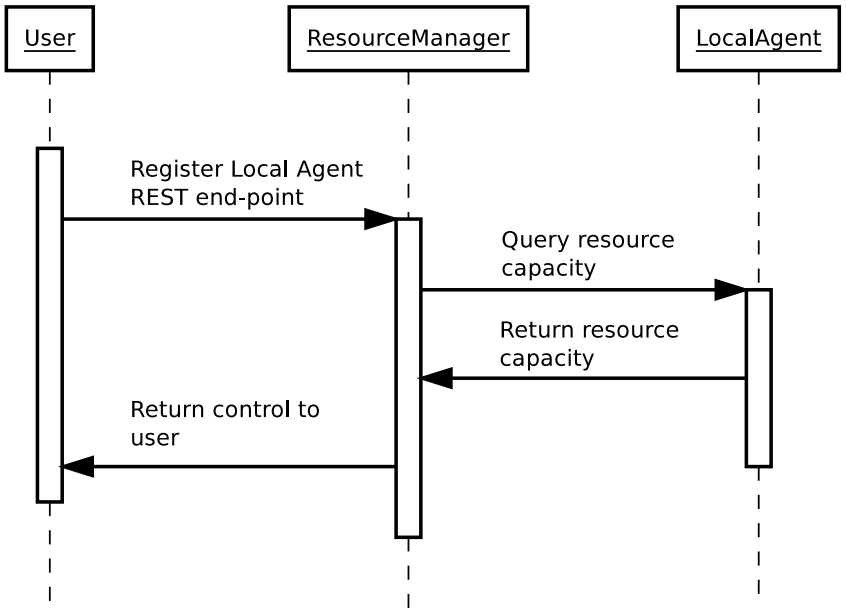


Figure 4.4: User resource registration.

Before the user can register a resource, they first need to implement the Local Agent in their resource. The Local Agent is simply a REST end-point with interfaces to submit computes and query the resource, such as the available capacity and currently running computes. In addition the user would also need to download and cache the standard compute images from the SoCC website.

After performing these steps, the user can register the resource by submitting the REST end-point of the Local Agent to the SoCC Resource Manager. The Resource Manager would do an initial query right away to determine if the Local Agent is running. If all is well control is returned to the user.

4.5.3 Virtual Organisation Creation

In SoCC, there is a one-to-one correlation between a VO and a facebook group. A VO is always linked to a facebook group and cannot exist without one. VO functions are split between facebook and SoCC as follows.

- **Facebook:** User management functions, such as accepting members and assigning roles are handled using the standard facebook groups UI. A user can join a group by navigating to the facebook groups page and clicking the Join Group buttom. The owner of the group can assign admin roles to members, as well as make the group public or closed. Members can post messages to the group, chat with other members, set up events and share documents.

- **SoCC:** SoCC handles VM creation, scheduling and resource management functions. When a users opens the SoCC app, all facebook groups that has been registered as VOs and which this users is a member of is listed. Clicking on one of these will take the user to the VO UI where the user can take various actions in the context of the VO. The particular actions a user can take would depend on their role in the associated facebook group. Admin users can create and delete VMs. Other users can view the resource usage charts and recent activity of the VO. SoCC uses the facebook API to post messages to the facebook group to inform other members about various actions related to the VO, such as when a member contributes more resources.

Thus, there are two UIs for VOs. One is the facebook group UI, which handles all the user management and social functions. The other is the SoCC UI, which is accessible from the facebook SoCC application. This UI handles user resource and VM management functions.

Table 4.1 shows the various mappings of facebook group roles to VO roles in SoCC, and their permissions.

Facebook Group roles	SoCC VO roles
owner Creates the facebook group. Assigns admin roles to other members.	*owner* Creates the VO by registering the facebook group in the SoCC app.
Admin Moderates the facebook group.	*Admin* Manages VM instances (create, delete etc). Represents the actual scientists working on the project.
Member (non-admin) Post messages, creates and shares documents etc.	*Contributor* Users who contribute shares to the VO. This may be ordinary users as well other parties not directly related to the VO, but are interested in contributing to the project.

Table 4.1: Facebook group and VO role mappings

Figure 4.5 shows the interactions that occur during the creation of a VO.

The Account Manager retrieves the groups which the user is a member of in facebook, and from this set filters out any groups not owned by the user and groups that has already been registered. The resulting list of groups is then presented to the user to make a selection. The selected group id is then saved in the local database and appears in the users SoCC menu as a VO.

4.5.4 Compute Requests

Compute requests in SoCC are made in the context of a VO. i.e individual users cannot make a compute request without creating a VO first. This mechanis is required to discourage selfish use of resources and encourage

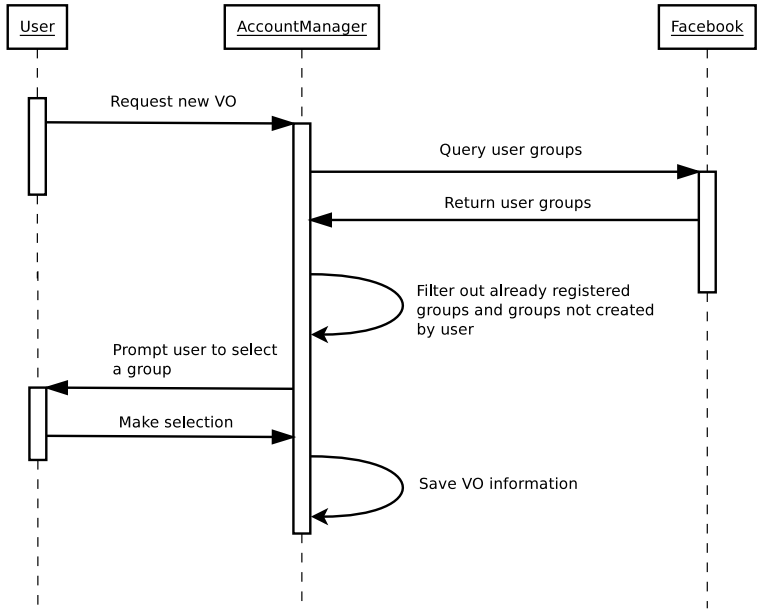


Figure 4.5: Virtual Organisation creation.

collective collaboration.

The normal flow of actions when a user uses a built-in standard image is shown in figure 4.6.

The user starts by first selecting a VO, and then submitting a compute request. The Resource Manager forwards this request to the scheduler which makes a decision to accept or reject the request. If an accept decision is made, the Resource Manager then submits the request to the Local Agent of the resource selected by the scheduler. At this point the Resource Manager returns immediately to the user without waiting for a response form the Local Agent. The Local Agent would then periodically push the compute state changes to the Resource Manager, and the user can monitor this information from the SoCC application. When the compute status becomes `RUNNING`, the SoCC application would also show the IP address of the compute reported by the Local Agent.

Instead of supplying a built-in standard image, users can also use their

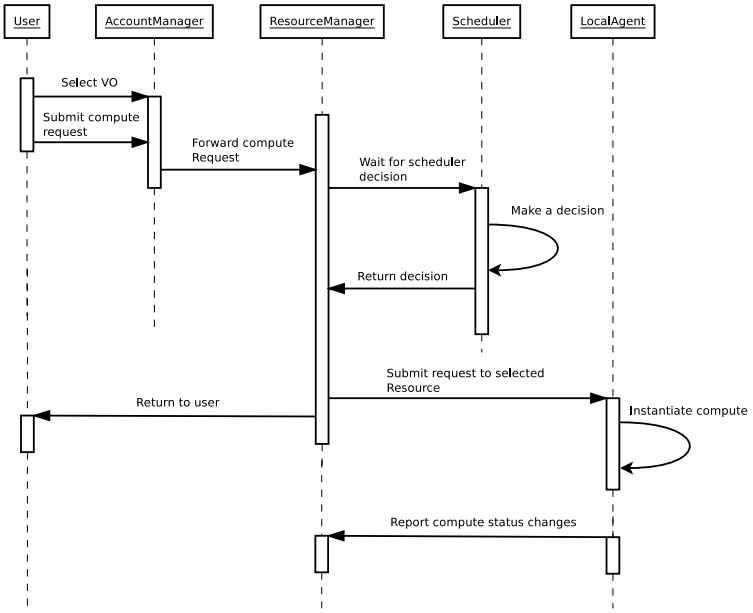


Figure 4.6: Creating a compute instance using a built-in standard image.

own custom images. In this case the user would first need to stage thee custom compute image to a remote location before submitting the compute request. This url of the location would then need to be submitted with the compute request. The interactions would be the same as in figure 4.6, except that the Local Agent would download the custom image before instantiation.

4.6 Virtualisation

4.6.1 Overview

As the name implies, virtualisation refers to a virtual representation of a resource, as opposed to an actual resource. Virtualisation can be applied to different resource types. For example, Network Virtualisation allows running multiple networks sharing a single physical network. Storage Vir-

tualisation allows representing multiple storage devices as a single single storage device. However, in this section, we are interested in look at virtualisation as a method of sharing computing resources.

When applied to computing resources, Virtualisation can be defined as the virtual representation of computing hardware. The virtualised hardware are called Virtual Machines. To the client users, Virtual Machines would appear as any other physical hardware, with CPU, disk and RAM. However, a single physical machine can host many Virtual Machines, which allows the capacity of the physical machine to be shared across the running Virtual Machines.

4.6.2 Advantages of Virtualisation

Virtualisation offers a number of advantages over grids jobs:

1. **Isolation**
   Virtualisation provides isolation greater isolation between users than the level of isolation provided by the Operating System (process isolation). Process isolation allows various means of inter-process communication, such via the file system, inter process communication (eg: dbus) or possibly from buffer overflow exploits allowing one process to read from another process address space. This is fully prevented by virtualisation, thereby providing a greater level of security and protection of the guest OS from other users running on the same system.

2. **Easier checkpointing and migration**
   A running VM can be easily saved directly to a file, moved and restarted in another machine with the state preserved. This comes without the overhead associated with migrating and checkpointing grid jobs, which requires separate handling of job execution state and stored data and also often requires the job to support it. The eas-

ier checkpointing and migration facilities also aids in implementing dynamic load balancing and distribution as well.

3. **Efficient resource usage**
   A single host machine can run many Virtual Machines. This allows a single high specification server to do the job of several low specification servers, but with lower total power consumption and required rack space. Virtualisation also allows the resources consumed by a running Virtual Machine to be scaled to the required amount by the user, which is not possible with dedicated physical machines.

4. **Greater flexibility in platform**
   Choices of guest OS allows running software not supported by the host OS. This eases maintenance and support for cloud vendors (such as Amazon EC2), as the client can instantiate a VM with a suitable OS to run the required software.

5. **Hardware transparency**
   Virtualisation hides the possible heterogeneity of the hardware in a data center. Using virtualisation, a client would only see a standard representation of hardware instead of the actual processor models, speeds and memory types. This allows for deploying different combinations of hardware in a data center, as well as upgrading existing hardware without affecting the clients.

The primary disadvantage of virtualisation is that multiples instances of an OS running on the same machine results in a lot of duplicated state, with each guest OS running a copy of base libraries and software stack. However, the advantages of virtualisation far outweighs this short coming, and with the popularity of the SOC paradigm, virtualisation has now become the prevailing choice for cloud vendors providing Infrastructure and Platform as a Service solutions, such as Amazon EC2, Microsoft Windows Azure and Rackspace.

4.6.3 Approaches to Virtualisation

Full Hardware Visrualisation

This is the virtualisation approach used by most cloud vendors (cite Amazon EC2, Rackspace, GoGrid). With this approach, the underlying hardware is fully virtualised, with a guest Operating System running on top of a hypervisor (figure 4.7). The hypervisor is a piece of software that provides the virtualisaton facilities, such as disk and network IO, memory management and processor sharing. To the guest OS, the hypervisor appears as a physical hardware, but under the hood the hypervisor uses the underlying host Operating Systems APIs to carry out its functions.

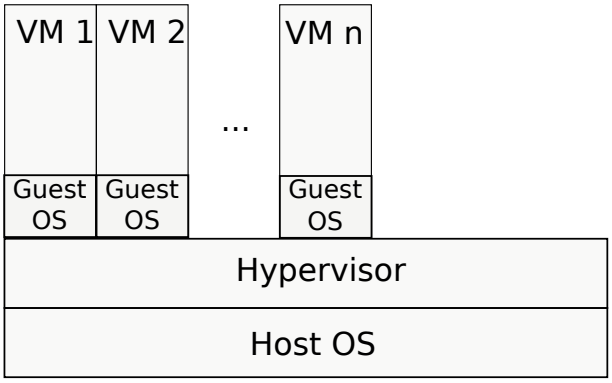


Figure 4.7: Full Hardware Virtualisation. Figure showing VMs running on top of the hypervisor.

There are two main methods for full hardware virtualisation:

1. **Para Virtualisation**
   With this method, the guest OS has a customised kernel to facilitate virtualisation. This does not require hardware support from the host machine, so can be used on old architectures.

2. **Hardware Assisted Virtualisation**
   This form of virtualisation is support by hardware extensions (eg:

Intel VT and AMD Hyper-V). A hypervisor supporting this method of virtualisation provides full support for the required architecture, so that any OS can be install in the hypervisor without any modifications to the guest OS kernel.

Operting System Level Virtualisation

In this approach, multiple VMs are run on top of the same OS, as shown in figure 4.8. The VMs share the basic OS kernel and other basic OS services as the host machine, but uses various isolation techniques, such as container based isolation methods and sandboxing [28, 16] to isolate the VMs. To the user, each VM appears as an stand alone entity, with its own root file system.

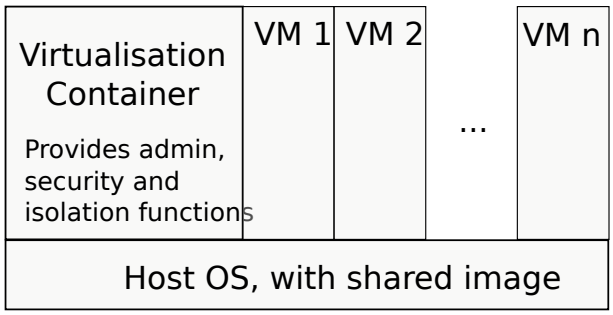


Figure 4.8: Operting System Level Virtualisation. Figure showing the VMs running inside a container.

This provides better efficiency than the above two approaches as basic OS features are not duplicated. However, the trade-off is that the VM OS has to be the same as the host OS and does not provide the same level of isolation among guest VMs [34], so this from of virtualisation does not provides all of the benefits expected of full hardware virtualisation. Despite this, it has found use in the real world (eg: PlanetLab).

4.6.4 Virtualisation Management Middleware

When using virtualisation for anything more than running a single virtual machine, such as when providing a cloud service, it is rare to interact directly with the hypervisor. More common today is to use a virtualisation management middleware to create, delete and manage a group of virtual machine instances. There are several such middlewares available today – Ganeti, Eucalyptus and OpenNebula being a few examples.

Although all these middlewares comes from different vendors, they all have a very similar architecture under the hood. Figure 4.9 shows the logical layout of this common architecture when deployed on a cluster.

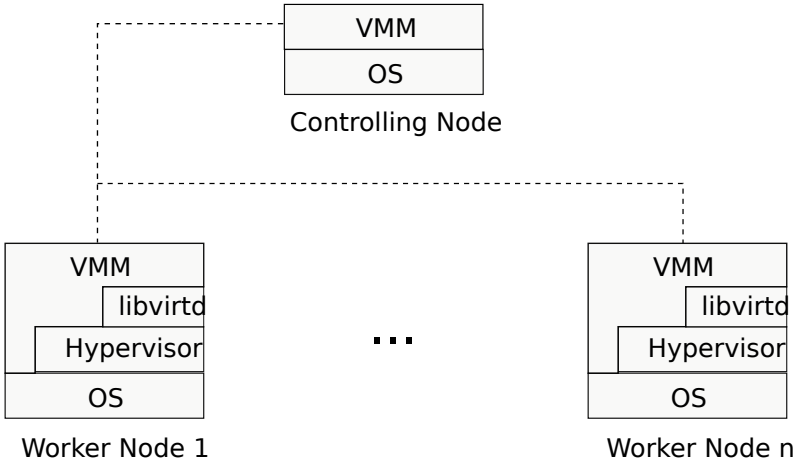


Figure 4.9: Virtualisation Management Middleware Architecture.

The basic deployment of a Virtualisation Management Middleware requires a copy of the middleware to be installed on each node of the cluster. One of the nodes is designated as a controlling node. This node acts as the interface to the outside world and can receive commands from users. The rest of the nodes acts as the worker nodes which hosts the created Virtual Machines. These nodes also have the required hypervisor and libvirt daemons installed. The controlling node takes the function of load balancing the worker nodes, scheduling and monitoring the running Virtual

Machines.

Some virtualisation management middleware, such as Ganeti, provides pure management of Virtual Machines. i.e it provides only the basic functions of creating, deleting, management and monitoring Virtual Machines. Other middlewares, such as Eucalyptus and OpenNebula provides a more complete middleware stack to facilitate deploying private and public clouds, such as user management and built in support for standard protocols such as OCCI (Open Cloud Computing Interface [10]). With these middlewares, the technical barriers and know how that is required to deploy a cloud service has now become very small.

4.7 SoCC FairShare Scheduler

Fair share scheduling allocates resources to users based on how much of the resource a user is entitled to. Originally described by Kay and Lauder [11] in 1988 for allocating machine shares for students, variants of this scheduler has also been applied in grid environments for job queueing [27, 25, 37]. This section presents an implementation of a fair share scheduling mechanism for SoCC.

4.7.1 Fair share scheduling concept

The concept behind faire share scheduler is that each user is entitled to a share of a scarce resource, and the scheduler would need to ensure that each user gets, in the long run, a portion of the resource proportional to their share. The original paper describes a fair share scheduler used in a university computing department for allocating CPU process properties. In that implementation, each user is given a numerical figure representing that users share. For example, a first year student may be given 100 shares while a second year student may get 200 shares, reflecting the amount of work each student is expected to do. The scheduler would then use these

figures to calculate the priorities of students CPU processes, resulting in a CPU usage picture corresponding to the students shares in the long run. Although this scheduler was used to schedule CPU processes, it can be adapted to any constrained resource shared among user for non-monetary purposes.

4.7.2 Using the fair share scheduler in SoCC

The incentive for users in contributing resources to SoCC is that they get a greater return than what is contributed. Therefore the goal of the scheduler in SoCC is that each VO is entitled to a minimum share of resources equal to that contributed by its users (its fair share), plus more depending on current utilisation. So a VO compute request should always be granted if resources are available and if it is currently not using its entitled minimum share of resources. If it is already using its minimum entitlement, then the scheduler must make best attempt to fulfill the request while favouring those who contributes the most.

The fair share implementation in SoCC prototype solves this problem by setting a threshold on the utilisation below which all compute requests are granted. Figure 4.10 shows a graph with the threshold marked at 70%. Below this value, all compute requests are accepted to allow users to make make maximum use of the available resources, even those who are contributing very little. This mechanism allows the the resources to be utilised even when heavy contributors are not using them.

When the utilisation threshold is reached, the fair share algorithm kicks in to favour heavy contributors. The following formula is used to calculate the minimum share a VO must posses in order for a request to be accepted in this situation:

This is the formula of the straight line graph in figure 4.10. It represents the minimum shares a VO must possess above the utilisation threshold. If the shares of the requesting VO is less than this value, then the request is

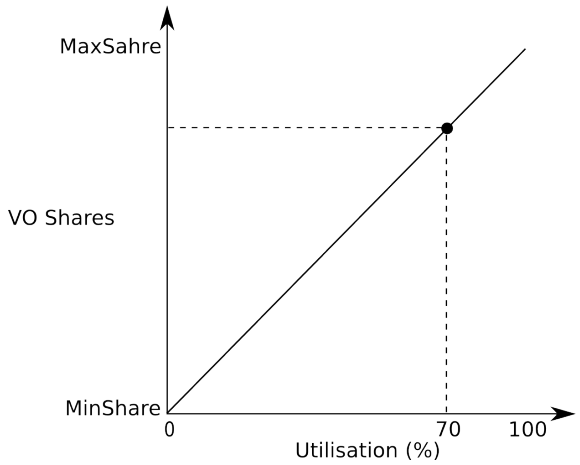


Figure 4.10: Graph showing the utilisation threshold above which the fair share algorithm would apply.

$$Share = \left(\frac{MaxShare - MinShare}{100} * Utilisation \right) + MinShare$$

Figure 4.11: Equation to calculate the minimum shares required to submit a request.

rejected. Some implementations have opted to queue the requests instead of an outright rejection [22], but this is not implemented in the prototype to prevent users from making advanced reservations and dominating the system.

The shares of a VO is a quantitative measure of entitlement of the VO to the resources available in SoCC. In the implementation its value is simply the sum of the resource units contributed by its constituent users, since this represents a VOs contribution. The variable MinShare represents the share of the VO with the lowest contribution, and MaxShare represents the share of the VO with the highest contribution.

This scheduler ensures VOs with few members, or with members that can not make a large contribution will have to make use of the cloud dur-

ing low utilisation periods. It does this while favouring VOs who contribute more during high utilisation periods, providing an incentive for contribution. Alternative approaches for scheduling, such as using virtual currencies, does not ensure this behaviour.

A performance evaluation of this scheduler on a real world trace and a synthetic balanced trace is presented in the next chapter.

Chapter 5

Testing And Evaluation

This chapter describes the tests done to evaluate the performance of the Social Collaborative Cloud. Two groups of tests were done. The first group comprises of real world tests to subjectively evaluate the suitability of SoCC in practical usage. The second group comprises of simulations done to evaluate the effectiveness of the scheduler.

5.1 Real World Tests

This section is intended to demonstrate the working of SoCC by testing real world use cases. It is not intended to demonstrate its readiness for production use, but rather the workings of the architecture of this initial prototype.

Figure 5.1 shows a screenshot of the tested prototype. The tests were carried out in a 1Gb LAN environment with one Intel Core2Duo 3.0GHz CPU, 4GB RAM machine hosting the SoCC application server, and three Intel Core i5 3.0GHz CPU, 4GB RAM machines acting as a user resource. Out of these three machines, one was dedicated to host the Local Agent.

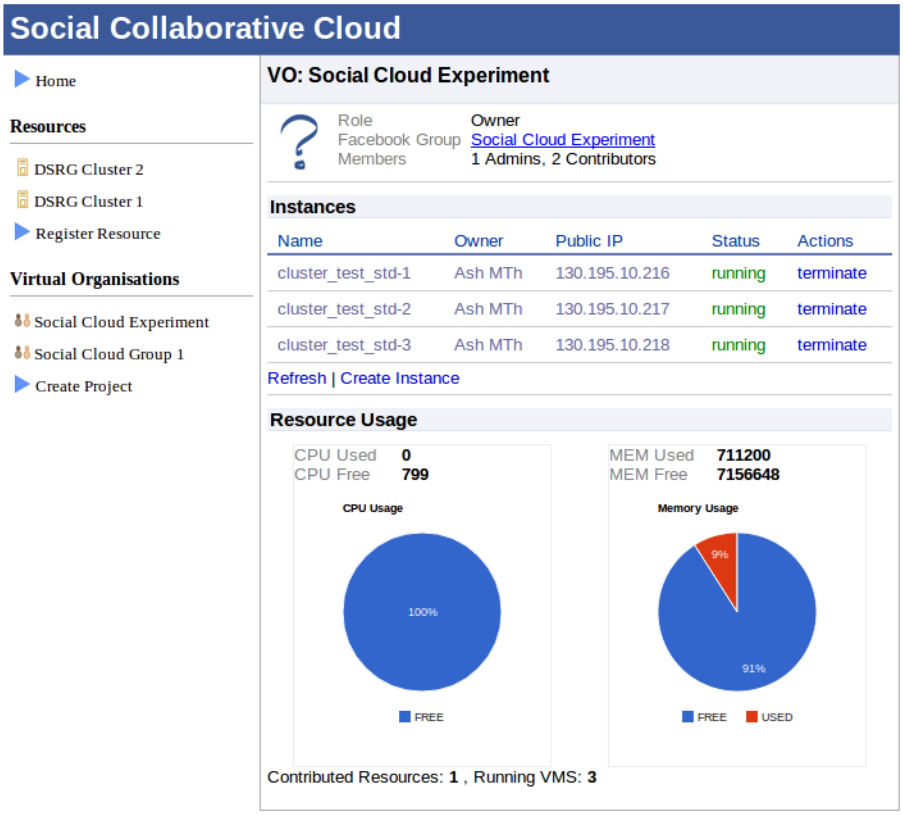


Figure 5.1: SoCC facebook application.

5.1.1 Performance of Compute Instantiations

These tests are intended to evaluate the performance of the two image distribution methods. As outlined in section 4.5.4, there are two image distribution methods.

- Method 1: The user selects a standard cached image from the VM creation screen. With this option, the image would already have been cached in the target resources, so the image transfer time to the target resource would be almost nil.

- Method 2: With this option the users prepare their own custom im-

age and upload it to a staging area. When the user creates the compute, the URL of this image is given to SoCC. When the compute is scheduled to a resource, the Local Agent located on the resource first needs to downloads the image the local storage before it can instantiate the compute. This option gives the user more flexibility while sacrificing compute instantiation time.

Figure 5.2 shows a screenshot of the compute request screen. With both methods, two types of images were test. The first image is of a lightweight, 40MB linux distribution called ttylinux. This is suitable when a user wishes to quickly deploy a compute, or to use as a starting point to create a custom image from scratch. It comes with only the basic linux packages, notably a C compiler and bash interpreter.

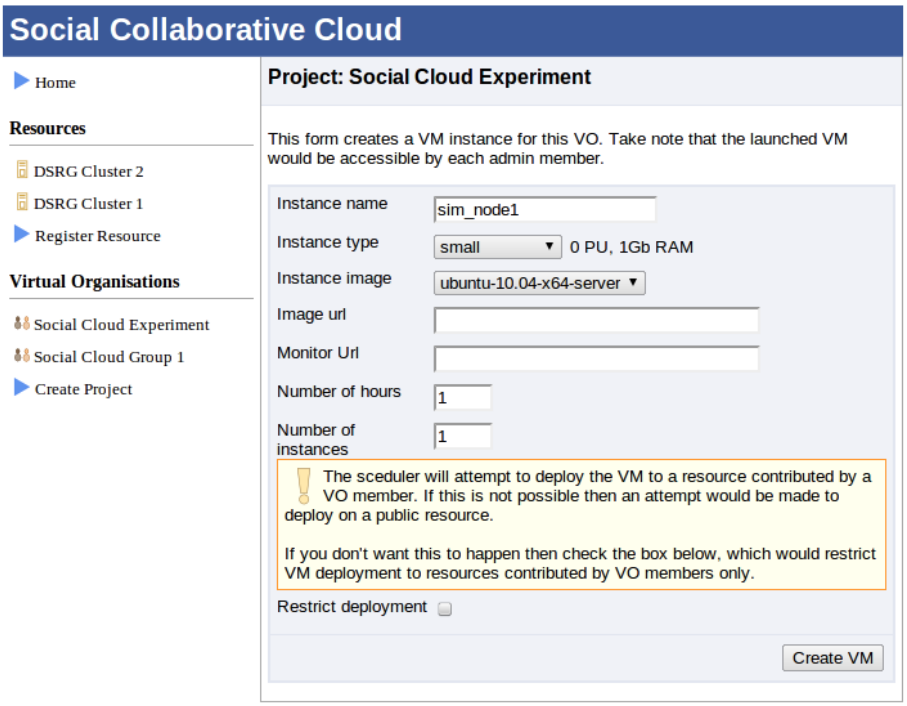


Figure 5.2: SoCC compute request screen.

The second image is a Ubuntu 10.04 server linux image. This image is 5GB in size, and comes with all of the common packages expected of a general purpose linux distribution, including the Apache web server, mysql database and Network File System support. This image is suitable for a compute to start using with minimal configuration for many common scenarios.

The following figure 5.3 represents the life cycle of the compute creation process.

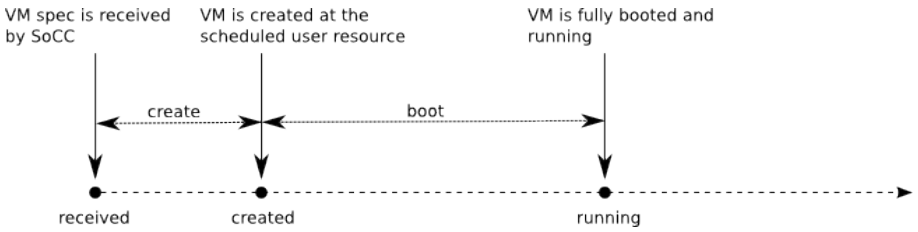


Figure 5.3: Compute life cycle state.

The states are as follows:

- **received.** The image is received by SoCC.
- **created.** The image has been created on the scheduled resource and is booting. For the image distribution methods 2, the image has to bee transferred before this can occur, thereby delaying the booting time.
- **running.** This state is reported when the image has a IP address assigned and is running.

Table 5.1 and figure 5.4 represents the results of this experiment (this particular result set was also published in the related publications in section 1.3):

As seen from the graphs, the total boot up time is dependent on the image size. Especially if the image is a custom image, the image transfer

	create (sec)	boot (sec)
ttylinux option 1	10	15
ubuntu 10.04 option 1	820	126
ttylinux option 2	2	10
ubuntu 10.04 option 2	2	125

Table 5.1: Compute instantiation timings with the different image distribution methods.

time dominates the total startup time, suggesting that its important to offer a collection of images suitable for many common scenarios.

5.1.2 Evaluation of Extension Mechanisms

The extension interface is intended for advanced users who would like to integrate SoCC with their own system, or would like more customisability. The following tests demonstrate the working and performance of this interface.

This test creates a virtual cluster in the SoCC cloud. To achieve this, the user uploads a customised image that has been seeded with contextualisation scripts that would configure the computes network settings with information about the other nodes of the cluster. This is made possible by the SoCC Event Publisher and a REST service setup by the user. The REST service is setup so that it can receive the event notifications published by the Event Publisher. Figure 5.5 shows this arrangement.

Interactions occuring in figure 5.5 are as follows:

- **Step 1.** User submits request for 3 computes, with custom images.
- **Step 2.** The Resource Manager forwards the request to the scheduled resource.

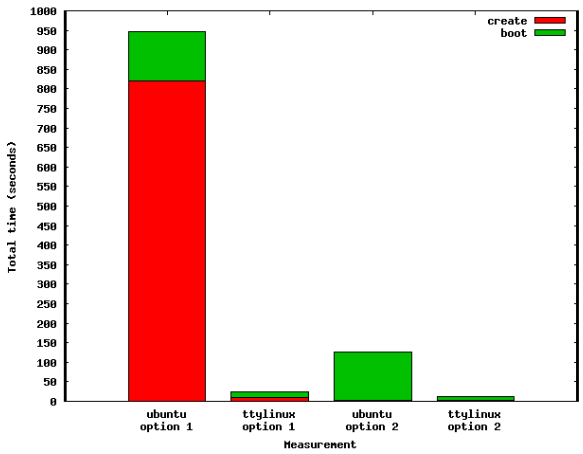


Figure 5.4: Compute instantiation with the different image distribution methods.

- **Step 3.** The Local Agent of the resource downloads the users images from the remote location and instantiates the computes. The Local Agent pushes the status updates of the computes to the SoCC Resource Manager.

- **Step 4.** The Resource Manager notifies the users registered REST service with the compute status updates.

- **Step 5.** When the REST service have received the RUNNING state for all the 3 computes, it ssh to each compute with the information about the others computes. The cluster contextualisation scripts on the computes then configures the network interfaces of the compute.

The REST was setup on a local dekstop computer. The only image used for this test was the ubuntu image. Figure 5.6 shows the results of this experiment.

For comparison, figure 5.7 shows the same test done using a cached image with the same contextualisation scripts. As both graphs shows, the overhead taken to take advantage of the extension mechanism to perform

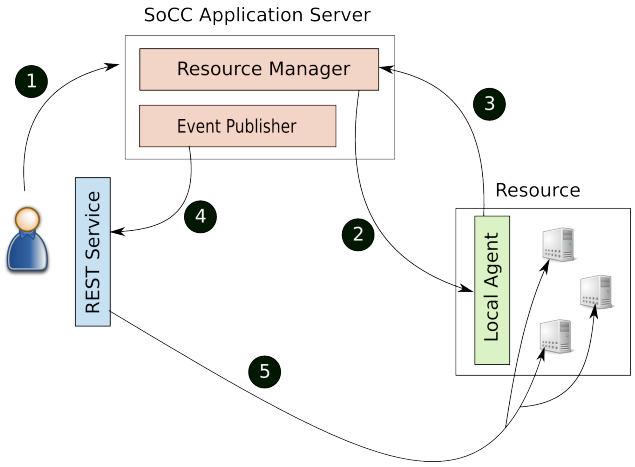


Figure 5.5: Using the extension mechanism to create a virtual cluster.

a contextualisation is very small compared to whole process.

5.2 Simulation Tests

The simulation tests evaluates the effectiveness of the fair share scheduler implementation in SoCC. The tests in this section compare the fair share scheduler against a random scheduler and two virtual currency based schedulers. These tests require a large number of users and several compute submissions which, due to the time scale set for this thesis, can not be conducted in a real world setting. Hence a simulation approach was adopted.

5.2.1 Design of the Simulator

The simulator is a custom Java programme written to simulate the various components of the SoCC architecture (figure 5.8). Instead of using an existing simulation package, a custom simulator was opted for in order to model the unique characteristics of a social network and also because

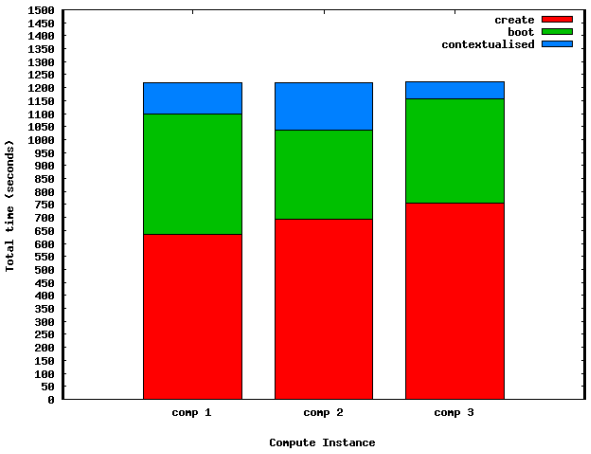


Figure 5.6: Virtual cluster creation using a custom image.

it was intended to improve and extend the simulator as a tool for use in future work.

The Scheduler component implements the scheduling algorithm and is swappable with different implementations. The simulator accepts three sets of input data (figure 5.9):

1. The description of the users and their associated Virtual Organisations,

2. Distribution of resources among the users and

3. A trace of the compute submissions

Each set of data is read by the simulator from a text file formatted in a simple description language. The users description input file consists of the users and their associated Virtual Organisation in the cloud:

```
#
# Users and VOs input file.
#
[USERS]
```

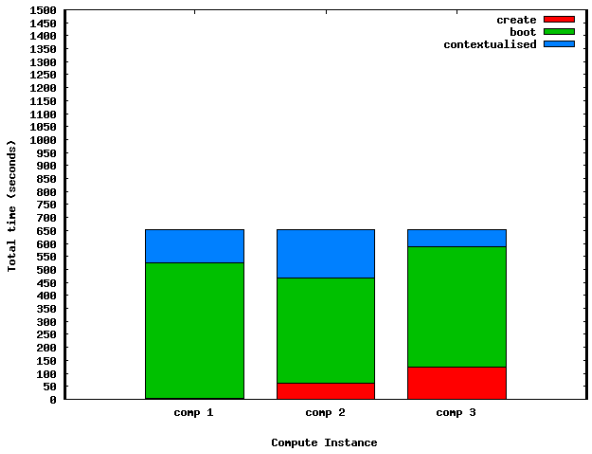


Figure 5.7: Virtual cluster creation using a cached image.

```
# ID NAME CREDITS
1 @user1 100
2 @user2 100
...
[VOS]
# VO_ID USER_ID
1 1
1 2
1 10
2 4
...
```

The resource distribution input file consists of how the resources are distributed among the users:

```
#
# User resource distribution input file
#
# USER_ID RESOURCE_UNITS
1 4
```

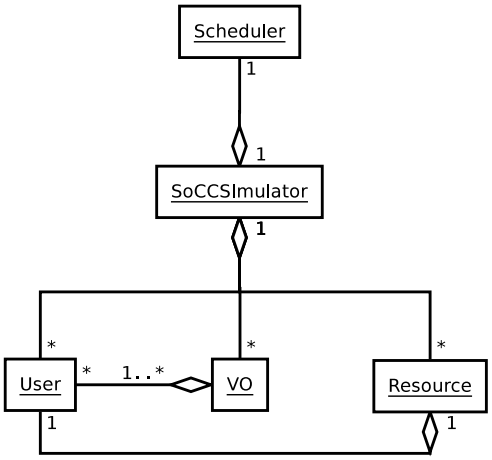


Figure 5.8: Simulator UML diagram.

```
2 4
...
```

The compute trace input file consists of a trace of the compute submissions to SoCC:

```
#
# Compute trace input file.
#
# ID USER_ID VO_ID SUBMIT_T RUN_T TYPE HOURS
1 1 1 0 39 small 72
2 2 2 11 1 small 72
3 3 3 19 56 small 72
4 3 3 20 55 small 72
...
```

Each record in this file is tagged with the Virtual Organisation that requested the compute, along withe the time, duration and the type of the compute.
The simulator submits compute requests from this trace to the scheduler,

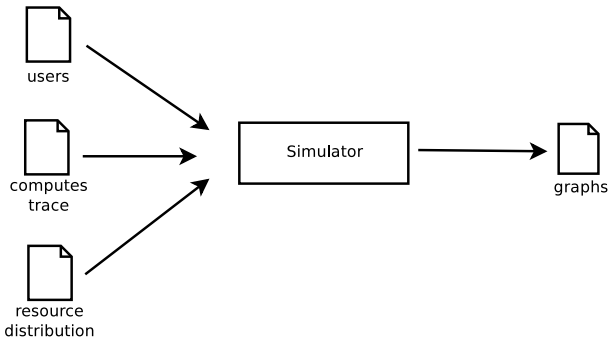


Figure 5.9: Inputs and outputs of the simulator.

and records various parameters throughout the simulation run. At the end of the simulation the recorded data is written to the output files.

5.2.2 Test Dataset

The test dataset chosen for this evaluation was the AuverGrid trace selected from the Grid Workloads Archive (GWA) [8]. The GWA hosts a number of contributed grid traces from various public grids. The AuverGrid is used primarily for high-energy physics research and consists of dual 3GHz Pentium IV Xeon nodes distributed over 5 clusters. In total the grid has a capacity of 475 CPUs, and the dataset contains the usage trace for one year. As the datasets in GWA are in Grid Workloads Format (GWF), for the following tests the trace had to be first converted to the compute trace formats described in the previous section. In addition, the dataset had to be *sanitised* by doing the following:

- All invalid jobs were removed (jobs that were cancelled before starting resulting in zero run time),
- Since the dataset is a trace from a grid environment, the jobs are described in terms of the required CPUs and memory. These were replaced with an equivalent compute type (eg: small, medium).

- Each job in the dataset contained a user and group identifier, representing the user that submitted the job and the group that user belonged to. From this information the number of unique users and groups was counted. The groups were modelled as VOs, and the users in the jobs that contained a particular group (VO) identifier was treated as that VOs members. The users and VOs were then copied to separate input files, and each users was given 100 initial credits for the virtual currency based simulations.

- Resources were assigned to users using a Poisson distribution with a mean value of 10 units.

After preparation, the AuverGrid trace had the following characteristics:

- Number of users: 401

- Number of VOs: 8 (VO1: 158 users, VO2: 107 users, VO3: 16 users, VO4: 47 users, VO5: 46 users, VO6: 11 users, VO7: 10 users, VO8: 8 users)

- Number of computes: 339314 with an average duration of 71 hours.

Figure 5.10 shows the number of users in each VO, and figure 5.11 shows the total resource units contributed by the users of each VO, which is the sum of the resource contributions of its constituent users. Figure 5.12 shows the resources used by each VO if there were no rejections (i.e unlimited capacity). This figure shows how much resources each VO consumes relative to each another, and is shown for the first hundred days for clarity (the resource usage for the rest of the days shows a similar pattern).

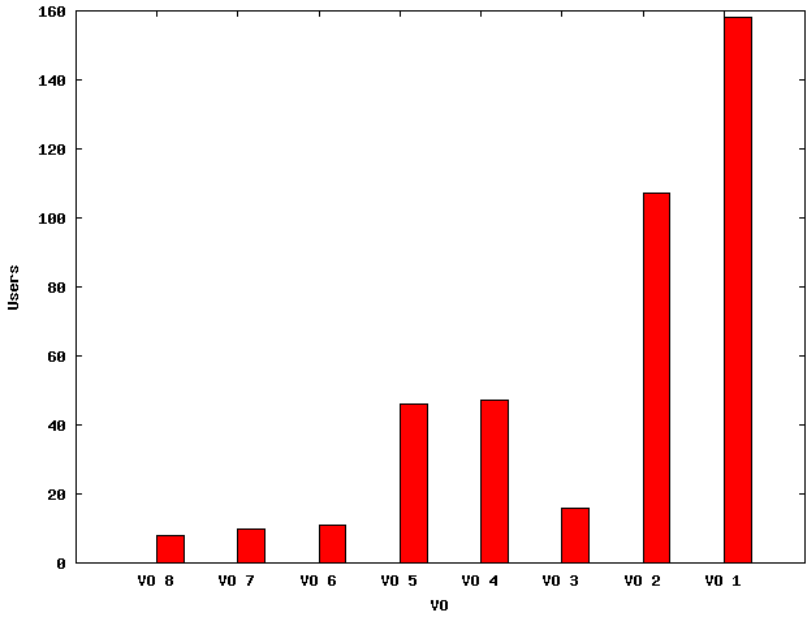


Figure 5.10: Number of users in each VO.

One observation from figure 5.12 is that the VOs are submitting compute requests roughly proportional to the resources they are contributing. For testing the effectiveness of the fair share scheduler, a more balanced trace was also generated where each VO submits the same amount of compute requests. This was generated by allocating an average of 300 computes each with an average of 71 hour duration (derived using a poisson process with a mean value of 300 and 71) to each VO every day. This results in resource consumtion of almost 4 times more than that is available from the cloud. This would allow to test the effectiveness of the scheduler in high utilisation scenarios. The new resource consumption with this balanced trace is shown in figure 5.13. The simulations that follows tests the various schedulers with both of these compute traces.

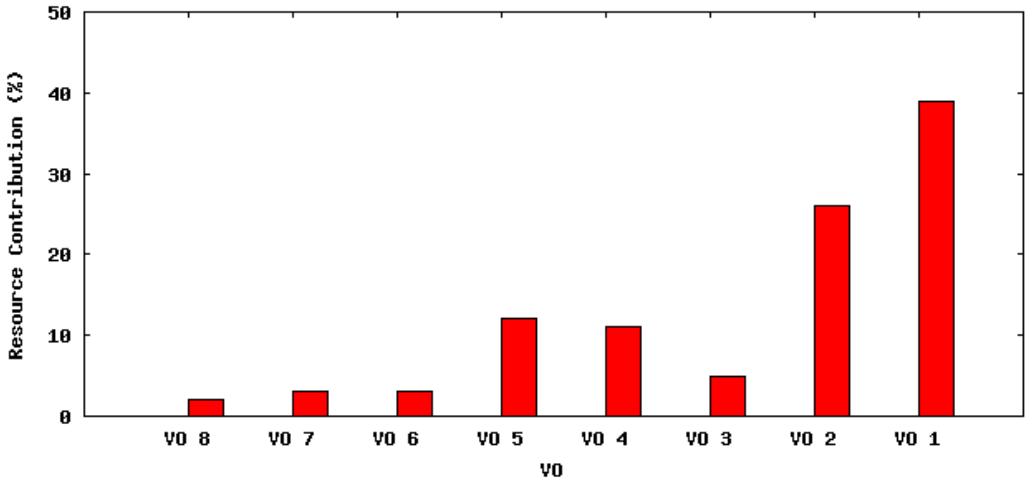


Figure 5.11: Resource contributions by each VO.

5.2.3 Algorithms Tested

The following scheduling algorithms was simulated against the fair share scheduler:

1. **Random scheduler.** This algorithm simply selects a random resource from the list of resources with available capacity when a compute request is submitted. It does not take into account which VO is submitting the request and operates on a fist come fist serve basis.

2. **Global pricing.** This algorithm uses the virtual currency to calculate a cost per unit resource per hour based on global supply and demand. Thus, if the utilisation is zero, the cost will also be zero. As the utilisation rises, the cost per unit resource also rises. Figure 5.14 shows how the cost is calculated.

   When a compute request is accepted, the scheduler transfers the credits from the users of the VO to the user hosting the resource. The hosting user is selected using a round robin algorithm, which

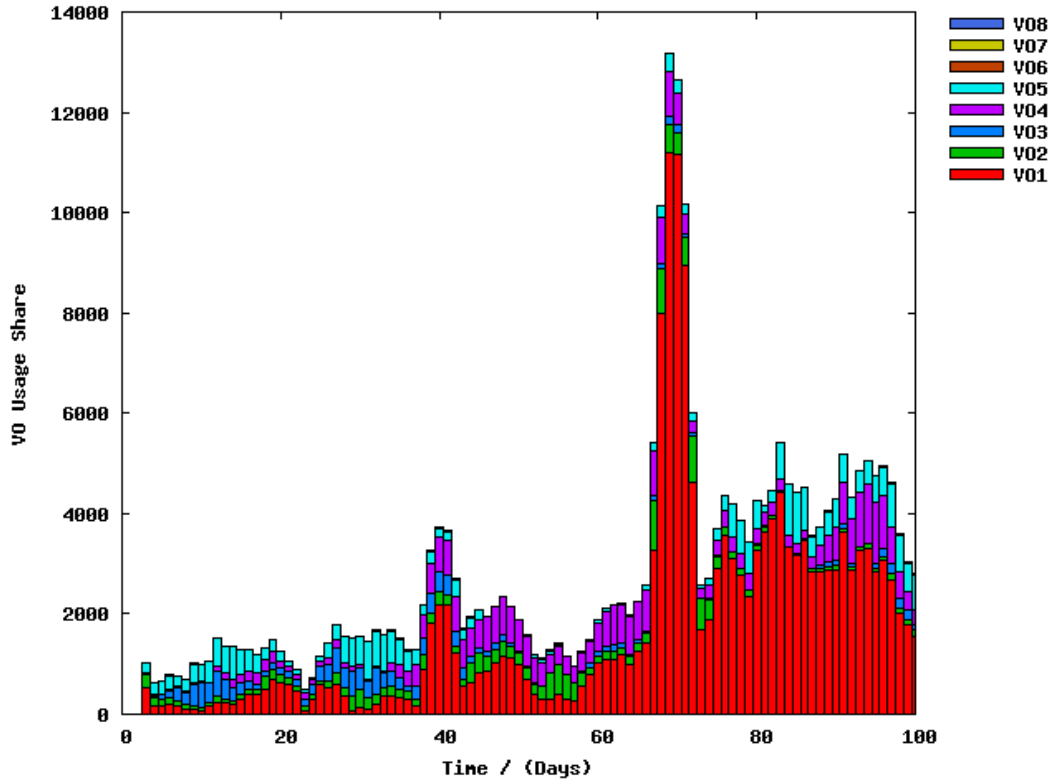


Figure 5.12: Daily resource consumed by each VO for the original Auver-Grid trace.

always selects the user who have not served a request for the longest duration and who have free capacity.

3. **Local pricing** This algorithm uses a similar unit resource per hour cost calculation algorithm, except that the pricing agent is on the local resource. The local resources calculate the cost based on its own load, which results in different resources offering different prices. The scheduler submits the request to the resource the resource that offers the lowest price and transfers the credit.

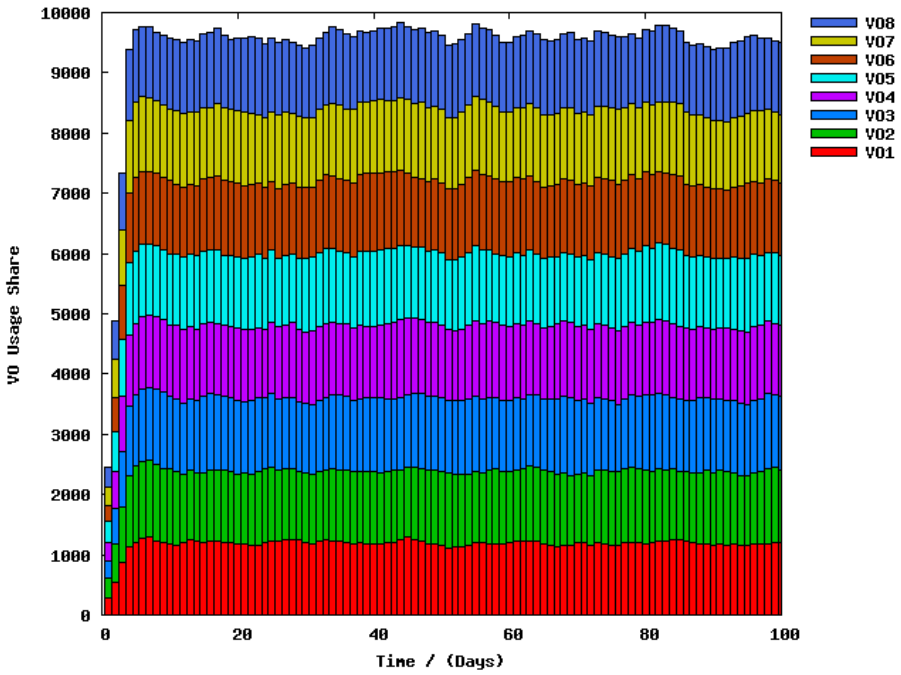


Figure 5.13: Daily resource consumed by each VO for the synthetic trace.

5.2.4 Simulations

This section describes results of running the simulator with the test dataset. Each scheduling algorithm was simulated using several resource distributions among the users. With each run, the following information was recorded and plotted at the end of the run:

- Daily max and min resource utilisations,
- Compute rejection rates, both due to lack of capacity and lack of credits,
- Spot price variations (for the Global Pricing Model)

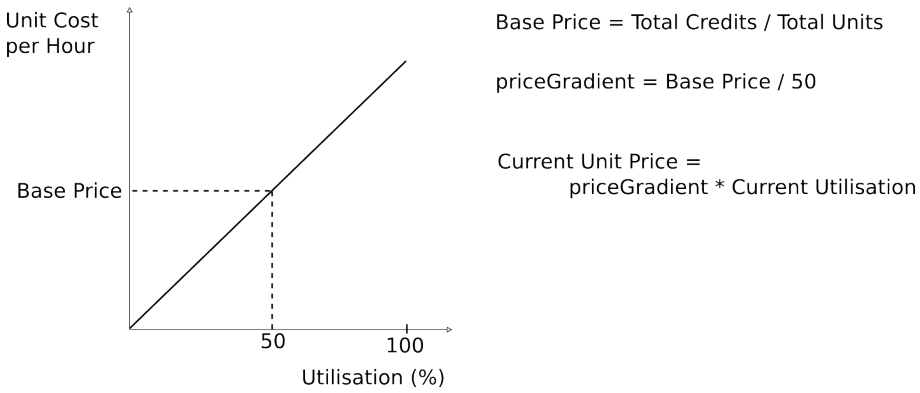
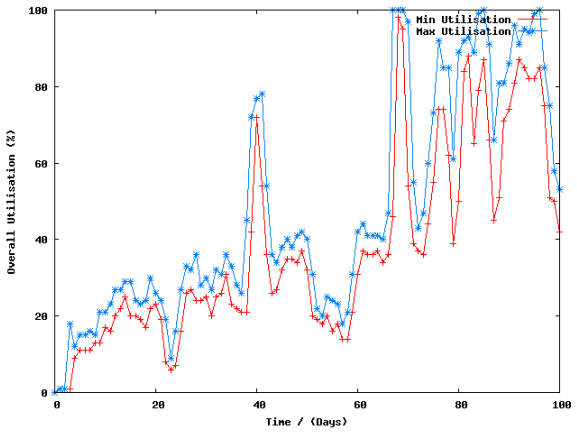


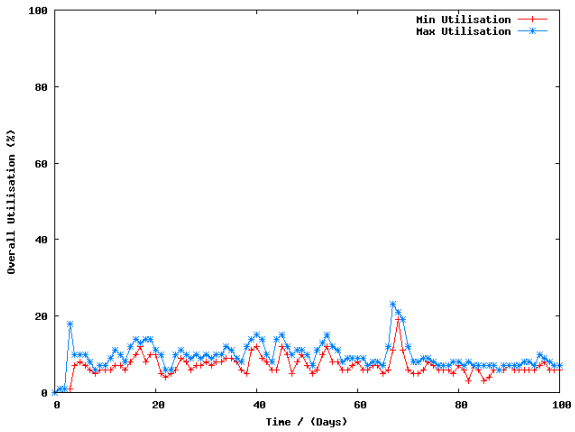
Figure 5.14: Global pricing algorithm.

5.2.5 Simulations Using the AuverGrid Trace

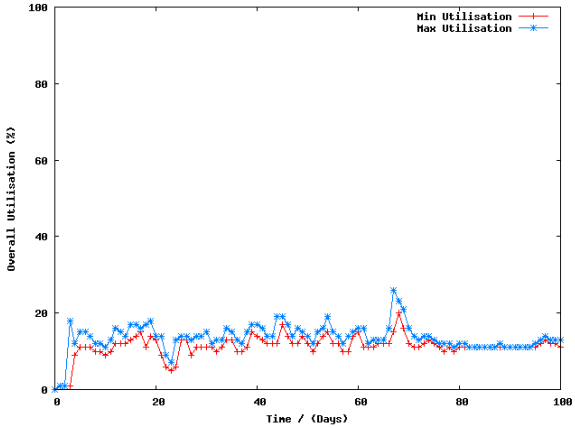
Utilisation



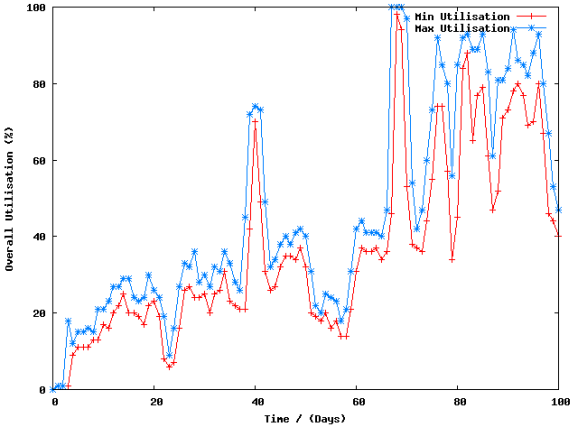
(a) Resource utilisation when using random scheduler with AuverGrid trace.



(b) Resource utilisation when using global pricing with AuverGrid trace.

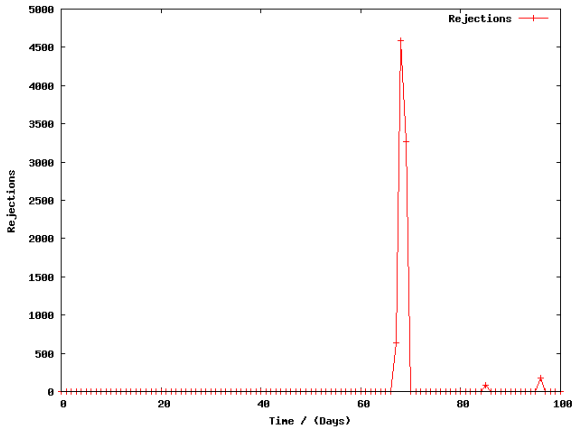


(c) Resource utilisation when using local pricing with AuverGrid trace.

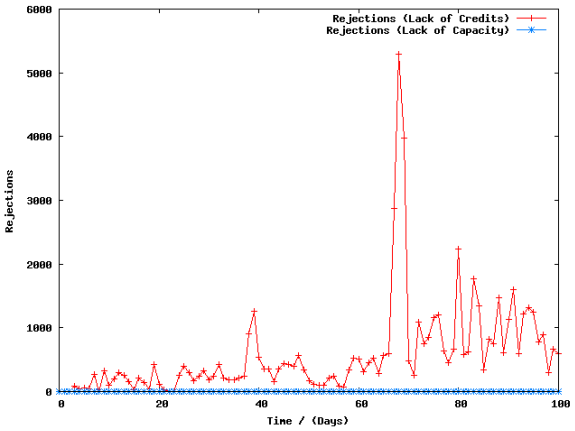


(d) Resource utilisation when using fair share scheduler with AuverGrid trace.

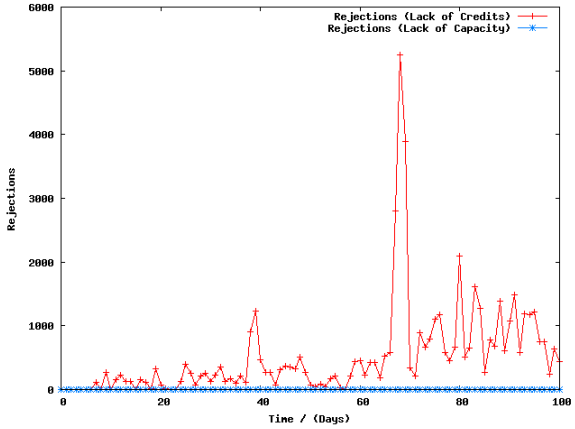
Compute request rejections



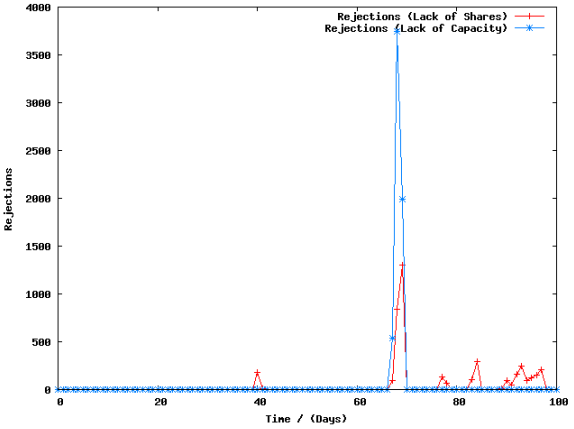
(e) Compute request rejections when using random scheduler with AuverGrid trace.



(f) Compute request rejections when using global pricing with AuverGrid trace.

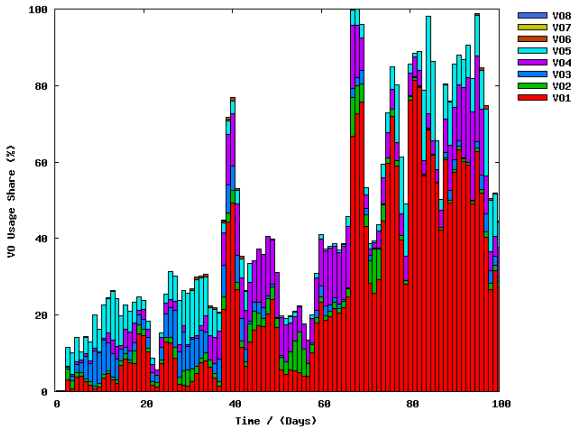


(g) Compute request rejections when using local pricing with AuverGrid trace.

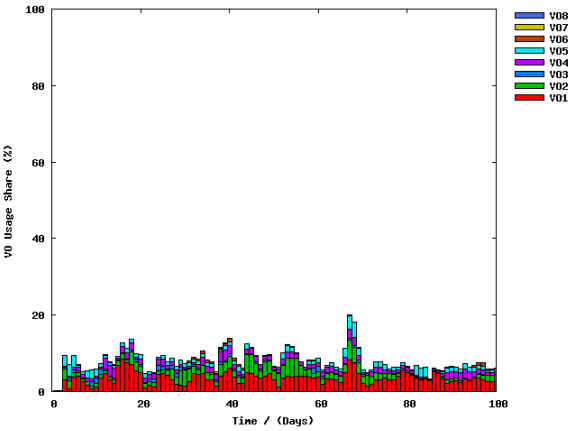


(h) Compute request rejections when using fair share scheduler with AuverGrid trace.

VO resource consumption



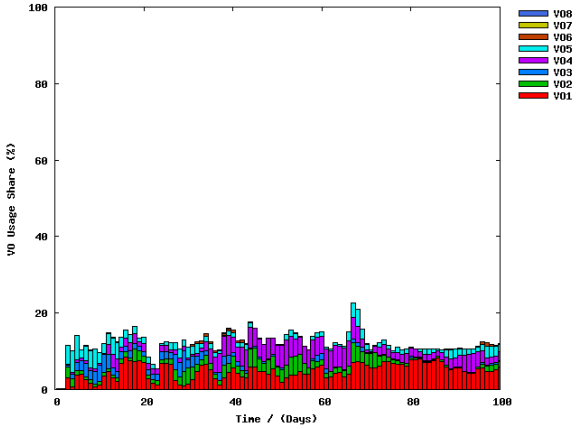
(i) Daily VO resource consumption when using random scheduler with AuverGrid trace.



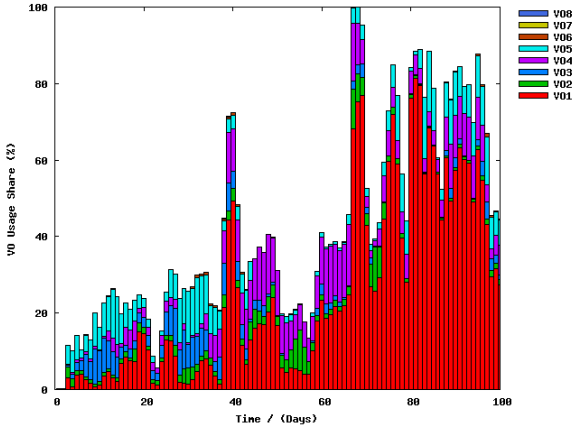
(j) Daily VO resource consumption when using global pricing with AuverGrid trace.

Discussion

Table 5.3 shows the utilisation of the schedulers in numbers.



(k) Daily VO resource consumption when using local pricing with AuverGrid trace.



(l) Daily VO resource consumption when using fair share scheduler with AuverGrid trace.

First thing that is noticeable from this set of results is the poor performance of the global and local pricing schedulers using virtual currencies. This is not surprising when how these two algorithms operate is considered. Each user is given a fixed amount of credit (100 in this case), and the VO uses the credits of its constituent users to buy computes. Eventually the VO exhausts its bag of credits, and if other VOs aren't requesting enough computes themselves, then the VO can not generate credits to buy anymore computes. Figure 5.12 shows that within the first 100 days only VO 1 submits the bulk amount of computes, so after a point VO 1 simply runs out of credits.

The performance of random and fair share scheduler is very close. The fact that a random scheduler is performing well might appear a bit surprising at first. The random scheduler just submits any request it gets as long as there is free capacity. Thus, as long as the total requests does not exceed the available capacity, those requests would be fulfilled. And this is what is happening here precisely. The rejections graph shows that close to the 70 day mark a sudden burst of rejections occurs corresponding to a sudden increase in requests during this time.

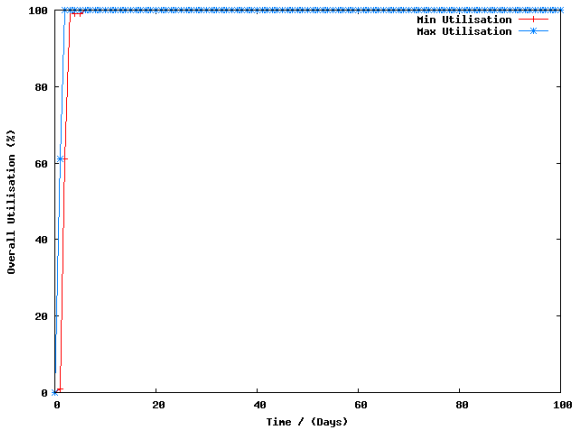
	Random Scheduler	Global Pricing	Local Pricing	Fairshare Scheduler
VO1	50256	88711	88711	88711
VO2	51432	60763	59147	88906
VO3	51590	9408	12249	18992
VO4	50256	26681	25016	45212
VO5	43668	26312	28204	47687
VO6	54740	6321	5527	10766
VO7	48965	5953	8273	12706
VO8	53134	4204	4449	8035
Total	404041	228353	231576	321015

Table 5.2: Utilisation figures for the AuverGrid trace in resource units consumed.

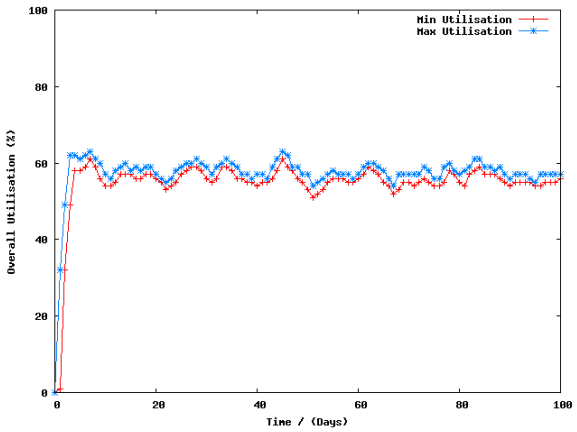
The performance of the fair share scheduler is not surprising. This scheduler grants every request as long as the total utilisation stays below a certain level (set to 70% in this case). As soon as the utilisation goes above this threshold the fair share algorithm comes into play and requests are granted to those making a greater contribution. In this case VO 1 makes the greatest contribution and also submits the most request during the period shown, so its requests are granted.

5.2.6 Simulations Using the Balanced Trace

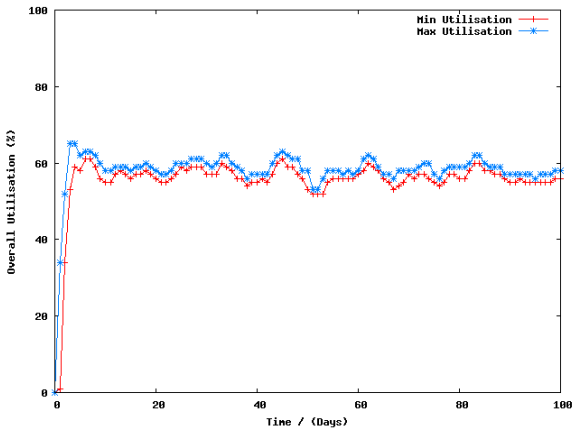
Utilisation



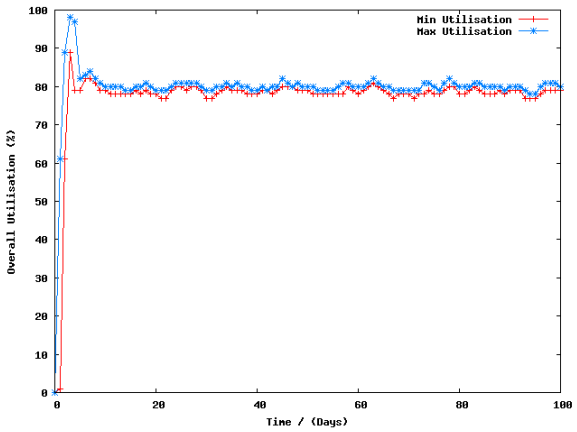
(m) Resource utilisation when using random scheduler with balanced trace.



(n) Resource utilisation when using global pricing with balanced trace.

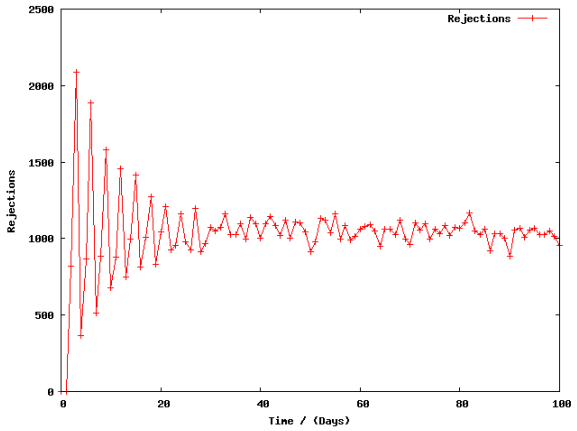


(o) Resource utilisation when using local pricing with balanced trace.

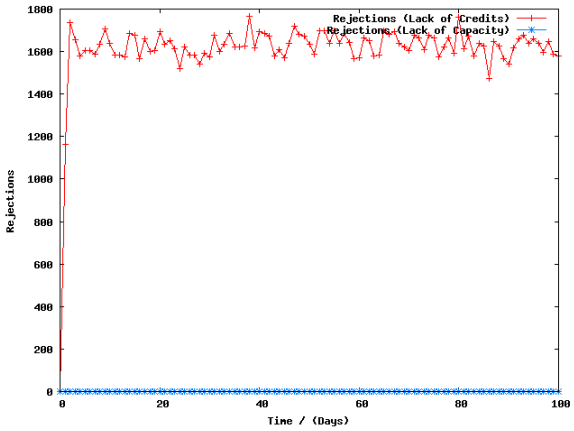


(p) Resource utilisation when using fair share scheduler with balanced trace.

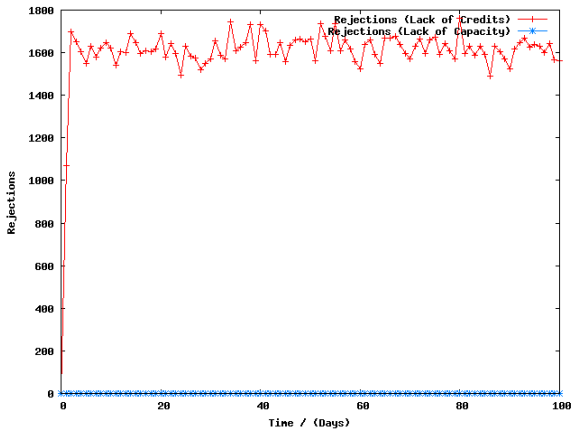
Compute request rejections



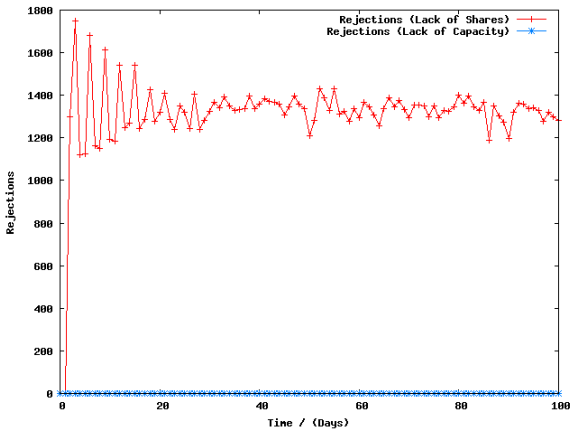
(q) Compute request rejections when using random scheduler with balanced trace.



(r) Compute request rejections when using global pricing with balanced trace.

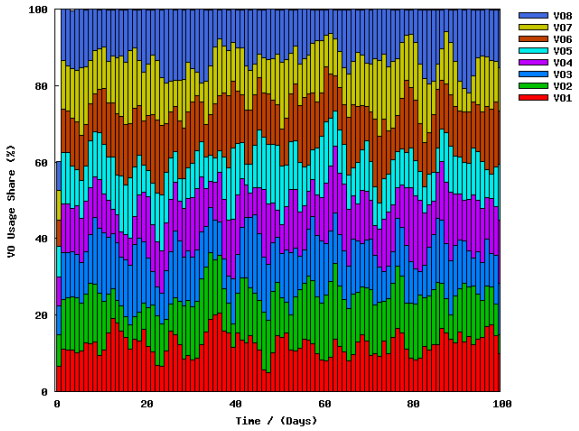


(s) Compute request rejections when using local pricing with balanced trace.

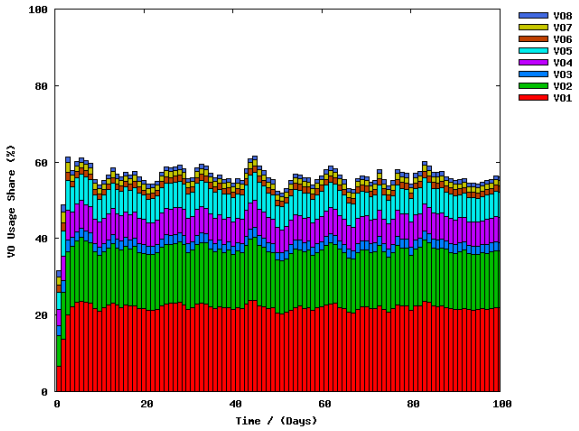


(t) Compute request rejections when using fair share scheduler with balanced trace.

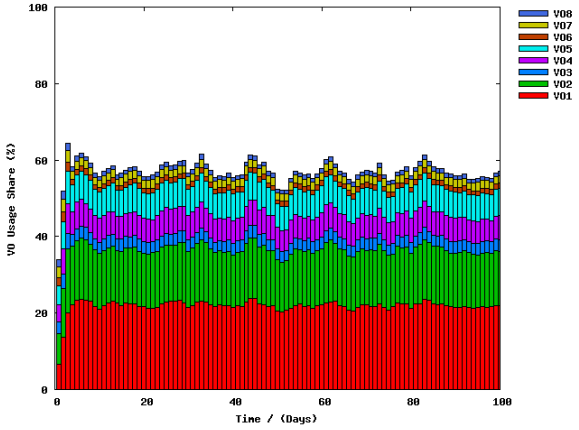
VO resource consumption



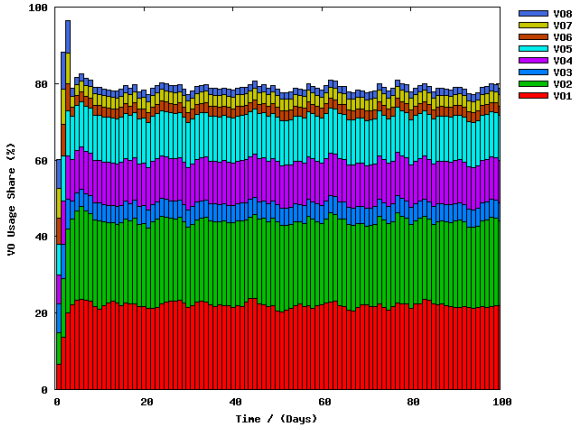
(u) Daily VO resource consumption when using random scheduler with balanced trace.



(v) Daily VO resource consumption when using global pricing with balanced trace.



(w) Daily VO resource consumption when using local pricing with balanced trace.



(x) Daily VO resource consumption when using fair share scheduler with balanced trace.

Discussion

Table 5.3 shows the utilisation of the schedulers in numbers.

	Random Scheduler	Global Pricing	Local Pricing	Fairshare Scheduler
VO1	50256	88711	88711	88711
VO2	51432	60763	59147	88906
VO3	51590	9408	12249	18992
VO4	50256	26681	25016	45212
VO5	43668	26312	28204	47687
VO6	54740	6321	5527	10766
VO7	48965	5953	8273	12706
VO8	53134	4204	4449	8035
Total	404041	228353	231576	321015

Table 5.3: Utilisation figures for the balanced trace in resource units consumed

Once again the performance of global and local pricing virtual currency schedulers is not surprising. The utilisation is higher in this balanced trace than the AuverGrid trace simply because all VOs are submitting requests fairly equally, so each VO is also generating credits as much as it is spending. However, since VO 1 has the most users, and hence have the most credits, it can buy most of the computes it is requesting. The other VOs are more unfortunate as their fewer users results in not being able to generate as much credit as the VO requires.

The random scheduler achieves 100% utilisation, and then starts rejecting. It accepts requests as long as their is free capacity, and does not distinguish in any way between who is making the request. The result is that very light contributors like VOs 6, 7 and 8 gets an equal share as heavy contributors like VO 1.

Most interesting from these results is the performance of the fair share scheduler. The compute requests in this balanced trace goes well above the total available capacity in the cloud, so the fair share algorithm appropriates the scarce capacity proportionally among all the VOs depending on their contribution. Since VO 1 is the largest contributor, it pretty much

gets all its requests fulfilled. The reason utilisation peaks at about 80% is that the fair share scheduler is saving the free capacity at this point for heavy contributors. Since VOs 6, 7 and 8 are very light contributors, their requests gets rejected at this point. This is clearly seen in the VO resource consumption graphs. It should be noted that, as far as utilisation goes, the random scheduler manages to best the fair share scheduler. However, it does this at the cost of fairness, treating each VO equally instead of rewarding VOs that contribute more. The utilisation tradeoff for achieving fairness the fair share scheduler makes in this regard is therefore the right behaviour for a social cloud.

Chapter 6

Conclusions

Social networks has brought a revolution in how people interact and share information. In the scientific research community, the opportunities provided by social networks presents a paradigm shift in how collaborative research can be done. The aim of this thesis is to build upon the strengths of social networks, taking advantage of its superior collaboration capabilities, to build a Virtual Research Environment (VRE) that would enable dynamic sharing of computing resources among scientific researchers. As a result, the Social Collaborative Cloud (SoCC) presented in this thesis fuses the previously separate worlds of cloud computing and social networks.

In order to realise the goals of SoCC, it was first necessary to identify ways to adapt a social network environment to a VRE. Facebook, which was selected as the social network for the SoCC prototype, provided a straight forward way of achieving this. Its user interface already had the necessary features for a VRE, and its development model provided an ideal way to integrate SoCC into the facebook UI.

To harness the collaborative and social capabilities of facebook, such as groups and chat boards, SoCC was designed not to duplicate any existing functionality of facebook. Thus, common activities such as authentication, role and membership management were all done using the facebook UI. The result was a SoCC prototype that felt as an extension to facebook in-

stead of a third party application. Users could use the familiar interface of facebook to share documents and files, form online communities and do group chatting and video conferencing, and use SoCC only to harness and contribute computing resources.

In the end, while realising the collaborative capabilities of social networks, SoCC solves another important problem in conducting scientific research – the scarcity of computing resources. Computing capabilities has always been a constrained resource traditionally provided by institutional grids, and more recently by public cloud vendors such as Amazon. SoCC solves this problem by allowing rapid and dynamic sharing of computing capabilities among the social network members. The fair share scheduler of SoCC allows users to contribute the limited computing capability they have, and in return gets access to the resources contributed by other members.

6.1 Major Contributions

In summary, this thesis has made the following major contributions:

1. *Identified a strategy for adapting the facebook interface to a VRE.* The various extension points of the facebook interface (such as groups and content areas) was utilised using the facebook developer API to integrate the SoCC framework into facebook. This effort resulted in a SoCC architecture that semlessly blends with the facebook interface. Facebooks group feature in particular was used to provide the Virtual Organisation functions of a VRE. This allows users to manage VRE using the familiar and easy to use group management tools of facebook, with all changes made seamlessly reflected in the VO.

2. *Architected a framework to integrate a Social Cloud with the facebook social network, and implemented a facebook app to realise it.* The solution requires only minimal information to be relinquished by the user,

namely the users facebook id and group information. With such minimal information, duplication of data was avoided as much as possible. An important side effect of this is that the users facebook security and privacy settings was carried seamlessly across to the SoCC application. For example, since the SoCC VO and facebook group has a one-to-one mapping, any changes made to the facebook group member permissions is immediately reflected in the SoCC VO members.

3. *Architected a resource sharing framework that allows users to dynamically share their resources, and implemented a scheduling framework that encourages contribution while making a best attempt to give more in return.* Using Virtualisation Management Middleware, it is not difficult for users to setup a small cluster using a few machines. The Local Agent in the architecture is very light-weight, requiring minimal effort on behalf of the users to implement it.

   The architecture enables users to consume the cloud using an IaaS or PaaS model. The IaaS model allows users to customise the required VM specification such as CPU, memory etc, and launch a set of VMs with the required specifications to run their workloads. Or if a user needs to deploy a VM quickly, he or she can use the streamlined PaaS model to select a pre–configured VM image and easily launch it with a single click to run a computation.

   The SoCC scheduler is an adaptation of the fair share scheduler to the SoCC environment. It ensures that all users are able to use the cloud during low utilisation periods, allowing VOs with resource scarcity among its users to make use of the cloud. During high utilisation periods, the scheduler favours VOs whose users contributes more, thereby giving an incentive for contribution.

4. *Demonstrated the workings of the SoCC prototype via real world tests and simulations.* The real world tests exercised two use cases to demon-

strate the functioning of the prototype, while the simulations evaluated the effectiveness of the scheduling algorithm.

6.2 Future Work

The work done on SoCC in this thesis only demonstrates the viability of the social cloud concept. More work is required to make it practical in a production environment. The following are a few suggestions for future work in this field, that I believe provides for interesting research subjects.

1. *Integrate SoCC into more social networks.* As mentioned in the requirements for the SoCC prototype, the reason facebook was chosen is because of its mature development mode. However, social networks like LinkedIn and Google+ serves an important user base that would be an ideal target for SoCC. I believe any future production version of SoCC must integrate with these social networks as well.

2. *Implement a robust security model for SoCC.* In this thesis security issues was largely ignored. This was because the resulting increase in the scope of the thesis that would have resulted if that was not the case would not have sat well with the time scale set for this thesis. However, a social environment provokes some interesting challenges in this area. Any practical SoCC implementation must deal with fake users, cheating users, Denial of Service (DoS) attacks etc.

3. *Extension to SaaS model.* The SoCC prototype was targeted towards scientific researchers. As such an IaaS/PaaS model was concluded to be more suitable. However, a social cloud such as SoCC could also target researchers in other fields, such as economists and artists. A SaaS model would serve this group well. For example, SoCC could be extended to deploy an online photo gallery host in one-click, or a web site for a conference in a similar manner.

4. *Integration with Commercial Cloud Vendors.* If it could be done, this could have immense benefit. For example, Amazon Simple Storage Service (Amazon S3, [3]) is a very popular cloud storage facility. It would be natural for a scientist to use S3 as a staging area for datasets. Some level of integrated support to ease the use of such services would make SoCC more appealing for scientists.

Bibliography

[1] Amazon ec2 instance types. http://aws.amazon.com/ec2/instance-types/.

[2] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/.

[3] Amazon simple storage service. http://aws.amazon.com/s3.

[4] Boinc volunteer computing framework. http://boinc.berkeley.edu/.

[5] Facebook vs. twitter vs. linkedin vs. google+. http://blogs.computerworld.com/18603/facebook_vs_twitter_vs_linkedin_vs_google_plus.

[6] Gogrid. http://www.gogrid.com/.

[7] Gogrid. http://www.rackspace.com/.

[8] The grid workloads archive. http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Main.Home.

[9] Linkedin vs. facebook for business in 2011. http://socialmediatoday.com/nealschaffer/252062/linkedin-vs-facebook-business-2011-battle-begins.

[10] Open cloud computing interface. http://occi-wg.org/.

[11] Opennebula virtualisation management middleware. http://opennebula.org/.

[12] Progress thru processors. http://www.facebook.com/progressthruprocessors.

[13] Seti@home volunteer computing project. http://setiathome.berkeley.edu/.

[14] ANDERSON, D. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (nov. 2004), pp. 4 – 10.

[15] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., AND ZAHARIA, M. Above the clouds: A berkeley view of cloud computing. Tech. rep., 2009.

[16] BANGA, G., DRUSCHEL, P., AND MOGUL, J. C. Resource containers: a new facility for resource management in server systems. In *Proceedings of the third symposium on Operating systems design and implementation* (Berkeley, CA, USA, 1999), OSDI '99, USENIX Association, pp. 45–58.

[17] CHARD, K., BUBENDORFER, K., CATON, S., AND RANA, O. Social cloud computing: A vision for socially motivated resource sharing. *Services Computing, IEEE Transactions on PP*, 99 (2011), 1.

[18] CHARD, K., CATON, S., RANA, O., AND BUBENDORFER, K. Social cloud: Cloud computing in social networks. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (july 2010), pp. 99 –106.

[19] CURRY, R., KIDDLE, C., MARKATCHEV, N., SIMMONDS, R., TAN, T., ARLITT, M., AND WALKER, B. Facebook meets the virtualized

enterprise. In *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE* (sept. 2008), pp. 286 –292.

[20] DAHAN, M., NUTHULAPATI, P., MOCK, S., DOOLEY, R., HURLEY, P., AND BOISSEAU, J. Increasing teragrid user productivity through integration of information and interactive services. In *Grid Computing Environments Workshop, 2008. GCE '08* (nov. 2008), pp. 1 –11.

[21] DE ROURE, D., GOBLE, C., BHAGAT, J., CRUICKSHANK, D., GODERIS, A., MICHAELIDES, D., AND NEWMAN, D. myexperiment: Defining the social virtual research environment. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on* (dec. 2008), pp. 182 –189.

[22] GRIT, L. E., AND CHASE, J. S. Weighted fair sharing for dynamic virtual clusters. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2008), SIGMETRICS '08, ACM, pp. 461–462.

[23] JOHN, K. The social cloud for public eresearch. Master's thesis, School of Computer Engineering and Computer Science, Victoria University of Wellington, 2010.

[24] JUVE, G., DEELMAN, E., VAHI, K., MEHTA, G., BERRIMAN, B., BERMAN, B., AND MAECHLING, P. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on* (dec. 2009), pp. 59 –66.

[25] KIM, K. H., AND BUYYA, R. Fair resource sharing in hierarchical virtual organizations for global grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing* (Washington, DC, USA, 2007), GRID '07, IEEE Computer Society, pp. 50–57.

[26] KLIMECK, G., ADAMS, G., MADHAVAN, K., DENNY, N., ZENTNER, M., SHIVARAJAPURA, S., ZENTNER, L., AND BEAUDOIN, D. Social

networks of researchers and educators on nanohub.org. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on* (may 2011), pp. 560 –565.

[27] KRAWCZYK, S., AND BUBENDORFER, K. Grid resource allocation: allocation mechanisms and utilisation patterns. In *Proceedings of the sixth Australasian workshop on Grid computing and e-research - Volume 82* (Darlinghurst, Australia, Australia, 2008), AusGrid '08, Australian Computer Society, Inc., pp. 73–81.

[28] LAADAN, O., AND NIEH, J. Operating system virtualization: practice and experience. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference* (New York, NY, USA, 2010), SYSTOR '10, ACM, pp. 17:1–17:12.

[29] LENK, A., KLEMS, M., NIMIS, J., TAI, S., AND SANDHOLM, T. What's inside the cloud? an architectural map of the cloud landscape. In *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on* (may 2009), pp. 23 –31.

[30] LUNDSTROM, M. Nanotechnology and cyberinfrastructure: The nanohub experience. In *University/Government/Industry Micro/Nano Symposium, 2008. UGIM 2008. 17th Biennial* (july 2008), p. 111.

[31] PRODAN, R., AND OSTERMANN, S. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In *Grid Computing, 2009 10th IEEE/ACM International Conference on* (oct. 2009), pp. 17 –25.

[32] ROTH, B., HECHT, R., VOLZ, B., AND JABLONSKI, S. Towards a generic cloud-based virtual research environment. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual* (july 2011), pp. 267 –272.

[33] ROURE, D., GOBLE, C., ALEKSEJEVS, S., BECHHOFER, S., BHAGAT, J., CRUICKSHANK, D., FISHER, P., KOLLARA, N., MICHAELIDES, D., MISSIER, P., NEWMAN, D., RAMSDEN, M., ROOS, M., WOLSTENCROFT, K., ZALUSKA, E., AND ZHAO, J. The evolution of myexperiment. In *e-Science (e-Science), 2010 IEEE Sixth International Conference on* (dec. 2010), pp. 153 –160.

[34] SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., BAVIER, A., AND PETERSON, L. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), EuroSys '07, ACM, pp. 275–287.

[35] STANOEVSKA-SLABEVA, K., WOZNIAK, T., HOFFEND, I., AND EBERMANN, J. Towards a concept for inclusion of social network information as context information. In *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on* (oct. 2009), pp. 1 –5.

[36] SUBRAMANIAN, V., MA, H., WANG, L., LEE, E.-J., AND CHEN, P. Rapid 3d seismic source inversion using windows azure and amazon ec2. In *Services (SERVICES), 2011 IEEE World Congress on* (july 2011), pp. 602 –606.

[37] TOMÁS, L., ÖSTBERG, P.-O., CAMINERO, B., CARRION, C., AND ELMROTH, E. An adaptable in-advance and fairshare meta-scheduling architecture to improve grid qos. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing* (Washington, DC, USA, 2011), GRID '11, IEEE Computer Society, pp. 220–221.

[38] ZHAO, Y. Service oriented infrastructure framework. In *Services - Part I, 2008. IEEE Congress on* (july 2008), pp. 99 –100.