Trust-based Accounting in Grid Systems

Peter Birkestrøm Due

Kongens Lyngby 2006
IMM-MASTER-2006-17

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

Distributed systems which offer access to computing resources via the use of the Grid paradigm, it is important to be able to find a suitable "price" which is to be paid for the use of these resources.

In traditional centralised accounting systems in computers, the price is simply a function of the number of resources used, the length of time for which they are occupied and possibly the urgency of the computing task must be completed. In turn this means that the price will depend on the load on the system and the Quality of Service desired by the user.

In Grid systems, where previously unknown users may start to use the system, the situation is slightly more complicated, because it may not be possible to trust all the users to fulfill their part of the contract, i.e. to keep within the limits of resource usage which they have asked to be allocated. Similarly, a user may not necessarily trust a provider to provide correct results or to supply the agreed resources. A just price for resource usage may therefore depend on the risk which the user and the resource provider are willing to run, and to the degree the provider and user trust one another.

In this M.Sc. project, the aim was to develop a system for trust management which can be used to evaluate a reasonable price for use of computing resources, with due consideration for the degree to the trust relationship between the site and user and vice versa.

The fields studied was the Grid architecture for Computational Economy, Beta trust reputation and revenue management. It lead to a model formulation for price calculation based on a trust value and lead to a heuristic for how a site and user should negotiate with each other. An simulation program was implemented

to investigate the price calculation and to find the optimum resource allocation from an economical point of view.

Keywords: Grid systems, Economical Grid, reputational trust, revenue management, optimum capacity allocation, price calculation.

Sammenfatning

I et distribueret miljø som tilbyder adgang til computer ressourcer vha. Grid paradigmet, er det vigtig at kunne finde en passende pris, som skal betales for at benytte disse ressourcer.

I traditionelle centraliseret computer accounting systemer, er prisen en simpel funktion af det antal ressource man har brugt, det tidsrum det har været brugt i og måske med hvilken prioritet som opgaven skal gennemføres efter.

I et Grid system, hvor tidligere ukendte brugere kan benytte systemet, er situationerne mere kompleks, da det ikke er sikkert at man kan stole på at alle brugere opfylder del af en kontrakt. I denne kontrakt er f.eks. sat en grænse for hvor meget en ressource må bruges. Tilsvarende kan en bruger ikke altid stole på at en udbyder giver det rigtige resultat eller allokere den mængde ressource som aftalt. En korrekt pris for en ressource kan derfor måske afhænge på den risiko som en bruger og en udbyder er villige til at tage samt til hvilken grad brugeren og udbyderen stoler på hinanden.

I dette projekt har målet været at udvikle et system for at håndtere tillid, som kunne bruges to at lave en fornuftig pris for en computer ressource med hensynstagen til de tillidsforhold som udbyder og bruger har.

Følgende felter er blevet undersøgt: Grid arkitektur for computer økonomi, Beta tillidsrygte og indtjeningsstyring. Dette til en modelformulering for prisberegning baseret på tillidsværdier og førte til heuristikker om hvordan bruger og udbyder skal kommunikere med hinanden. Et simulerings program var lavet for at undersøge pris beregning og for at finde optimalet ressource allokering ud fra et økonomisk synspunkt.

Nøgleord: Grid systemer, Økonomisk Grid, Rygte tillid, Indtjeningsstyring, Optimal allokering af kapacitet, pris beregning.

Preface

This thesis was carried out in the period of September 1st 2005 to February 28th 2006 at Informatics Mathematical Modelling,the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Master degree in engineering. The thesis was supervised by Robin Sharp.

The thesis deals with different aspects of mathematical modeling of a computational grid using data and partial knowledge about the structure of computational grids. The main focus is on extensions on the economic aspects of computational grids.

Lyngby, Februar 2006

Peter Birkestrøm Due

Acknowledgments

I wish to thank my supervisor for his interest in my project and for his constructive criticism and inspiration during the period.

I also wish to thank Jan Overgaard at Novo Nordisk Scientific Computing for his introduction to the grid in commercial way.

The Danish Grid Forum and the people on Morpheus at the Niels Bohr Institute get my thanks for their response and helpfulness in regards to collect real data on the Grid.

I wish to thank my girlfriend and my family for their support and encouragement at times when this was especially appreciated.

Contents

List of Figures

Chapter 1

Introduction

Since the concept of computational grids has emerged, the development has been going fast. The scientific community saw the potential for distributed computing very fast and is leading today in the usage of a Grid system. The object of a computer grid is like a power grid, you plug into a socket and then you have power. In this case you just have computer power ready to be used. In Europe CERN is pushing in an attempt to utilize this idea to the maximum extent in order for them to have en computational power ready when their Large Hadron Collider is done, which is scheduled to be in 2007. They have started the LHC Computing Project(LKC), with the purpose of building and maintaining a data storage and analysis infrastructure for everyone that uses the LHC[Knobloch and L.Robertson, 2005]. Since the projected data use amounts to 15 Petabytes annually and a CPU capacity of approx 25000 cpus [Eck, 2005], the terms trust and accounting is becoming more and more prevalent.

Trust since there will be more and more transactions where the users are previously unknown. This is problem, since there is no guarantee that the user will keep within the limits allocated by the system. When the limit is breached, there has to be a consequence one way or another. One way could be to let the trust be a parameter in the price the user has to pay for the resource usage.

Another issue regarding accounting is the demand for many cpus. The question

is how much of this should governmental funded and how much of this could be handled by private hosting. In order for this to happen there has to be a computational economy in place.

Large corporation like Intel, IBM, Silicon Graphics, Sun and Microsoft all invest large sums into their research in GRID computing. Intel can use the grid to simulate the communications on embedded chips, IBM can see a potential in consolidating cpu power in large centers and Microsoft believe that after a computational grid, there will come a data grid, where their SQL servers will be ready.

All of this requires an infrastructure, that can handle pricing and trust. This thesis will focus on how to derive a "just"' price for the computational power in regards to trust.

1.1 Motivation

To find a price is not just a question of looking at the costs. There are many other parameters to be considered. First of all where is the trade taking place. From the point of view of thesis, this is done on a computational grid where accounting mechanisms exits but is not used. A study of relevant literature shows that there is only one, who has looked into the economy of resource usage and the chapter detailing this shows that there are still much to be done in this area.

With regards to trust it is all within the realm of authentication and authorization. What is unique in a Grid system is that users and sites initially don't have an established trust relationship and hence it is difficult to determine if the user or the site is trustworthy.

One way of solving this is to use reputational trust and combined this with a direct trust. This is used in this thesis as an input for the price.

In regards to economy there are many ways to calculate a price and in order for this to make sense there has be a definition of value.

This thesis investigates how to use revenue management as well as bond theory,

since it could be applied on a Grid system. The usage of revenue management is could be used to at make the sites more attractable with the use of discount prices. The revenue management is could also be used to find the optimally allocation of resources on the resource sites. This is investigated if this can be applied to a Grid system.

1.2 Structure of the Report

The structure of the report is a follows:

Chapter 1 is the introduction.

Chapter 2 introduces the concept of Grid Architecture for Computational Economy (GRACE) and other proposed architectures for the grid.

Chapter 3 defines the trust notation and analyzes a trust simulator.

Chapter 4 investigates financial theory with regards to costs and capacity planning. Furthermore revenue management and bond theory is analyzed.

Chapter 5 presents a pricing model and show process flows for how the users and sites interact. A pricing function for resource usage is also derived.

Chapter 6 analyzes the simulation requirements and presents the different components in the simulation.

Chapter 7 presents simulation experiments and introduces the different test cases. The results of the experiments are also presented.

Chapter 8 presents a discussion of the simulation results and possible future work.

Chapter 9 concludes on the work presented in this thesis.

Chapter 2

Background: The Grid

2.1 Overview

The Grid is a service for sharing computer power and data storage over the Internet. The aim of is to turn a global network of computers into one vast computational resource and the idea is shown in figure 2.1 on the following page. Since jobs on the Grid normally involves large-scale resource intensive applications, the need for resource management is ever present. But job management is very complex as resources are geographically distributed, heterogeneous, and owned by different organizations, firms etc. who have diverse policies etc. Furthermore different sites all have different access and cost models.

In this chapter two architectures are investigated with the aim of establishing an economic model to be employed on a Grid system.

The analyzed models are:

- Grid Architecture for Computational Economy (GRACE)
- G-commerce

Figure 2.1: Generic view of the Grid

2.2 Grid Architecture for Computational Economy

In order to make an economic model, one has to realize that the producers (resource owners) and consumers (resource users) all have different goals, objectives and strategies. In his Ph.d. thesis Rajkumar Buyya proposed [Buyya, 2002] a framework for economy-based distributed resource management and resource scheduling for Grid computing. Buyya points out that there are many applications for the Grid such as Globus [1] and Legion [2], but none address the issue of computational economy.

In order to make a real world scalable Grid system there is a need for computational economy in order have mechanisms to regulate supply and demand.

Buyaa introduces the following terms for the two major players in GRACE as shown on figure 2.2 on the next page:

Ressource Providers These are also called Grid Service Providers (GSP), who owns and administrate resources. They seek to maximize their profit

[1] http://www.globus.org/
[2] http://www.cs.virginia.edu/ legion/

and consequently their resource utilization.

Resource Consumers These are also called Grid Resource Brokers (GRB). The GRB's act as the consumer's software agent. The consumer's strategy is to have its jobs solved in the shortest amount of time within a given time frame. The brokers are also referred to as super-schedulers.

In the Grid environment Buyaa also introduces the Grid Marked Directory(GMD), which discovers resources on the grid and handles communication with the Broker. Each resource site has a Grid Trade Server (GTS), that handles current jobs and makes sure that the site is listed in the GMD. Also Deal Template is introduced as a form of contract between the GSP and GRB. This gives a typical market-oriented Grid environment as shown in figure 2.2 It is a prerequisite that there is a bank in the environment, that controls the flow of money.

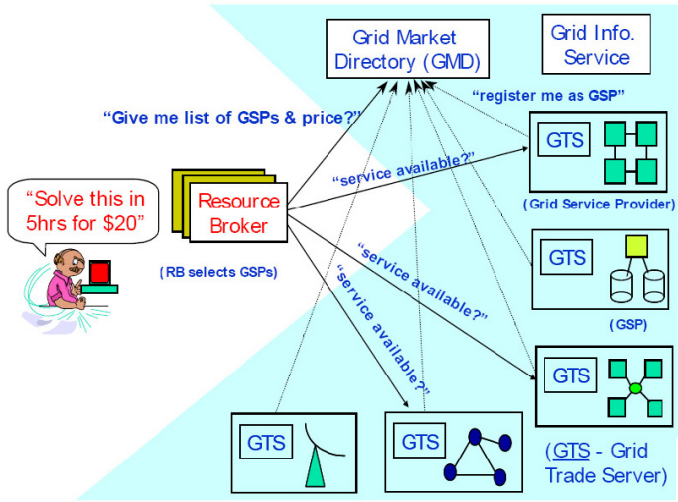

Figure 2.2: Generic view of interaction between players in an Economy grid

The GSP and the consumers are meant to communicate as follows:

1. Client (agent) initiates communication with the trade server at the GSP to obtain the price for resource usage,
2. Negotiation takes place according to the current GSP's pricing scheme,
3. Client and GSP agrees and the trade server requests a credit check in the bank.

4. Job is added to the site's job queue with the state wait.
5. Job is started when Bank returns with a credit check ok.
6. Job starts
7. Job finishes.
8. Resource usage is accounted and money is transfer to the site's account in the bank.

This way no job is started without the user having money.

The broker's job consists of the following:

1. The broker finds GSP's
2. It identifies suitable resources and locates a price (negotiation with the different trade server listed in the GMD).
3. It selects the GSP which matches the utility function and deadlines the user has specified.
4. Job is submitted and money is subtracted from the user's account in the bank.

In a given grid economy the economic model has to be defined for the market. The are many models but the ones relevant to Grid systems are:

- A Commodity Market Model
- Posted Price Model.
- An Auction Model

In a commodity market the resource providers set the price and advertise their prices. The pricing policies can be static or dynamic and the users choose sites after a cost-benefit analysis. The Posted Price Model is the same as the Commodity Market Model except the sites give the price before the scheduling. In the Auction Model the GSP invites bids and stops when there are no more new bids from users.

Which model to choose is a matter of optimization. Section 2.3 on the next page includes there is a comparison between the Commodity and Auction Model.

A fixed pricing scheme is ill suited as it does not support various levels of Quality of Service for the consumers whereas consumer requirements will be biased in terms of runtime, price, etc.

The demand function of the consumers needs to be investigated in order to do a reasonable forecast function. The reason for this is in a population of quality-sensitive buyers all pricing strategies leads to a price equilibrium, where in a population of price sensitive buyers the strategies leads to price wars. Byuaa [Buyya et al., 2001] defines the following items to be included in accounting in order to locate a price:

- CPU usage time
- Memory
- Maximum resident set size - page size
- Amount of memory used
- Page Faults
- Storage used
- Network activity
- Signals received
- Utilized software libraries

Another parameter for the price specification could be time of the day etc. Other parameters can be:

- Consumer id
- Peak timer price
- Lunch time price
- Offpeak time price
- Discount price (not fully loaded)

- High rate (full load)
- Holidays price
- Wall clock time price (within 2 hours, over 2 hours)

Today the price is defined from production cost and profit margins. But the perceived value of resource differ between producer and consumer. The following value function is assumed:

Resource value = f(Physical costs, Service overhead, Demand, The value from the users point of view, Preferences)

This is the price function, but the three last parameters are subjective and valued differently by different users. Preferences is when the job has to be executed etc.

2.3 G-commerce

This is another model for resource allocation in a Computational Grid. The model is called G-commerce[Wolski et al., 2001] and is built on the notation of a market economy. It focuses on 2 categories of economic models: Commodities Market and Auctions. The commodity market is used, since the grid tries to use resources as interoperable commodities. The auction market requires no global pricing information and is easy to implement.

The following criteria are used to evaluate the 2 categories:

1. Grid-wide price stability
2. Market equilibrium
3. Application efficiency
4. Resource efficiency

The price stability is required in order to maintain scheduling stability. If the price fluctuates the overall performance will be poor since the scheduling efficiency will decrease. Market equilibrium is a measure of how fair the price

is. How ever this assumptions only holds if the users are quality-sensitive as mentioned in 2.2 on page 6. Application efficiency measures how effective the Grid is as a computational platform. Resource efficiency measures how well the grid utilizies it resources.

In the article by Wolski [Wolski et al., 2001] the following assumption is made:

The relative worth of a resource is determined by its supply and the demand for it

In other words it is not the production cost or profit margins alone that defines the value of a resource. Another criteria is the Market equilibrium 2.3, which is a means of price adjustments.

In the simulation section in the article [Wolski et al., 2001] the commodities are CPU and disk storage and the result turns out to be that a commodities market is a more suitable choice than an auction market.

Market Equilibrium The market equilibrium is the work of Debreu[Debreu, 1959] A market economy consists of produces, consumers and supply and demand functions for each commodity, which is determined by a set of market prices for the various commodities. A unique equilibrium price is guaranteed to exist according to the theorem of Debreu[Debreu, 1959]. For all commodities their prices are represented by the price vector $p = (p_1, p_2 \dots p_n)$, then the excess demand vector is defined as z_j for the j^t commodity. It is assumed that all z's are interrelated so that each z_j is a function all prices in p. This means there is a equilibrium point p^* such that $z(p^*) = 0$ exists. This also means that for any value of p the $n \times n$ matrix can be derived.

$$D_z(P) = \frac{\partial z_i}{\partial p_j} \tag{2.1}$$

This means for any value of λ, which has the same sign as $D_z(P)$, the economic equilibrium by obeying the differential equation

$$D_z(P)\frac{dp}{dt} = -\lambda z(p) \tag{2.2}$$

2.4 Summary

In this chapter I have shown current status of a economical Grid system. The 2 most dominant ways are the GRACE and G-commerce. GRACE is an architecture for how the process flow should be for an economical Grid system. It also shows which parameters should be accounted for and how they should influence the price. However GRACE does not answer which market condition that apply for a Grid system, but it sums up how the work flows differs compared to the market condition. G-commerce believes that the market containing Grid systems will either be a commodity market or an auction market. Furthermore it is believed that on a market the governed condition is market equilibrium. With this condition it shows that commodity market is more applicable than an auction market.

Chapter 3

Background: Trust Reputation Systems

3.1 Overview

Trust is easy for humans to understand. When somebody behaves good, they are to be trusted whereas when somebody behaves bad the trust is low. In the business world people look at the risk of a economic loss if customers do not pay. The trust relationship is neglected initially. Over time the customer will be granted more and more trust as long as transactions are successful. If the customer ill behaves the firm will not decline to have any further business with that person.

In this chapter the trust notations that apply tot he above mentioned scenario will be investigated. Especially trust reputation systems as mentioned by Jönsang [Audun Jøsang and Faccer, 2003] will be investigated. The trust simulator by M. Brinkløv [Brinkløv and Sharp, 2006] is also investigated.

3.2 Trust

3.2.1 Notation of Trust

In order to handle trust there is two leading definitions in the literature which is something called *reliability trust* and *decision trust*, and they are defined as follows:

Reliability Trust Trust is the subjective probability by which an individual A, expects another individual B, to perform a given action on which B's welfare depends.[Gambetta, 1990]

Decision Trust Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible.[Mcknight and Chervany, 1996]

Notice in the first definition there is dependence on the trusted party, reliability of the trusted party whereas in the second also risk attitude is accounted for. You might do an action even though negative consequences are possible. In other words there is a trade-off.

3.2.2 Reputational Trust

Since the Internet was invented people have used it to perform business transactions in small and large scale. There are eBay where people sell and buy goods, there is large invoice systems like the one at Novozymes [1] where the customers can buy enzymes on-line. Often these transactions take place where the players often have insufficient information about each other. In order to create trustworthiness you could create a reputation system. Reputation is by definition what is generally said about or believed about a person's or thing's character or standing [Jøsang and Ismail, 2002]. In an e-Commerce environment reputation can be used to generate trust among the participants. Such a system is called a reputation system and its objective is to gather, distribute and aggregate feedback about participants. Audun Jøsang proposes a reputation engine in [Jøsang and Ismail, 2002], which is based on the beta probability density function. The reputation engine calculates the user's reputation ratings from various inputs including feedback from other users.

[1] http://www.novozymes.com/

The framework for such a system can be seen in figure 3.1 as described in [Audun Jøsang and Faccer, 2003].

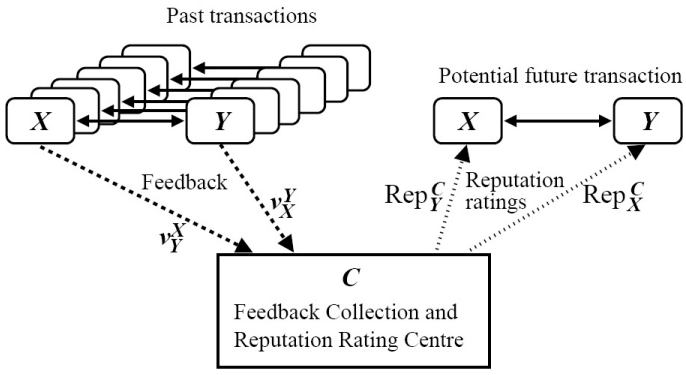

Figure 3.1: General framwork for a reputation system

The reputation center collects the input from all agents and updates each agents reputation rating as a function of feedback. These updated ratings are online and can be used for agents to decide if the agents wants to transact with another agent. In Jøsangs beta reputation system [Jøsang and Ismail, 2002] the backbone of the system is the beta probability density function, which can be used to represent probability distributions of binary events. In the next paragraph the beta density function is elaborated.

Beta Density Function Beta function uses the two 2 parameters α and β, who can be expressed as a Γ function:

$$f(p|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}, \text{ where } 0 \leq p \leq 1, \alpha \succ 0 \text{ and } \beta \succ 0$$

This gives a probability value of the beta distribution:

$$E(p) = \frac{\alpha}{(\alpha+\beta)} \tag{3.1}$$

If you have a process of 2 possible outcomes $\{x, \overline{x}\}$, where r is the observed number of x and s is the observed number of $\overline{x}$, then the future possible outcome of x can be expressed as:

$$\alpha = r + 1 \text{ and } \beta = s + 1 \; where \; r, s \geq 0$$

If you have 11 observations of x and 7 observations of $\overline{x}$ it matches a beta function $f(p|12, 8)$ which is plotted in figure 3.2

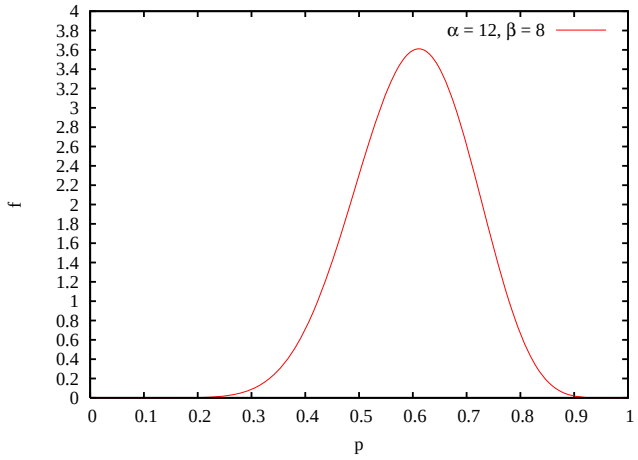

Figure 3.2: Beta function of event x after 11 observations of x and 7 observations of $\overline{x}$

This gives an $E(p) = 0.6$ which indicates that the outcome of x is uncertain. Had the value been 0.9 the future outcome of x would have been high. This is the mathematical background for the reputation system and reputation rating in 3.1 on the preceding page.

Beta Reputation Since it is build on the beta distribution, the agent will always have a reputation rating between 0 and 1. The definition of reputation rating is:

Reputation Rating Let (r(Z),s(Z)) be Z's reputation parameters and let (r_{base},s_{base}) be the a priori parameters. r(Z) is the positive feedbacks while s(Z) is the negative. Then the reputation Rep(Z) will be:

$$Rep(Z) = \frac{r(Z) + r_{base}}{r(Z) + s(Z) + r_{base} + s_{base}} \tag{3.2}$$

This score is for one agent, but in order to be a reputation system it has to be possible to combine more than one rating. To do this the combined feedback from X's and Y's feedback on target Z is defined as follows:

$$r^{X,Y}(Z) = r^X(Z) + r^Y(Z) \tag{3.3}$$
$$s^{X,Y}(Z) = s^X(Z) + s^Y(Z) \tag{3.4}$$

Now the system can combine feedback. But some agents is more highly reputed than other and therefore should carry more weight than others. This is done by discounting the feedback as a function of the reputation of the agent who gives the feedback. To do this an opinion is needed and Jøsang proposes to use Subjective Logic fundamentals [Audun Jøsang and Kinaeter, 2005]. Moreover this can be bijective mapped to the beta distribution. The opinion $\omega_x^A = (b, d, u, a)$ where *b=belief, d=disbelief,u=uncertainty* and *a=base rate*. At all times $b + d + u = 1$ and $a \in [0, 1]$, which is also used to calculate the expected opinion probability density value determined by $E(\omega_x^A) = b + au$.

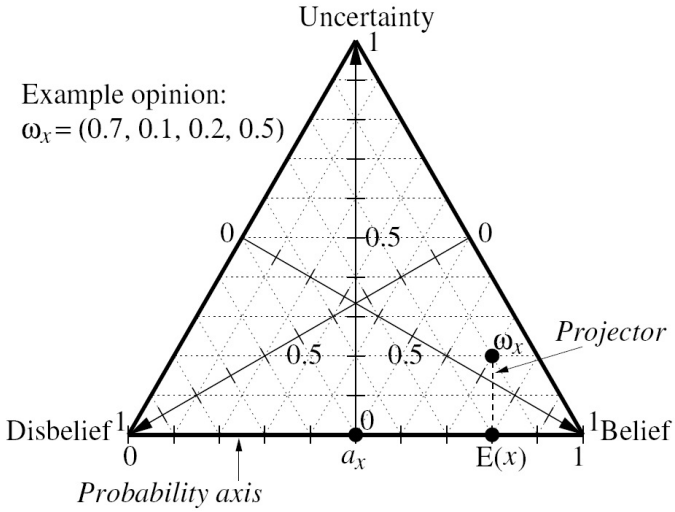

Figure 3.3: Opinion of 7 observations of x and 1 observations of $\overline{x}$

The opinion can be used to define belief discounting as described in [Audun Jøsang and Kinaeter, 2005]. The opinion metric is mapped bijective to the beta function defined by:

$$
\begin{aligned}
b &= \frac{r}{r+s+2}, \\
d &= \frac{s}{r+s+2}, \\
u &= \frac{2}{r+s+2},
\end{aligned}
\tag{3.5}
$$

These lead to the following definition of reputation discounting:

Reputation Discounting x,Y and T are all agents where $\varphi(p|r_Y^X, s_Y^X)$ is Y's reputation function by X, and $\varphi(p|r_T^Y, s_T^Y)$ is T's reputation by Y. Let $\varphi(p|r_T^{X:Y}, s_T^{X:Y})$ be defined as:

1. $r_T^{X:Y} = \frac{2r_Y^X r_T^Y}{(s_Y^X+2)(r_T^Y+s_T^Y+2)+2r_Y^X}$
2. $s_T^{X:Y} = \frac{2r_Y^X s_T^Y}{(s_Y^X+2)(r_T^Y+s_T^Y+2)+2r_Y^X}$

It is also called T's discounted reputation function by X through Y.

This way the system can now handle weigh the different opinions between the agents. The next problem to be solved is that old feedback is not always relevant for the actual reputation rating, since people can change behavior over time. One could go from god to bad and vice versa. The way Jøsang handles this is to let old feedbacks carry less weight than newer feedbacks[Jøsang and Ismail, 2002].

He introduces the feedback longevity factor $\lambda \in [0, 1]$ which is used this way: Z_i is the resulting reputation parameters after i transactions denoted by $(r_i(Z), s_i(Z))$. Then after $i+1$ transaction the parameters is $(r(Z), s(Z))$ and the longevity is introduced:

$$
\begin{aligned}
r_{i+1}(Z) &= ((r_i(Z) + r(Z)) * \lambda \\
s_{i+1}(Z) &= ((s_i(Z) + s(Z)) * \lambda
\end{aligned}
\tag{3.6}
$$

where $0 \leq \lambda \leq 1$. This is also recursive, so it makes it possible to keep reputation parameters on all agents without to store all the feedbacks given.

All of the above mentioned rules will give a feedback system, where it is mathematical formulas, which is the base for the trust ratings. In the next section an implementation of a reputation system is discussed.

3.3 Incremental Trust in Grid Computing

The paper by [Brinkløv and Sharp, 2006] defines that trust on a distributed system consists of two components, identity trust and behavioral trust. The identity trust is mainly used for authentication and gives an á priori trust value. The behavioral trust is also called the incremental trust, since it develops as more transactions has taken place. The behavioral trust consist of a direct trust and a reputational trust. The direct trust is agent A's personal experience of B's behavior where the reputation is discounted rating on B from different parties like agent C, etc.

The problem has always been how to combine the direct trust with the reputation trust. According to [Brinkløv and Sharp, 2006] there are two predominant ways of combining the trust: *discrete models* and *fuzzy logic*. The problem with a discrete model is that it uses linear combination of simple numerical values like -1 for total distrust and 1 for absolute trust. However trust is much more of linguistic concept, so it is proposed to use fuzzy logic instead.

The basic idea of fuzzy logic is that for each behavioral parameter, a set of fuzzy sets is defined in terms of so-called membership functions, which specify the degree of membership of the various sets for the possible values of the parameter. When the result of using several parameters $(x1, x2, \ldots, xn)$ has to be combined to a result y, a set of rules is used of the form:

$$if(x_1 \in S_i \wedge x_2 \in T_j \wedge \ldots)\, then\, y \in O_k$$

where $S_i, T_j, \ldots O_k$ are all fuzzy sets and $\in$ here indicates fuzzy membership. The membership functions μ_k for the output fuzzy sets $O_k, k = 1, \ldots, r$ are very commonly given by the so-called min-max formula:

$$\mu_{Ok}(y) = max[min[\mu_{Sk}(x1), \mu_{Tk}(x2), \ldots]]k = 1, 2, \ldots r \tag{3.7}$$

The actual value is found by defuzzification.

In the simulation in the article various methods are evaluated and the way to combine the trust score is shown in figure 3.4 on the next page

The results show that discrete methods tend to detect change in behavior faster than fuzzy logic, but to find the "right" parameters is much harder. The use of fuzzy logic combined with beta reputation offers a slower detection of behavioral change.

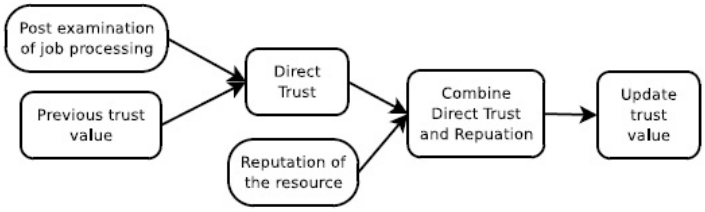

Figure 3.4: Trust score derivation

3.4 Summary

In this chapter I have shown what the trust notation can be. I showed how the Beta distribution can be used to build a reputation system as proposed by Jøsang. He proposes how to handle transactions with two possible outcome, success or failure. This is mapped to a Beta distribution, which enables how to predict possible outcomes. He then maps the Beta distribution to opinion containing the following 4 parameters:

1. Belief
2. Disbelief
3. Uncertainty
4. Base rate

From this a reputation system can be build up. Next it is showed how incremental trust can be applied in a Grid system. The incremental trust is a reputation system which is build on among other things the Beta distribution. It is shown that a trust score can be derived by combining a direct trust with a reputation trust using both fuzzy logic and beta reputation.

Chapter 4

Background: Economic Models

4.1 Overview

This section gives an overview of financial models, that exist today, that can be used in e-Commerce and especially of a Grid system. First of there is the definition of the goods to be sold (assets). Since it is computer power and storage, things are only occupied for a limited period of time, before it can be sold again. This kind of behavior is similar to a hotel with rooms to rent and it is similar to how the tickets are sold for a flight fare. Theories for these industries are evaluated to see if they can apply to an economic model for accounting in a Grid system.

The last section deals with the situation where people buy the option to use a resource within a fixed time frame, but not a specific time. Here the theories of options is evaluated and the black-scholes theorem is investigated.

4.2 Perishable Assets

In order to define the goods on the grid I have looked at the following items, which is to be sold/rented. In a Grid system a job consists of CPU time, how many CPU's the job is expected to use, the amount of storage and memory. In other words the goods are:

1. CPU time
2. Disk capacity
3. Memory capacity

Common to all of these are that you normally have a fixed number of them. If you have a cluster you have certain amount of s, memory and storage. In revenue management there is a term called perishable asset in [Weatherford and Bodily, 1992] where one of the main features is that you have a fixed number. Another feature is the demand is uncertain. The following features are all common for perishable assets[Weatherford and Bodily, 1992]:

- A date from where the service or the product is available and a date from where it is no longer available. The service can not be stored unless it is at a high cost.
- There are a fixed number of units. Furthermore these units have a low variable cost, which enable multiple prices for the same unit. This also means it is always better to sell than let it go to waste. The fixed amount of units is often followed with a wide fluctuation on the demand, which shift the balance between supply and demand.
- It is possible to segment the customers into price-sensitive and quantity-sensitive. In the airline industries it is the time of purchase which is used for segmentation. In a Grid system it is the capacity of the resource site which can be segmented.

The aim with perishable assets must be to find the optimal trade off between average price and capacity utilization. The questions to be answered in such a system on a unit-to-unit basis is:

1. How many units should be made available initially at the various price levels?

2. How should this availability of units change over time?

In other words when is the optimal to change between prices for the same commodity. The theory of revenue management tries to answer this.

4.3 Revenue Management

Normally you want to sell at the time when the market conditions are most favorable and to the right price. This is a difficult business decision to make. In order to help you with this Revenue Management (RM) can provide an answer [Talluri and Ryzin, 2005], as it addresses three basic categories of decisions:

1. Structural decisions: What selling format to use (posted price, negations or auctions).
2. Price decisions: How to set a price over time, how to markdown over the product lifetime.
3. Quantity decisions: When to accept or reject an offer to buy, how to allocate capacity to different segments.

This also leads to a distinction between quantity-based RM and price-based RM. The quantity-based uses capacity-allocation decisions, the price-based uses price as a tool for managing demand.

In order to give an example of industries normally price-based RM is used in retailing and E-business and quantity-based RM is used in airlines.

In relation a Grid system will be quantity-based since it will be controlled by resource sites. However since the aim is to use trust as a parameter for discounting rates, this is price-based RM. The next sections will look into quantity-based RM, since it is highly relevant for the sites in a Grid system.

4.3.1 Single Resource Capacity Control

Single resource capacity control addresses the problem of optimally allocating capacity of different classes of demand for a single resource. This is similar to the sale of different fare classes on a single flight or the sale of hotel rooms for a given date. The term classes refers to different prices for the same good.

It is assumed that the firm sells its capacity in n distinct classes. It is also assumed that the product appeals to distinct and mutually exclusive segments of market. Customers in each class can afford only the class corresponding to their segment. The units of capacity are homogeneous.

The following control mechanisms can be used to control the amount of capacity.

Booking Limits Booking limits are used to limit how much capacity can be allocated within every class at any given time. These can either be *partitioned* or *nested*; Partitioning splits the capacity into separate blocks. *Nested* means that the capacity overlaps each other in a hierarchical manner. This implies that the highest classes have access to the entire capacity allocated for the lower classes. For instance if there is 12 units for class 1, 10 units for class 2 and 8 units for class 3 the total capacity is 30. This way you will always have units available for the highest revenue.

For nested booking limits:

- The nested booking limit for class j is b_j
- Then b_j is the maximum units of capacity you are willing to sell to classes j and lower.

In table 4.1 it is shown.

Class 1	Class 2	Class 3
12	10	8
$b_1 = 30$	-	-
-	$b_2 = 18$	-
-	-	$b_3 = 8$

Table 4.1: Nested booking limits

Protection Limits Another control mechanism is a protection level. Protection level specifies how much of the capacity you can reserve for a specific class. It can be summarized as follows:

- Protection level for j is called y_j

- y_j is defined as the reserved capacity for the lower classes *j,j-1,...,1* all together.

In table 4.1 the protection levels for class 1 be 12, for class 1+2 be 22 and for alle 3 classes it would be 30. This is also shown in table 4.2 The connection between protection limits and booking limits are:

$$b_j = C - y_{j-1},\ j = 2, \ldots, n\ where\ C = capacity, b_1 = C,\ and\ y_n = C \quad (4.1)$$

Class 1	Class 2	Class 3
12	10	8
$y_1 = 12$	-	-
-	$y_2 = 22$	-
-	-	$y_3 = 30$

Table 4.2: Nested protection levels

4.3.2 Common Requirements

In order to solve optimal capacity control the following requirements have to be met:

1. Allocation of resources only happens if its revenue is greater than the value of the capacity required to satisfy it.
2. The value of capacity should be measured by its displacement cost, which is the expected loss in future revenue form using the capacity now than later.

The first requirement one basically states that you will only sell if you can profit. The second requirement is harder to control, but is important since it dictates that the value of capacity is more than just the cost of production.

4.3.3 Static Single-Resource Model

The most common model in revenue management [Talluri and Ryzin, 2005] is the static single-resource model. In order for it to be static the following is assumed:

1. Demand from each class arrives in non-overlapping intervals, but the order is not from low to high revenue.
2. Demand for different classes are independent random variables.
3. Demand for chosen class is not dependent on the capacity controls.
4. It suppresses many details about demand and control mechanism within each period.
5. Group bookings are not considered.

It also assumes risk-neutrality, since maximizing average revenue is most important.

4.3.4 Littlewood's Two-Class Model

This model is a simple two-class model. It is build on Littlewood's work [Talluri and Ryzin, 2005]. It states that there are two classes with associated prices $p1 \succ p2$. The capacity is C and there is no cancellations or group bookings. Demand for each class j is denoted D_j and its distribution is $F_j(\cdot)$. Demand for class 2 arrives first. The question is now when to change from class 2 to class 1. It could be when to go from cheap price to normal price.

Solution Assume there are x units of capacity left and a request for class 2 arrives. If the request is accepted a unit of class 2 is sold. If the request is denied, then the unit will be sold to price $p1$ if and only if the demand for class 1 is x or higher. In other words:

$$D1 \geq x \tag{4.2}$$

The expected gain by reserving the $x^t h$ unit of class 1 is[1] $p1P(D1 \geq x)$, which is also the expected marginal value. This means that you should accept demands for class 2 as long as:

$$p2 \geq p1P(D1 \geq x) \tag{4.3}$$

[1] P is the probability function

An optimal protection limit exists, denoted y_1^*, such that class 2 requests are accepted as long the remaining capacity exceeds y_1^* and the request is rejected if the remaining capacity is y_1^* or less. y_1^* satisfies:

$$p2 \prec p1P(D1 \geq y_1^*) \; and \; p2 \geq p1P(D1 \geq y_1^* + 1) \tag{4.4}$$

If the demand is assumed to follow a continuous distribution $F_1(x)$ the the optimal protection level y_1^* is given by:

$$p2 = p1P(D1 \succ y_1^*), \; equivalent \; to \; y_1^* = F_1^{-1}(1 - \frac{p2}{p1}) \tag{4.5}$$

If the demand *D1* is a normal distribution with mean of μ and standard deviation of σ then Littlewood's rule 4.5 gives:

$$F_1(y_1^*) = 1 - \frac{p2}{p1} \tag{4.6}$$

This leads to:

$$y_1^* = \mu + z\sigma \tag{4.7}$$

where$z = \Phi^{-1}(1 - p2/p1)$ and $\Phi(\cdot)^{-1}$ is the inverse cummulative normal function. This also states that if *p2/p1* is larger than 0.5 the optimal protection limit y_1^* is less than the mean value.

4.4 Hardware Costs

In a Grid system available resources represents a large investment. In order to determine the total investment all costs have to be located[S. Holm-Rasmussen, 2005]. A single server includes the following costs:

- Hardware cost
- Software costs
- Cooling cost
- Power cost
- Depreciation

In addition to this there are fixed costs such as rent, investment in network infrastructure, etc. In accounting this is done this way:

total turnover
- variable costs
= contribution margin
- fixed cost
=earnings before interest rate
- interest rate cost
=earnings before tax
- tax
=profit

In this thesis the main focus will be within the variable costs and particularly the servers costs, since this includes CPU's, storage capacity and memory.

When a server is bought, the normal procedure in Denmark is to let it depreciate over 3 years. Due to legislative regulations depreciations can be deducted from the tax later in the financial year. At the same time it is a recognition that the value of the server is decreasing due to usage. After three years the value of the server is zero, and you would by a new one. The most common way of using depreciation in accounting is on an annual basis, so the invested amount is divided into 3 equally large amounts (linear depreciation), which is deducted annually.

Another parameter in the hardware cost is the profit. In order to make a profit, you use a markup percentage [S. Holm-Rasmussen, 2005] which is added on a yearly basis. When doing business there is always the risk of people not paying, which is accounted for by adding a percentage a year. In other words the total cost will be:

$$totalcosts = (initialcost)*(1+depreciation)^{year}*(1+markup)^{year}*(1+risk)^{year} \tag{4.8}$$

This does not take into account how much energy the server uses. This has to be taken into account when calculating the total cost. This also applies to cooling.

4.5 Black-Scholes Model

Since large computational tasks easily can take many days to complete, there has to be a mechanism for charging the adequate amount for this task. From a providers point of view the earnings from a job taking days to be better than the displacement cost mentioned in section 4.3.2 on page 25. Seen from the consumers point of view it could be nice to reserve the capacity but not necessarily use it right away. If this was stock trade instead of computer economy one would issue a bond or option the stock. There are two types of options - the call option and the put option.

Option Gives the holder right, but not the obligation, to buy or sell a given asset on an agreed price before or on the exercise date[2]

Call-option The buyer has the right to buy the underlying asset. The seller has the obligation to sell the underlying asset at the agreed price

Put-option The buyer has the right to sell the underlying asset. The seller has the obligation to buy the underlying asset at the agreed price

In the context of a Grid system, the call option should be used for the service between consumer and resource and the put option for the relation between 2 resource sites. One could imagine that a site bought put options in other resource sites, in order to save its own capacity.

In this thesis only the call option will be investigated.

In order to put a price on the option there is a theorem called classical Black-Scholes Model [Frey, 1997] which can be used.

In the Black-Scholes model the market that is considered contains a risky asset, the stock, and a riskless bond B that is being traded. The price fluctuation on the stock are described by a Stochastic process $X = (X_t)_{0 \leq t \leq \infty}$, which is defined in the probability space (Ω, F, P) where $(F_t)_t \geq 0$ and $B_t = 1$ for all $t \geq 0$[2].

A typical model for price process of the stock is the *generalized Black-Scholes model*, where X is given by the solution to the following Stochastic differential

[2] http://en.wikipedia.org/wiki/Option

equation:

$$dX_t = \mu(t, X_t)X_t dt + \sigma(t, X_t)X_t dW_t \tag{4.9}$$

W is a standard Brownian Motion in the same probability space. μ and σ are chosen so they are unique solution to 4.9.

The interpretation of 4.9 is that at a given point in time $\mu(t, X_t)$ describes the instantaneous growth rate of the stock, where the *volatility*$\sigma(t, X_t)$ measures the instantaneous variance of the return proces *ln X*. In other words is volatility a metric for the risk taken by investing one unit of money into the stock.

As long as $\sigma(t, X_t)$ is constant or follows at function of time, then model 4.9 is called the *classical Black-Scholes model*.

In order to put a price on the call option you need to know the trading stock price S, where the option has an exercise price K (the right to buy the share at price K at time T years in the future), the constant interest rate r and the constant stock volatility σ. This will give a price C[3] given by:

$$C = SN(d_1) - Ke^{-rt}N(d_2) \tag{4.10}$$

where the rest of the parameters are:

N The cumulative standard normal distribution

d_1 $\frac{ln(S/K)+(r+\frac{s^2}{2})t}{s\sqrt{t}}$

d_2 $d_1 - s\sqrt{t}$

s standard deviation of stock returns

In order to locate K or the strike price, you need to have a forecast function of the stock. This means if the K is higher than S on the exercise day, the seller will earn on it. If on the other hand K is lower than S then the buyer will benefit from it. In terms of a Grid system the stocks will be CPU's, memory, storage and the strike price will be the production cost. If bonds were traded on a Grid system each site needs to have a forecast system to estimate their cost in the future.

[3]definition by Black and Scholes at http://bradley.bradley.edu/ arr/bsm/pg04.html

4.6 Summary

In this chapter I have shown what a perishable asset is, which is a service. Then I show how revenue management with single resource capacity can allocate optimally their resources according to a rule called Littlewood's rule. I then shown how production cost is handle in accounting and in the last section I elaborate on how bond theory with the theorem of Black-Scholes can be applied to a Grid system.

Chapter 5

Model formulation

5.1 Overview

In this chapter the final parameters in the total price are selected. Furthermore the price formula is derived and the inputs for it are selected. Section 5.3 on page 38 shows the different steps the broker and resource site must perform in order to communicate.

5.2 Pricing Scheme for a Grid system

Goods in a Grid system can be defined as perishable assets according to the definition in 4.2 on page 22. On a resource site there are many servers connected in a cluster. This gives the site all its CPU's, all its storage and all its memory. So in order to calculate the price for one job, the price for one server has to be calculated. According to GRACE 2.2 on page 6 there are many parameters to be include in the price. I have selected the ones shown on figure 5.1 on the following page

The parameters are divided into three main categories:

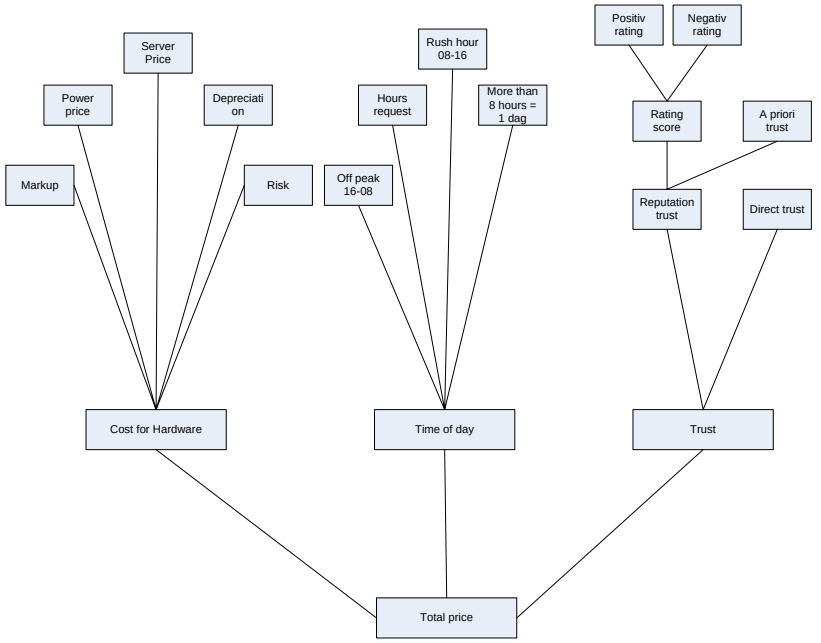

Figure 5.1: Price parameters in the total price

1. Cost for hardware
2. Time of day
3. Trust

The categories are elaborated in the following subsections.

5.2.1 Hardware Cost

In this section the hardware cost is analyzed and set. At figure 5.1 cooling is an variable. Cooling is here meant to be the power usage used for the cooling equipment in the different server rooms at the resource sites. Studies from The Electricity Conservation Fund and Technological Institute [S.Ø. Jensen and Viegand, 2004] show that cooling constitutes about 30% of the total power consumption in the

server room. This means that the only input which the resource site can control is the power price. The rest of the variables are busniess decisions which has to be taken by the managers of resource sites. The variables are then:

Name	Abbreviation	Formula
Purchase price	INVEST	
Power in kWh	PO	
Power usage per day	POU	$24 * PO$
Cooling	COOL	$0,3 * POU$
Depreciation	DEPR	6% p.a
Risk	RISK	10% p.a
Earning	MARKUP	10% p.a
The daily price	DAYCOST	Price per hours for the specific day
Server life time	INVESTPERIOD	1000 days

Notice that the investment period is 1000 days and is a approximation to 3 years which is 1092 days. The number 1000 is easier to work with and that is why it is chosen. The markup is chosen to be 10% and so is the risk. The depreciation is set to 6% which is a normal rate in Denmark. All these precentages are mangement decisions to take in order for the prices to follow the sites price policies.

In order to find total hardware cost with the current power price the following formula is used:

$$HWCOST = INVEST + (POU + COOL) * INVESTPERIOD \tag{5.1}$$

where the parameters come from 5.2.1.

$$TOTALCOST = HWCOST * (1 + DEPR)^2 * (1 + RISK)^2 * (1 + MARKUP)^2 \tag{5.2}$$

The daily price is found by:

$$DAYCOST = \frac{TOTALCOST}{1000} \tag{5.3}$$

This gives the daily price for day j with the power price PO_j. The power price is obtained from the power supply company or in Denmark it can be located on the Nordic Power Trade Market[1]. Since the power price fluctuates almost daily the resource sites have to calculate a new price every day.

[1] www.nordipool.no

5.2.2 Trust in the Price Model

In order to connect price and trust, I will let the trust score from the reputation system be mapped to the risk parameter in the hardware cost. This way any change in the trust score will have immediate effect on the price according to the model 5.2 on the previous page. The risk parameter is set to go from 0 % to 20%, which is chosen since it is not to high a percentage nor to low percentage. If you are highly trusted the risk is almost 0 and if you not trusted your risk is 20. It can be selected differently according to the sites price policies. Some sites will not conduct business with somebody who is not trusted, whereas other sites will take a higher percentage for doing transaktion with one with a low trust score.

In the scenario where the risk fluctuates from 0% and 20 % and the trust score is taken from [Brinkløv and Sharp, 2006], which gives a range from [-1;1] the following mapping is used:

$$RISK = (1 - trustscore) * 10 \tag{5.4}$$

If you look at the formula 5.4 any change in the trust score is amplified by a decade. This means that small changes in trust scores will have a larger impact on the price.

5.2.3 Time of day in the Price Model

In order to segment after time, there has to be a standard node. HP Proliant DL380 G4 rack server [2] is chosen as many Danish IT-firms uses this for hosting. In this standard node there is 2 CPU's, 4GB memory and 300GB storage.

The jobs on in Grid system contains information about how much CPU time is required, how many CPU's are to be used, memory usage and storage usage. These have to be combined with the daily price. This is done by weighting them equally:

$$x * 2CPU + y * 4GB\ RAM + z * 300GB\ HDD = 1$$
$$\Updownarrow$$
$$x * CPU + y * 2GB\ RAM + z * 150GB\ HDD = 1$$

[2] Specs. are available at http://h18004.www1.hp.com/products/servers/proliantdl380/index.html

Since they all carry the same weight:

$$x * CPU = \frac{1}{3}$$
$$y * 2GB\ RAM = \frac{1}{3}$$
$$z * 150GB\ HDD = \frac{1}{3}$$

Now the parameter α is introduced as a scalar dependent on the job information on hardware resources:

$$\alpha = (\frac{1}{3} * x\ CPU + \frac{1}{6} * y\ RAM + \frac{1}{450} * z\ STORAGE) \tag{5.5}$$

From this the hourly rate is deducted as:

$$HourlyRate = \alpha * DAYCOST \tag{5.6}$$

The hourly rate only holds for one day or until the power price changes. If a job request is longer than 8 hours the user has to buy an option for the job and the price for that is calculated using Black-Scholes from 4.10 on page 30. The price will rise since you have to pay for the option and for the usage of the resource.

5.2.4 Discount Pricing

To attract jobs to sites, which otherwise not receive many jobs, the site can use discounting as a strategy. It makes sense since the units have low variable cost and since the market is homogeneous where the consumers only choice of comparison between sites is the to use prices. If the site have a lower price than the rest, then this site will raise its probability of receiving jobs. If the resource sites have a discount price and normal price for a capacity C of goods, then according to Littlewood's rule 4.3.4 on page 26 the optimum protection level is

$$F_1(y_1^*) = 1 - \frac{p2}{p1} \tag{5.7}$$

Since it is assumed the demand follows a normal distribution this equals to:

$$y_1^* = \mu + z\sigma\ where\ z = \Phi^{-1}(1 - p2/p1) \tag{5.8}$$

Since discount often is taken from the end price, the discount price is $Dayprice - (discountpercentage * dayprice)$, which leads to the following prices:

- $p2 = (1 - discountpercentage) * dayprice$

- $p1 = dayprice$

Since z in the protection limit comes from the fraction of $p2/p1$, this leads to:

$$\begin{aligned} z &= \Phi^{-1}(1 - \frac{(1 - discountpercentage) * dayprice}{dayprice}) \\ \Updownarrow & \\ z &= \Phi^{-1}(discountpercentage) \end{aligned} \tag{5.9}$$

Then for the demand D with mean μ and standard deviation σ and the discount price is dependent of the dayprice in 5.3 on page 35 the protection limit is:

$$y_1^* = \mu + \Phi^{-1}(discountpercentage)\sigma \tag{5.10}$$

How to set C it's up to the site managers, but a good indicator is the Demand Factor, which is the ratio of the total demand to capacity. If this is less than one, the capacity is a not sufficient for very long.

5.3 Process Flows for Job Submission in an Economy Grid

Another part of the model formulation of a just price on in Grid system is how the process flow should be. GRACE 2.2 on page 6 gives a good indication of how this could be done. There are the point of view from the consumer and the point of view from the resource site. What is new it's the trust evaluation compared to the methodology in GRACE. In this system there has be an entity, which has absolute trust from both the consumer and the resource site, in order to introduce money as a concept. The´entity will be referred to as a bank, but it is the whole monetary system it refers to.

Figure 5.3 on the next page shows the process the user has go through in order to submit a job. The user will accept the offer from the site if the user thinks it it the best offer at the time. In the end of the process the trust between the site and user is evaluated and money is transferred from the user to the site.

In figure 5.4 on page 41 the site is asked to give a price for a job. But before the site calculates the price the user has to pass three checks:

1. Credit check. There is no need to use time on a user who does not have any money.

2. Is the current users trust score above the sites threshold? In other words does the site wish to conduct business with a user with the current trust.

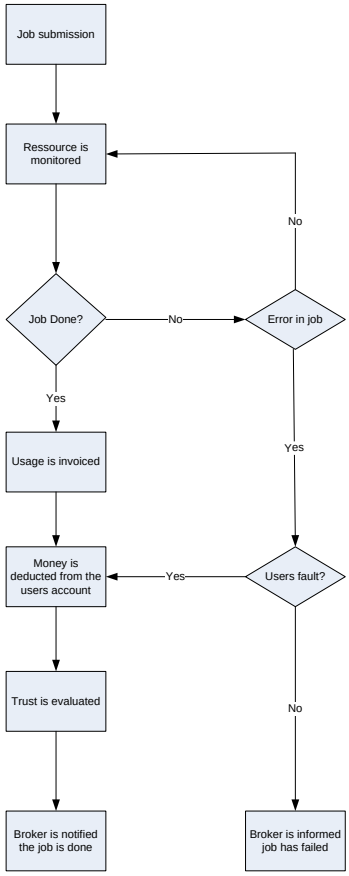

Figure 5.2: The process of a job done at the site.

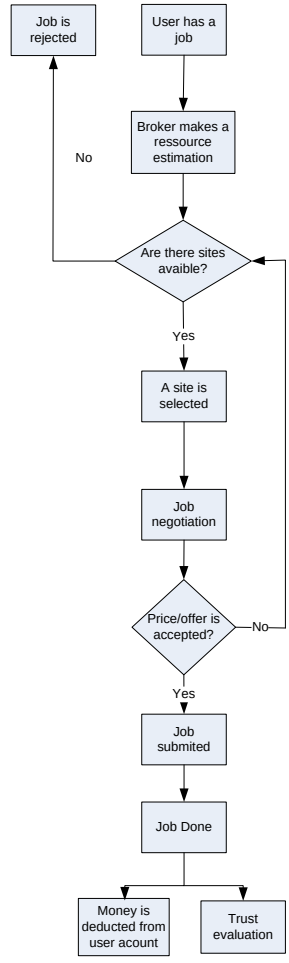

Figure 5.3: The process of a users job submission.

3. Does the site have enough available resources to meet the demands in the user's job? This is build on the assumption that the sites are fair and will not take jobs when they don't have enough resources.

If all three checks add up, the site is willing to do the job. In the last figure 5.2 on the previous page the site is deducting the money for a job done. If there is an error and the job is not completed, the site will decide whether it is a technical mistake on the site or a user mistake. Dependent on the outcome the following would happen: If it is the user's mistake the money will be deducted, and if it is the site's mistake the job would re-run on the site.

5.4 Summary

In this chapter I derived my price model and I derived how the user and sites interact with each other. In the section Pricing scheme for a Grid system I derive my calculation model for prices and I show how discount prices can be handled. In the process flows I show the different scenarios to be handled.

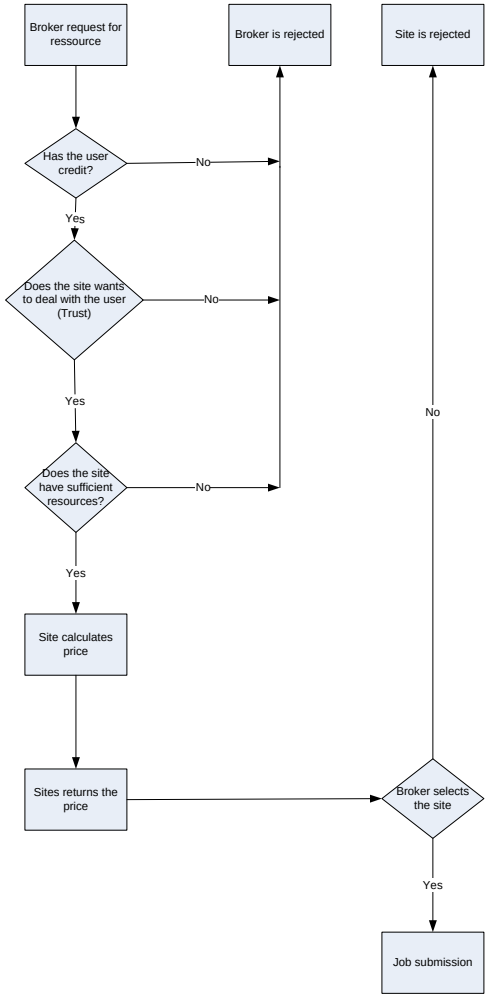

Figure 5.4: The process of a users job submisson to a site from the sites point of view.

Chapter 6

Simulation

6.1 Overview

In this section the presumptions and delimitations for the simulation is defined. The market behavior is defined as well as the user and site behavior. In the two last sections two test cases are formulated.

6.2 Analysis

6.2.1 Presumptions and Delimitations

Since the simulation has to simulate a economic computer grid which is highly complex, I have made certain assumptions and delimitations for the simulation.

The following items apply:

- The market is a competitive market, since there is little entry cost to join

the market. In theory everybody can offer their computational resources to a Grid system.

- Since the goods are computational power and specific CPU's, it is also a homogeneous market.
- The depreciation model is linear and not exponential. I selected this since it is easier to handle.
- In the simulation the sites are not allowed to overbook. This ensures that all jobs will be executed on some site.
- In the simulation only CPU time requests and request for CPU's is investigated. The cross elasticity between CPU, memory and storage is not simulated.
- The user cannot cancel a job once it is submitted to the sites.
- The general rule of optimization is price and not time.
- The I/O operations and the network performance is assumed to be perfect in this simulation. In the real world this may be a major parameter.
- The price elasticity is perfectly inelastic, so any change price will not have an effect on the quantity
- The cross elasticity for CPU, memory and storage is defined by 5.5 on page 37.

6.2.2 Market Behavior

To determine the market behavior, the Nordugrid was investigated for demand and supply. The log files from January 2005 to November 2005 was obtained and from that the demand for job requests and the distribution of the jobs arrival was found.

The log files from the Nordugrid contained 187724 successful jobs, but there was no logging of memory used, CPU time used and how many CPU's used. In order to get this the end time of a job was deducted from the start time which I use a CPU time. This is not the exactly CPU time, but an approximation to it. I believe the approximation is appropriate.

The data showed that demand arrived with a Poisson distribution whereas the request for CPU time followed a normal distribution.

In order to see money flow in the system and the market an bank is introduced. The banks purpose is to check creditability of the users, to keep track of accounts with money and to show transaction history. It does not have any influence on the market with the traditional financial instruments such as to raise the short-termed interest rate or the long-termed interest rate.

6.2.3 Derivation of Demand

To derive the demand the data set from the Nordugrid was analyzed. The data indicated that the arrivals between job requests followed Poisson distribution. The arrivals was derived by subtracting one jobs start time from the next job start time. The result can be seen in figure 6.1

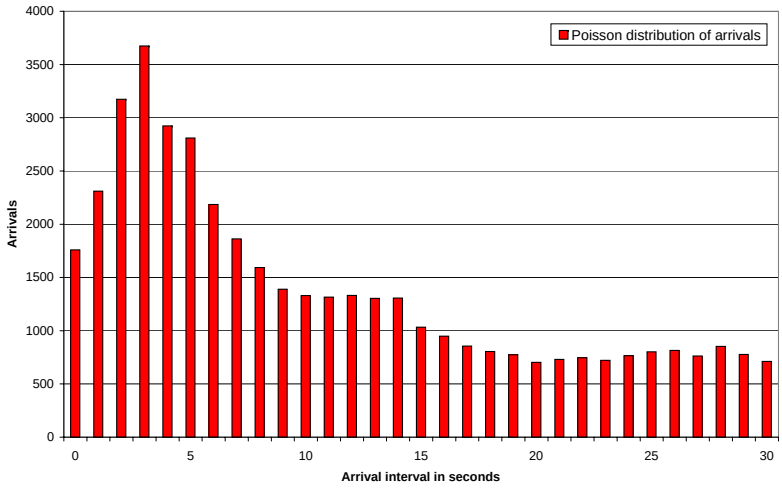

Figure 6.1: CPU request arrival interval in seconds taken from the NorduGrid

Since Poisson distribution is used to model the number of events occurring within a given time interval this model was examined to see if it match the data. After deriving the mean from the sample from 0 to 30 seconds, where the mean is 10,73 seconds this was used as λ in the Poisson distribution. The derived Poisson distribution shown in equation 6.1 where X is the arrival time and its graph is shown in figure 6.2 on page 47.

$$p(X, \lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!} \; for \; X = 0, 1, 2 \ldots, 30 \tag{6.1}$$

In order for it to be a Poisson distribution the samples have to satisfies 3 criteria [J.D. Petrucelli and M.Chen, 1999]. To state those criteria it is assumed that λ equals the mean number of counts in the entire interval. Next it is assumed that the entire interval equals M units. The last assumption is that the entire interval can be partitioned into subunits of size δ. The criteria is then:

1. The counts in disjoint units are independent.
2. $\frac{1}{\delta}$P(There is exactly 1 count in U(δ)) $\rightarrow \frac{\lambda}{M}$ as $\delta \rightarrow 0$
3. $\frac{1}{\delta}$P(There is more 1 count in U(δ)) $\rightarrow 0$ as $\delta \rightarrow 0$

In the data set the entire time interval is 30 seconds with a mean of 1485 counts and a mean number of counts per second to be $0,696^{-3}$. It is assumed that the 30 second interval can be partitioned into less and less units. Then the criteria is:

1. The counts in disjoint units are independent.
2. $\frac{1}{\delta}$P(There is exactly 1 count in U(δ)) $\rightarrow \frac{1485}{30} = 0,696^{-3}$ as $\delta \rightarrow 0$
3. $\frac{1}{\delta}$P(There is more 1 count in U(δ)) $\rightarrow 0$ as $\delta \rightarrow 0$

This shows that the Poisson distribution can be used to model the arrival time in the simulation.

6.2.4 Request for CPU time and CPU's

When I displayed this on a graph it showed that the request from 0 to 5 minutes followed a normal distribution which can be seen in figure 6.3 on page 48.

The statistical analysis showed that the requests follow a normal distribution with $\mu = 156,75$ seconds and a variance of $\sigma^2 = 3198,633$ and is shown in figure 6.4 on page 48.

Since there was no data on CPU requirements, I had to choose this parameter. I have chosen it to resemble the distribution for CPU time requests. However I have selected a lower mean to be $\frac{1}{10}$ of the mean of CPU time and setting a variance of 25. This can be seen in figure 6.5 on page 48.

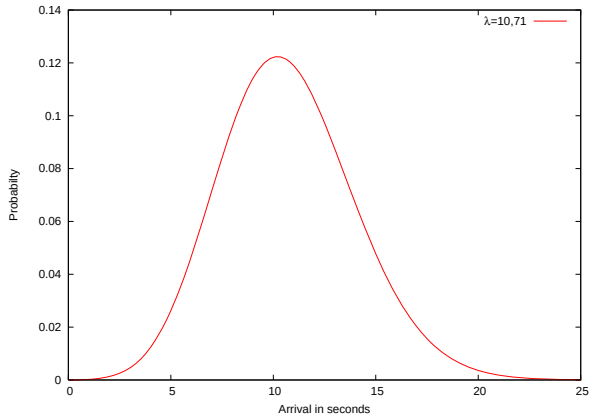

Figure 6.2: Poisson distribution with $\lambda = 10,71$ used to model the users requests for CPU time

6.2.5 User behavior

The users are assumed to be myopic customers [Talluri and Ryzin, 2005], which means they will buy the goods as soon as the offered price is below their willingness to pay. Their willingness is defined to be if they have money on their account in the bank. However the user still checks the price with other sites in order to get the cheapest price. If the user encounters a site with a lower price, it will select this site instead as long as the site in question have capacity to handle the job. By definition the cheapest site for the user will be the site with least occupied CPU's and where the trust score between the user and the site is best and where the sites is trustworthy. The users do not have any preference for if which site is most trusted just along as the site is trusted. Summing up the cardinal rule for the users behavior is:

1. Buy at the cheapest price from a resource that is trustworthy.

6.2.6 Site behavior

The cardinal rule for the site is:

1. Capacity should be allocated to a request if and only if its revenue is

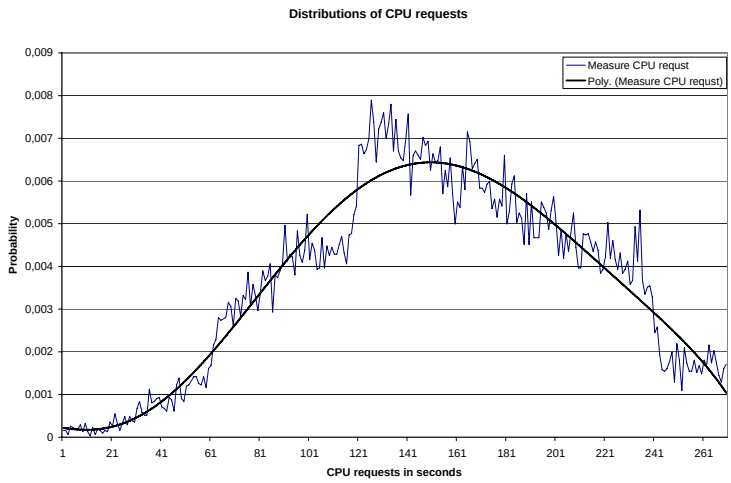

Figure 6.3: CPU request from the NorduGrid.

greater than the capacity cost

Moreover a site is interested in having the highest profit and not particularly the highest amount of jobs. In order for sites to attract jobs they can have a discount price for a certain amount of the capacity. The amount is governed by the site. How to change between discount price and normal price has to be

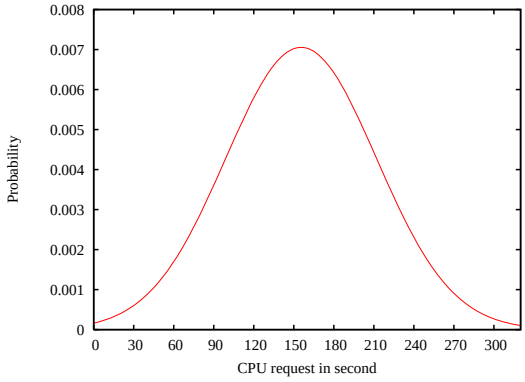

Figure 6.4: Request for CPU time with $\mu = 156,75$ and $\sigma = 56.56$.

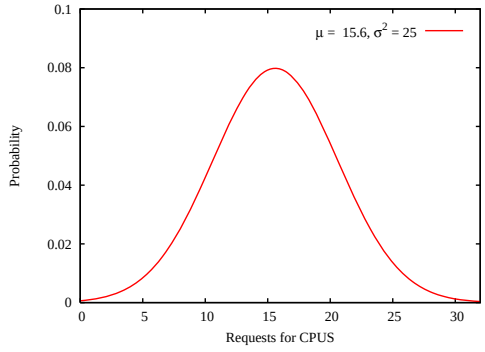

Figure 6.5: Requests for CPU with $\mu = 15.6$ and $\sigma = 5$.

done dynamically.

Furthermore the site calculates the price to a user from the users trust score. If a user's trust score is below a threshold value determined by the site, the site will raise the risk parameter to a high percentage. That way business will still be conducted but a high price.

6.2.7 Power Market Behavior

The behavior of the power market is modeled after the Danish Power Market. The power prices fluctuate according to the demand and supply on the power market, where a Grid system is located. I obtained the monthly power price from 2003-2005 for the usage of 4000kWh in Denmark[1] which has to be piped into the trust simulator. A normal server uses approximately 3500 kWh a year, the figures (see figure 6.6) from Danish Energi can be used.

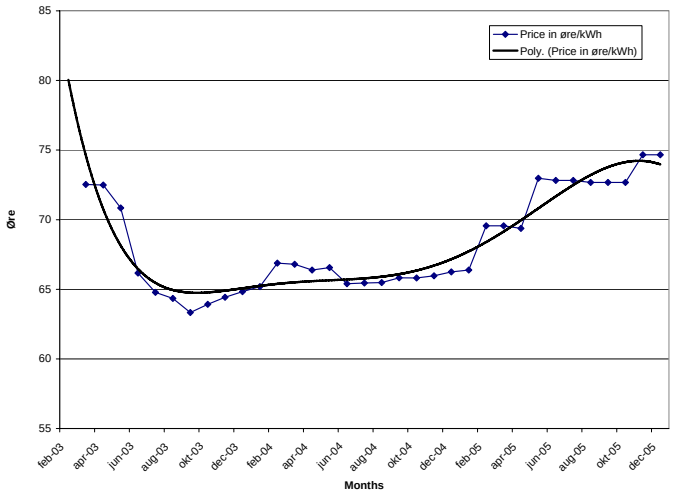

Figure 6.6: The price fluctuation of kWh on the danish power market.

[1] Available at http://www.danskenergi.dk/webtech/statistik.nsf/fWEB?ReadForm&Load=KJEN-5B8F7R

6.3 Design

6.3.1 The Trust Simulator

In order to run the simulations a framework is provided by Michael Brinkløv [Brinkløv, 2005], which is the base of my simulations. His system is a discrete event simulator designed to examine the progression of trust and managing this in a grid system.

The requirements for the Trust simulator is to keep track and provide trust scores for sites and users. Furthermore the system has to handle beta reputation.

6.3.2 Elements extending the Trust Simulator

In order to use the Trust simulator for economic simulation the following classes has to be added:

- Pricechange event. This is used when power prices changes. See the appendix D.1 on page 99 for source code.
- Salary event. This is used to simulate that the user receives a monthly salary or an annual budget. See the appendix D.2 on page 100 for source code
- Creditrejected event. Used to check if the user has moneny or not. See the appendix D.3 on page 102 for source code
- bank. Controls all money flow. See the appendix D.4 on page 103 for source code

The following elements has to modified:

- Modifications to resource user. This is particularly how to select a site for the users job. See the appendix D.5 on page 107 for source code
- Modifications to resource site. The resource site gets some more methods to calculate prices. See the appendix D.6 on page 116 for source code

- Modifications to configuration class. Since economy is introduced, some new data structures are made. See the appendix D.7 on page 127 for source code

In the following sections the above mentioned changes are elaborated.

6.3.3 Pricechange event

An event is something that happens at a separate time in the simulator. The event is made from the input file "price.txt" which is contains a list of power prices. It also contains the time for the power price to change. When the event is triggered in the simulation is runs through the array of sites and ask them to update their prices with the method *update_price(price)* where the argument *price* comes from "price.txt" . The site update their internal power price, with *price*.

6.3.4 Salary event

This event is to simulate the user receives money. The reason for this is that firm, universities have a budget for each year and the receive funding for a period of years. When the event is triggered is ask the Bank entity in the simulation to update all the users accounts with amount specified in the "salary.txt". The text file contains a point in time where an amount is added.

6.3.5 Creditrejected event

This event is used for tracking how many jobs that have been rejected by the site, due to the user have no money. It also shows if the rejection is due to the site not having capacity to meet the users job request.

6.3.6 Bank

The bank should keep track of sites and users account. It is a single object which is created when the simulation starts and is created by the configuration object. The configuration is inherited by all the users and sites. In order to

manipulate the bank, there are dummy function in the configuration file, that mappes the functions in the bank.

Data Elements The data elements in the bank are:

- A list of accounts

Every resource needs to have an account in order to track the money flow.

- A list of jobs awarded

Since every job interacts with the bank, the bank can count the amount of jobs given to the sites. This makes sense when testing if the site with the most jobs also is the site with the highest revenue.

- A start amount for accounts.

Every user account starts of with an initial amount, while no sites have money on their account. The idea is that site accounts grows as the user accounts decreases.

Methods The following methods are needed for the bank:

- To put a salary on a user account

Since the system operates with salary at a given time to the user, the account need to be modified with the amount.

- Transfer money from the user account to the site account.

When the job is done, the site needs to be paid. The site uses this function in the bank where the price deducted is the price informed to the user.

- To setup the bank

When the bank is initialized all user account are created with the amount specified in the bank section of the configuration file 6.3.9 on page 55 and the sites' accounts are created with the amount of zero.

- To check the credibility of the user

This method looks at the user account. If the account exists and there is more than zero money in the account it will return true. Else the user has no money and it returns false.

6.3.7 Modifications to Resource User

In the resourceuser class the major changes were in the method "submit_job", where the user now has to select the cheapest site. Since the only method that is changed is "submit_job" I will only elaborate this.

The first two changes is how the CPU time and CPU's are requested. Now the request for CPU time and CPU requests are taken from a random normal distribution with a μ and σ defined in the configuration file. The random generator is taken from GNU Scientific Library [2].

The next major change is how the user selects the site.

The users will ask all sites for a price where the sites will return the price based on the current trust score between the site and user. The user will then take the site with the cheapest price. In pseudo code that can be seen in 6.1 on page 58. The behavior for the user if it encounters a site, which behaves badly, is not award the site with a job. The extreme situation is all sites are ill behaving, which will lead to the user refusing to send any jobs to any sites.

6.3.8 Modifications to Resource Site

In the existing resource site class, the concept of cost had to be introduced. In the configuration file 6.3.9 on page 55 the Economy section specifies the markup, depreciation, discount, protection level and the hardware cost. In order for the sites to know this, they all have protected variables which is set in

[2]See the documentation at http://www.gnu.org/software/gsl/

the constructor, when the site object is created. Another major change in the constructor that calculates the protection level for the capacity as mentioned in 4.3.4 on page 26.

methods The following functions needs to be added to the resourcesite:

- To calculate the protection level for the given capacity

When discount is specified in the configuration file for the site, the protection level has to be calculated. The method calculate the z of the formula in 4.7 on page 27 where it returns the inverse cumulative normal distribution for the discount rate and returns the protection level. This is used to find the booking level, which is the $capacity - protectionlevel$.

- To answer if the site is fully booked.

When a user asks for a site, he first asks if the site is full. This method returns true if the sites private variable "current available CPU's" deducted the users CPU request is less than zero.

- To update the power price, when the power price changes.

When the pricechangeevent happens this method is called. It takes the price specified in the events and coping it to the site's own private power variable.

- To calculate the price for a job

This function calculates the users price for a job. It takes the user and his request for CPU's. First of it is checked if the user and site have ever conducted business together.

If this is not the case, the site assumes that the user has a neutral trust, which is the trust value of 0. If the user and the site have conducted the business, the trust score is calculated and used. The site then has to check if it finds the user trustworthy according to the sites threshold for trust. If the user is not trustworthy the site will still conduct business, but now the site has to guard itself from a economical loss. It should handle this by raising the risk parameter

to a high percentage such as 80%. However if the site trust the user the the risk is calculated as specified in 5.4 on page 36.

The total cost has to be calculated and an internal price, which is the production cost at the given time. When the internal price is known the maximum discount has to be calculated in order to make sure that no discount given is below the maximum. It it meant to be a safety valve, which will, if triggered, raise the price to 10 times the normal level. This way the user will not select this site, as it is to expensive. And this way the site will not sell with loss.
When discounting is used, there has to be a check that the site does not sell above the booking limit. The psedo code can be seen in tabel 6.2 on page 59.

- To calculate the price for a job during the call "add_job"

In the method "add_job" the private variable "cpuasked" is set from the Job object. The method creates an entity object (see figure 6.7 on the following page) if it has not been created. Next it calculates the price for that specific user with the site to user trust score as an argument. Then in the end the method checks that the site will not be overbooked if it accepts the job. If not the counter of available CPU's is decreased and the counter of occupied CPU's is increased.

- To write out the price in method "job_done" and deduct money from the user in the bank

When the job is done at the site, the price history is updated with a time stamp and the price, the user has used. This prices is also used to deduct the correct amount from the user's account to the site's account.

6.3.9 Modifications to Configuration Class

In configuration file there has to be a new economy data structures and there has to be methods to handle the communication between the sites/user and the bank.

6.4 Implementation

The Trust Simulator is implemented in C++ and my modifications are also made in C++ and can be seen in appendix D on page 99. The event simulator follows the flow chart on figure 6.7:

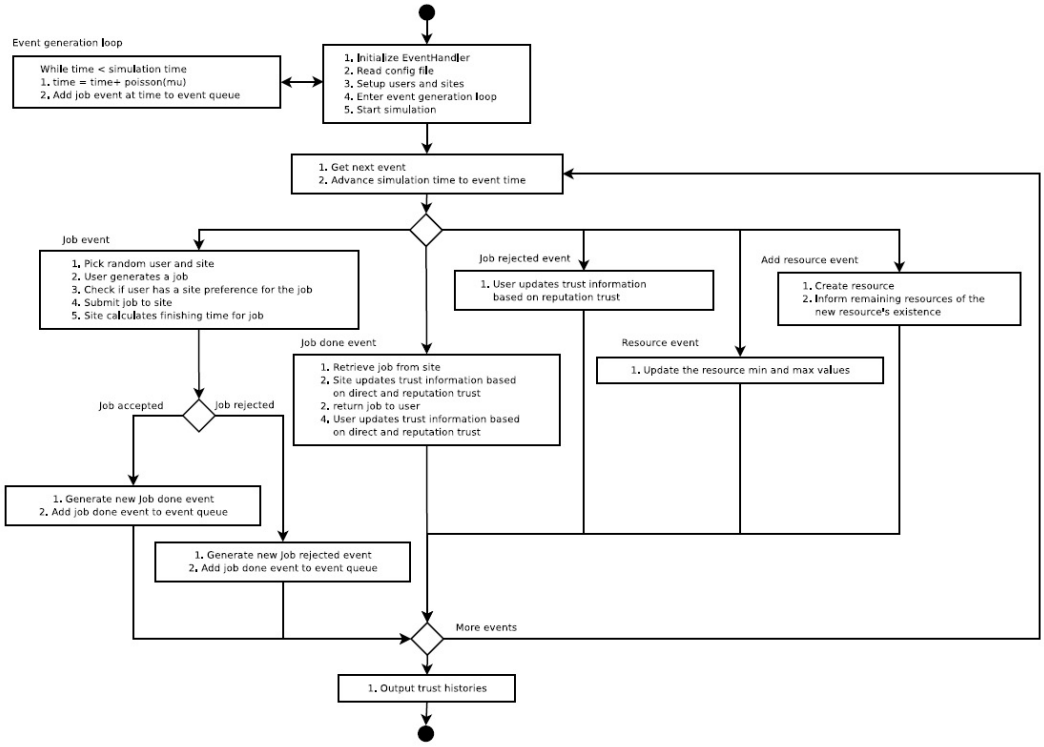

Figure 6.7: Trust simulator flow chart

This system is modified to simulate grid economy based on trust.

Since it is a discrete system it models a system by a representation of state variables that changes instantaneously at separate points in time. The simulator is designed as an object-oriented application, so all objects are inherited. In order for me to extend it, I have to add some new objects and modify existing objects. Now the sites have to calculate a price for each job and the users have to make a conscious decision about which site to use. The system should also handle external input from the world in terms of changing power prices. To

simulate an economy there has be a bank, which handles monetary issues. The main properties of the trust simulator is not changed in the sense that it will still be a discrete system.

I added a new structure to the configuration files. It is called economy and contains all the relevant parameters for each site. Here is the markup rate, the depreciation, the hardware cost etc defined in.

The rest of the changes has been implemented according to the specifications of section 6.3.2 on page 50 and the source codes can be seen in D on page 99.

6.5 Summary

In this chapter I have shown what it will require to extend the Trust simulator to handle economy according the Model formulation chapter. I have defined the behavior of the players in the simulation and what component that needed to be made in order to extend the Trust simulator. In the end I show how it is implemented.

Listing 6.1: Pseudo code for how a user selects a site to submit a job

```
1
2           The site id given randomly assumes to be the cheapest.
3
4           Site id is copied to a temporary site  rs
5           Price is calculated for site id and put in a temporary
                price
6
7           for{siteiterator=start of site array; siteiterator!=end of
                array; siteiterator++}
8           {
9                   Select site from siteiterator.
10
11                  if(site have CPU capacity)
12                          if(current sites price is cheaper than
                                temporay price)
13                                  check the users trust to the site
14
15                          If(first time talking to site)
16                          assume site has a trust value of a stranger
17
18                          else(not first time communicating)
19                          temporary trust = the current trust between
                                 user and site
20
21                          if(temporary trust is above the current
                                trust threshold for the user=
22                          this is now the cheapest site
23                          site id=temporary id
24                          price=temporary price
25
26          }
27
28          Check to see if the for loop has found a cheaper site than
                the randomly site?
29          if(random site and user have communicated before)
30          check trust against threshold
31
32          site found.
```

Listing 6.2: Pseudo code for how a site calculates the price

```
function calculatesite price(user, request for CPU's)
{
if(first time communicating with site)
trust=the trust for stranger
else
trust=trust value for user


risk=trust mapped to risk

if(site trust to the user is above the current treshold)
risk=80

calculate hardware cost
calculate total cost
calculate the internal cost for day
calculate the maximum discount

        if(the site has a discount rate and this is below maximum
            discount)
        {
                if(the current occupied CPU's + the amount
                    requested by the user < the booking limit)
                {
                user price=day price-discount
                }
                else
                if(the discount is higher than maximum discount)
                user price=10 * user price

        }

Return the user price

}
```

Listing 6.3: Economy struct from Configuration.cpp

```
//Parameters to calculate prices for the sites
struct economy{
        unsigned int markup;
        unsigned int depreciation;
        unsigned int discount;
        unsigned int hardwarecost;
        double protectionlevel;
        unsigned int sitecpus;
};

//map of parameters and sites.
typedef map<string,economy> ecmap;
```

Chapter 7

Simulation Experiments

7.1 Overview

In the first two sections I define my test cases. In each of these sections the different setups for testing the simulator is elaborated. In the last two section the results of all the tests are presented and elaborated.

7.2 Test case 1: Price is Dependent on Trust

The main idea in this part of the simulation is to show the correlation between the users trust score and the price the site will offer to the user. In other words there is a mapping between $\Delta trust$ and $\Delta price$. The test was done with 10 repeats with different seeds. Seed going from 100 and incremented by 10 every time. The number of sites and users is never higher than 100 for each.

7.2.1 Price Variation due to Different Price Parameters

7.2.2 Test Case 1: Experiment 1 - Price Change due to Trust

In this setup the power price is kept constant and all sites have the same economic parameters. All that can influence the price is the site-user trust score.

7.2.3 Test Case 1: Experiment 2 - Price Change due to Trust with Varying Power Prices

In this setup the power price follows the power consumption data and all sites have the same economic parameters. All that can influence the price is the site-user trust score.

7.2.4 Test Case 1: Experiment 3 - Different Prices due to Different Parameters on the Sites

In this setup the power price follows the data found in and all sites have the different economic parameters. The variable input is still the site-users trust score.

7.2.5 Test Case 1: Experiment 4 - How Threshold Affects Site Capacity and User's Price

In the setup both the trust from site to user and vice versa is tested. First the situation were a site is ill behaving and the users refusing to send jobs to this site. The a user will behave badly and his price will rise at the site to a very high price.

7.3 Test Case 2: Site have 2 prices and Switches Dynamically Between Them

The main objective of this simulation is for sites to dynamically switch between a discount price and normal price based on the capacity. Furthermore to check that the user selects the cheapest site available. Now the sites have to handle request for CPU time as well as CPU's. The requests follow the distribution from 6.2.4 on page 46. In the end the sites with the right capacity and usage of discount prices will have the highest earning.

7.3.1 Test case 2: Experiment 5 - User Selects Cheapest Site

In this test it is checked that users selects the cheapest site. The setup starts out with the sites all behaves good and then one site choses to behave bad. The specific site is also the cheapest site, since it is the only one that gives discount. When the user realizes this, he stops to send jobs to the specific site and instead of send it to other sites even though it is more expensive. If in the end there is no sites the user trust, the system stops, since then all sites are untrustworthy.

7.3.2 Test Case 2: Experiment 6 - Site Calculates Protectionlevel

In this setup some sites have discount prices and calculates dynamically the prices. For this test only 3 sites have discount prices and the discount percentages is 10%, 20% and 30%. The capacity is set to 300 CPU's and users can ask all sites for price. Furthermore the users have enough money in the bank to buy.

7.3.3 Test Case 2: Experiment 7 - Site Checks that Discount in not Below Cost Price

In this setup it it verified that no matter how high the discount percentage is set to, it can never be higher than the production cost.

7.3.4 Test Case 2: Experiment 8- Site Capacity is Limited

In this setup the capacity is set low, in order to generate many overbookings. This way the site with the highest revenue is the site who handles the most jobs.

7.3.5 Test Case 2: Experiment 9 - Users can Only Select Few Sites

In this setup the user cannot ask all sites, but only a certain amount of sites.

7.4 Results From the Simulation

The sections containing the results from test case 1 and test case 2 are presented.

7.5 Results from Test Case 1

The main objective in this test case is to test the price calculation in a Grid system. The price calculation is dependent on user's trust scores and energy prices. The different setups shows how the price changes when a user is ill behaving and vice versa. It is also tested that user's always selects the cheapest site to submit their jobs.

7.5.1 Results from Experiment 1

- Aim: To show price change due to trust

The experiment was carried out with 10 sites and 10 users and repeated 10 times with different random seeds to test the stability of the simulator. User zero acts nice in the beginning but changes behavior and becomes bad for a short while. Then in the end User zero act good again. There is no threshold set for the trust score and it is assumed that strangers have the trust score of 0, which is neutral. As seen on figure 7.1 and at figure 7.2 the price rises when the trust decreases.

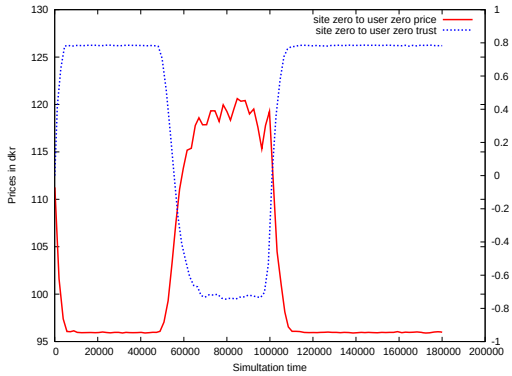

Figure 7.1: The price vs trust for user zero on site zero.

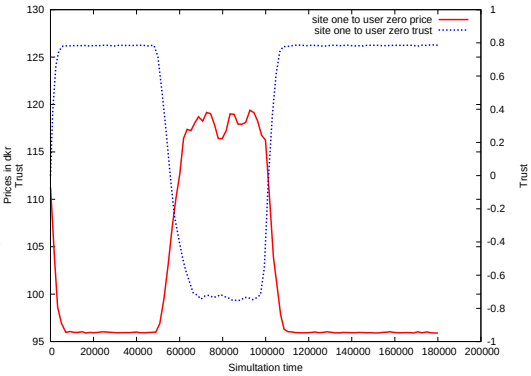

Figure 7.2: The price vs trust for user zero on site one.

The price does so at the same rate at the trust. What is more interesting is when the user is bad, there is longevity factor that makes the system forget the users ill behavior. If you look at the price when it is highest it fluctuates more than when it is at it lowest. The reason why the changes in price is more visible is mapping of trust to risk (equation 5.4 on page 36 applies) is an amplification by a factor ten. In figure 7.3 and figure 7.4 the trust of User zero is seen from

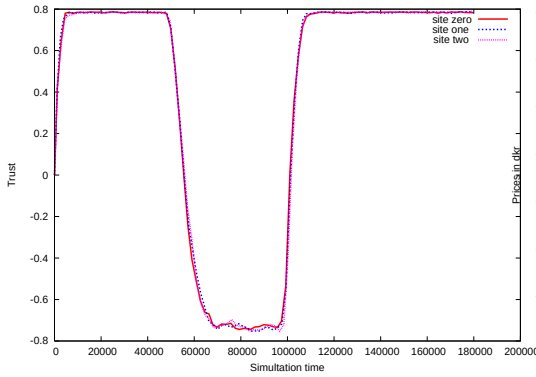

Figure 7.3: The trust for user zero on 3 different sites.

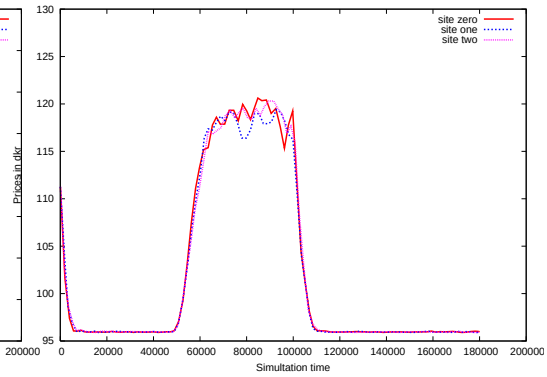

Figure 7.4: The price for user zero on 3 different sites.

3 different site and 3 different prices. Again it shows the price fluctuates more when it is high than low.

7.5.2 Results from Experiment 2

- Aim: To show price change due to trust with varying power prices.

It is the same setup as in experiment one, except now the power price fluctuates. There is no threshold set for the trust score and it is assumed that strangers have the trust score of 0, which is neutral. In the beginning User zero is ill behaving but after a while he behaves good again as seen on figure 7.5. User Two behaves good all the time. The price for User Two fall quickly and have some small peaks in the beginning as seen on figure 7.6 on the next page. This is due the the changes in power price. Another noticeable feature that power is changing is in the end of the experiment the prices keeps on lowering.

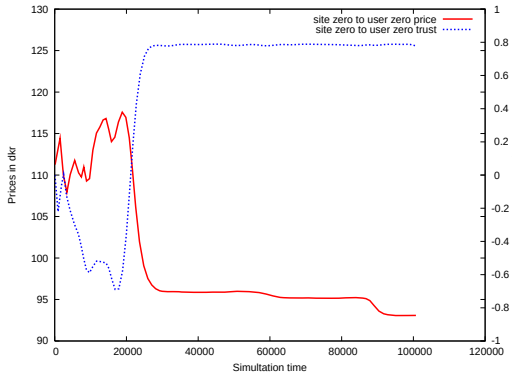

Figure 7.5: The price vs trust at different power rices.

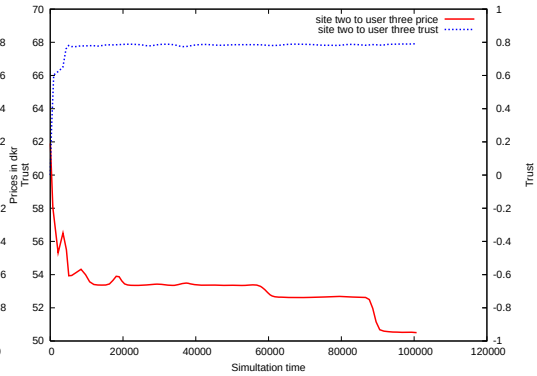

Figure 7.6: The price vs trust at different power prices

7.5.3 Results from Experiment 3

- Aim: To show different prices due to different parameters on the sites

The test setup is the same as in experiment three except that now all the parameters is different. Now Site zero, one and two all have different hardware costs. As expected the prices is very different with the lowest price at the site

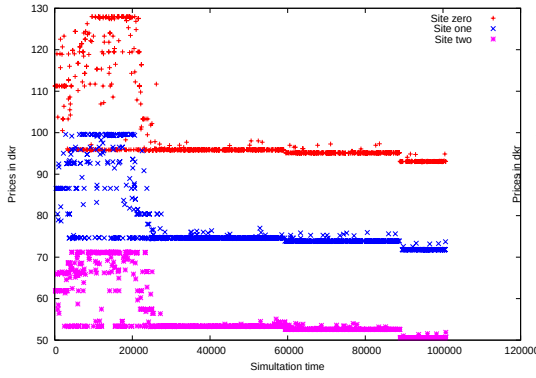

Figure 7.7: The price for different hardware prices.

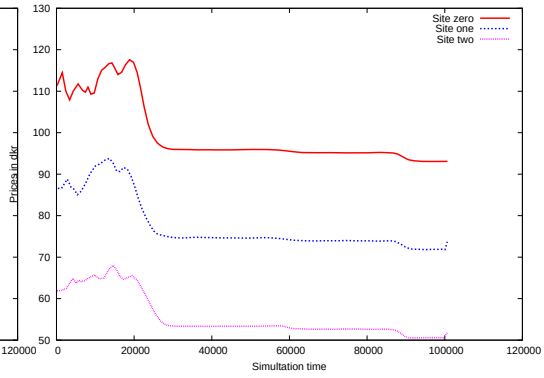

Figure 7.8: The price approximated for different hardware prices.

with the lowest hardware cost. Since the harware cost is the largest item in the equation 5.1 on page 35 and used to add the percentages in 5.2 on page 35 it makes good sense.

7.5.4 Results from Experiment 4

- Aim: To show how threshold affects site capacity and users price

In this setup the threshold was set for both users and sites. Each site is assigned 3000 CPU's and there is 15 users and 20 sites as can be seen C.3 on page 90. If the sites does not trust the user, which is the users trust score is lower than the threshold, the site will return a higher price, since the risk is set to 80 percent. It is the strategy of we will do business with you but you will pay overprice for it, since we lack trust in you. If the site ill behave, the user will now not select that site. It will simply as another site. The test starts out with site one behaving good and giving discounts. Site one it the most popular site, until the site ill behaves. Then the users will not assign jobs to the site. As seen on figure 7.9 the users stopToo send job when the site is not behaving very good. On figure 7.10 the users trust To the site is shown.

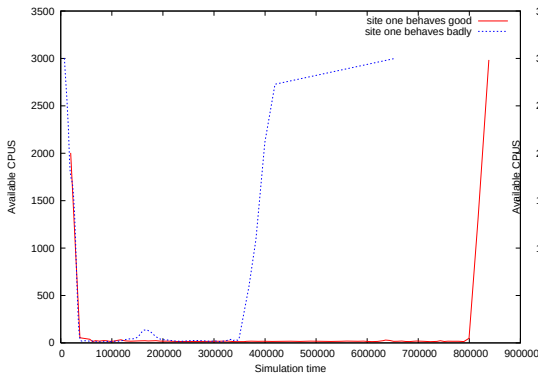

Figure 7.9: Site one behaves good at all time and Site One behaves badly.

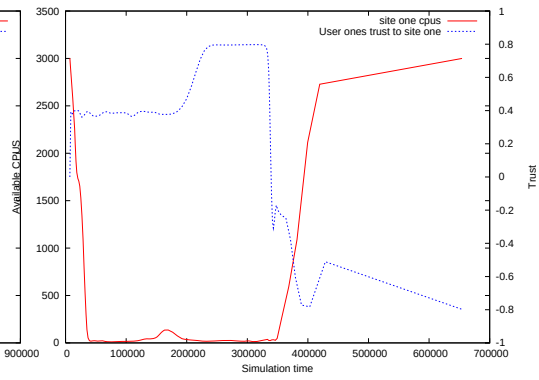

Figure 7.10: When the trust declines To the site, the site does not receive new jobs. This leads To an increase in available CPU's.

In the case where the user ill behaves and he violates the site's threshold, it can be seen that the price rises to 300 Dkr. from approximately 100 dkr. The user can still submit job as long there is money on his account in the bank, but as soon as this is empty no site will perform a job for him. The price behavior is seen on figure 7.11 on the next page.

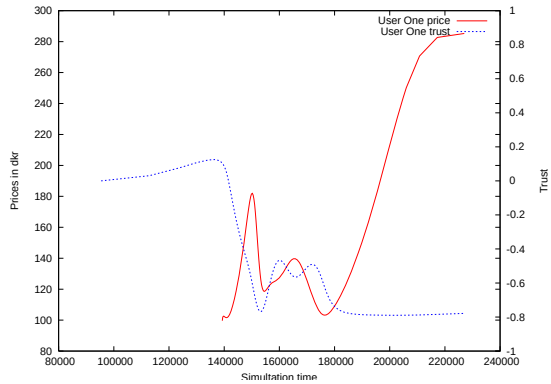

Figure 7.11: User One is not liked and the price rises.

7.6 Results from Test Case 2

The primary objective of test case two is to implement the revenue management on the system. Now all the sites have CPU's to account for and they need to track how many CPU's are used at all time. There is also a bank now who has to keep track of the money flows from user to sites accounts, keep track of credit rejections. The bank also have an extra function of counting how many sites who had to turn down a job, if the sites is overbooked. This happens only if all sites is fully booked at the same time. Furthermore the bank keep track of which sites has the most jobs and which site had the most revenue.

7.6.1 Results from Experiment 5

- Aim: To show the users selects cheapest sites.

In this setup there is 10 user and 10 sites. The sites now have a capacity of 500 CPU's each and site zero have a lower hardware cost than the other sites. As seen on figure 7.13 on the following page all sites is filling up quickly since they all within the 20000 first units of time is fully occupied. Notice the slope in the end of the simulation. Notice how site zero is more reluctant To let go, which indicates that the last jobs is given To site zero. The price difference is very clear in figure 7.13 on the next page, where the prices offered To user zero from site zero and site one. In figure 7.14 on the following page it is shown the site zero receives most jobs at any test trial.

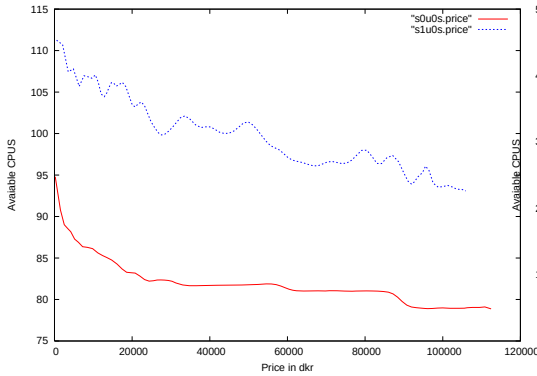

Figure 7.12: The price for user zero is cheapest at site zero.

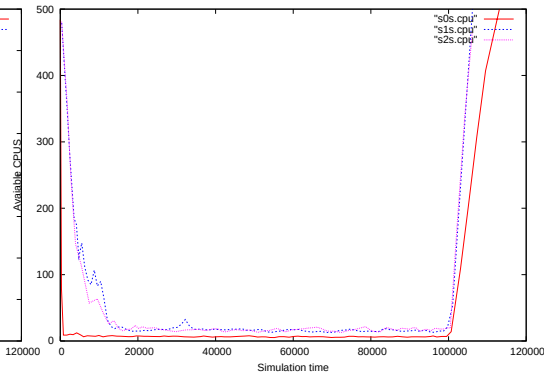

Figure 7.13: All the sites is occupied fast. How ever site zero is occupied immediately.

Figure 7.14: Out of three re-runs site zero has the most jobs.

7.6.2 Results from Experiment 6 and 7

- Aim for experiment six: To show that sites calculates protection levels.
- Aim for experiment seven: To show that sites checks that discount in not below the cost price.

The setup contained 4 out of 100 sites which has discount rates. The site was site zero, one, two and three.The special thing about three discount rate was it was set very high. In fact set so high that the site tries to sell with loss, since the price is below production cost. The discount rates ranged from 10% to 25%. When the simulation starts it calculates the protection level for the site if dis-

count rate is set. As seen in listing in 7.1 for site zero the protection level is set To 228, for site one 203, site two 193 and for site three it is set To 170.

Listing 7.1: Output from a simulation with discountsrate

```
BANK OPRETTET
site id: s0
my discount is: 10
my CPU's are is: 600
my name is s0 and my protectionlevel is: 228
my name is s0 and my bookinglimit is: 372
site id: s1
my discount is: 20
my CPU's are is: 600
my name is s1 and my protectionlevel is: 203
my name is s1 and my bookinglimit is: 397
site id: s2
my discount is: 25
my CPU's are is: 600
my name is s2 and my protectionlevel is: 193
my name is s2 and my bookinglimit is: 407
site id: s3
my discount is: 40
my CPU's are is: 600
my name is s3 and my protectionlevel is: 170
my name is s3 and my bookinglimit is: 430
site id: s4
my discount is: 0
my CPU's are is: 600
site id: s5
my discount is: 0
my CPU's are is: 600
site id: s6
my discount is: 0
my CPU's are is: 600
```

It allows the discount sites, which all have the capacity of 600 CPU's, to sell $600 - protectionlevel$ cheap. The rest has to be sold at normal price. Since the users always will select the cheapest site, all the discount sites are fast booked, which should leave many jobs to the 96 other sites. However the result show that the discount sites receives a total of 20% of all jobs, which is 1,4 million jobs in this setup. In other words the 96 other sites had to share 80% of all jobs, which on average leaves 0,83% of jobs To each one. The distribution of earnings and jobs is shown in figure 7.16 on the following page and 7.15 on the next page.

Notice how site three does not show any where. It makes sense since the price is 10 time higher than the normal price, since the discount rate is to high. No user have selected to send a job since it is to expensive. Another finding is that

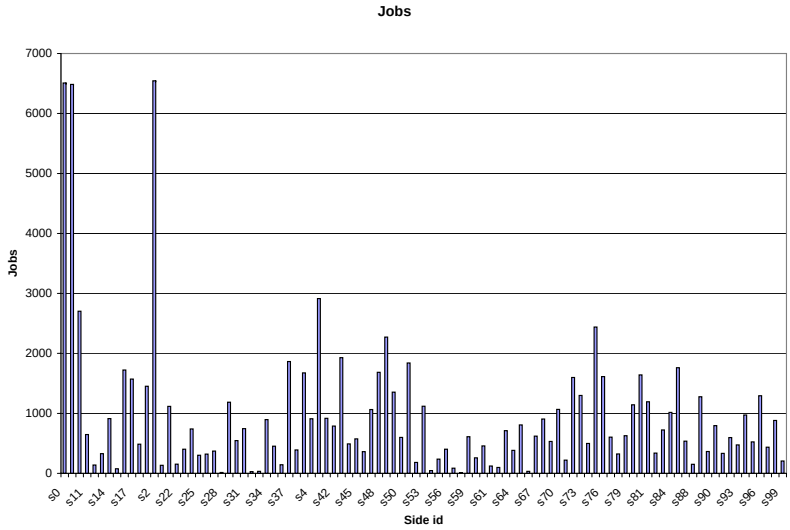

Figure 7.15: The job distribution for 1,4 million jobs.

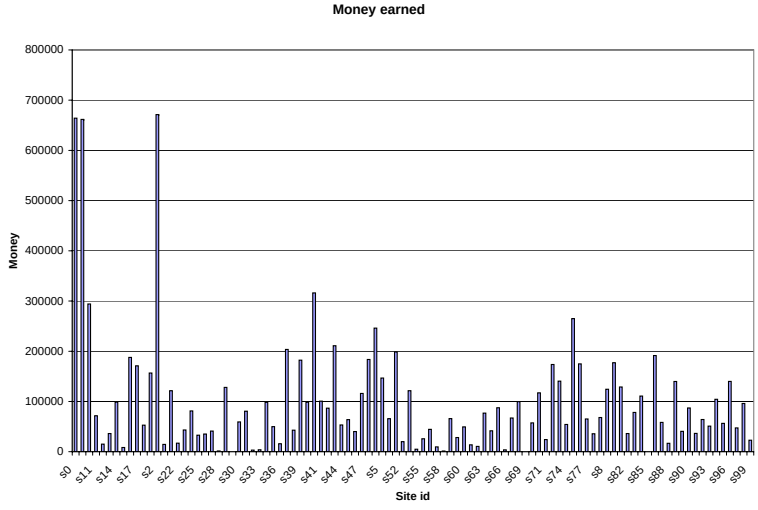

Figure 7.16: The earnings distribution for 1,4 million jobs.

the sites with most earnings also are the sites with the lowest average price for jobs as shown in figure 7.17 on the facing page.

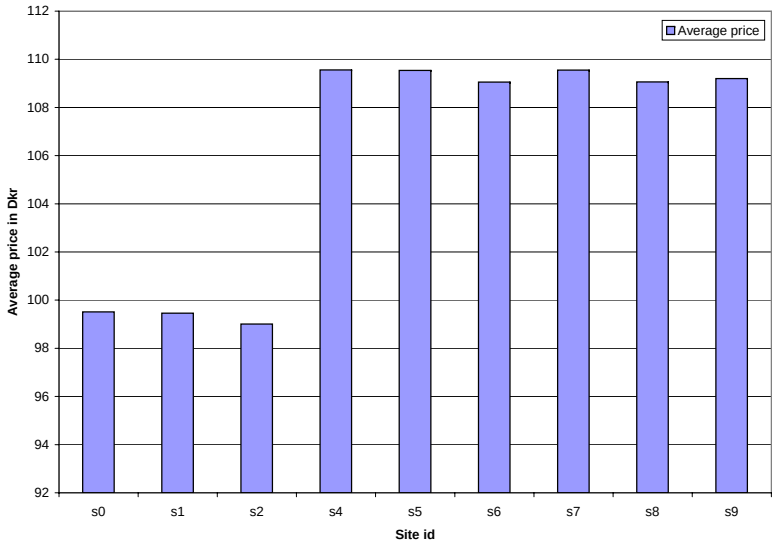

Figure 7.17: The average price for 10 first sites. Site 3 is not in since it had no jobs because of to high price.

7.6.3 Results from Experiment 8

- Aim: To show site capacity is limited.

The setup is similar To the setup in 7.6.2 on page 70 also with discount sites. To test it throughly I made three subtest with the following setup:

Test 1 20 users and 10 sites. Sites have 600 CPU's.

Test 2 20 users and 10 sites. Sites have 1200 CPU's.

Test 3 20 users and 20 sites. Sites have 600 CPU's.

The first test is to see what happens when the environment has limited capacity compared To the jobs it has To serve. In the second test I have the same amount of sites as in test one, but now the capacity has doubled. In the last test I have increased the number of sites and kept the CPU's at the same level as in test one.

The result of test one is shown in 7.18 on the following page and it shows an almost even distribution of jobs expect from site one and six. Its hard to tell

which sites is the discount sites and who is not.

The result of test two is shown in 7.19 which show a little increase in jobs per site in overall and site one and six still does not receives many jobs.

The result of test three is shown in 7.20 on the facing page which show an even distribution of jobs and each site has less jobs than in the other test.

Another way of looking on the three test is to see how many times there were overbookings. The overbooking event indicates that ***all sites*** is fully booked. This will tell more about how the system handles capacity issues. The result is shown in figure 7.21 on the next page and it shows that a system with many sites and few CPU's handles better than any other system. The test 4 is result from experiment nine which is located 7.6.4 on the facing page.

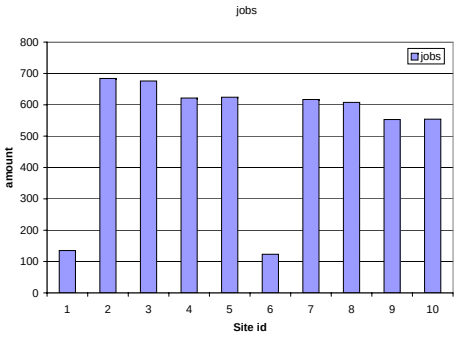

Figure 7.18: Test 1: Amount of jobs with few sites with few CPU's.

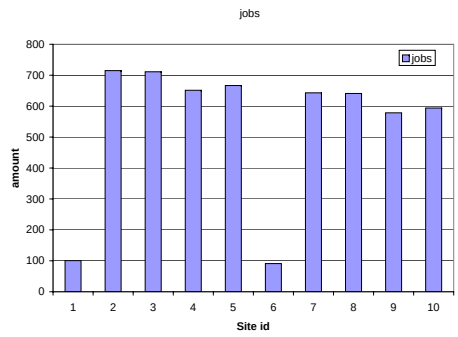

Figure 7.19: Test 2: Amount of jobs with few sites with many CPU's.

It can be noticed that the average jobs conducted per site is: The difference

Test	jobs
1	519
2	539
3	424
4	383

Table 7.1: The average jobs for the 4 test per site.

between test two and three is 120 jobs on average per site. In test two the sites had To handle 120 more jobs within the same time frame as in test three. Since the amount of CPU's follow a normal distribution with a mean of 15 CPU's, this lead To an increase in CPU's of 1800 per site. In test two the sites have

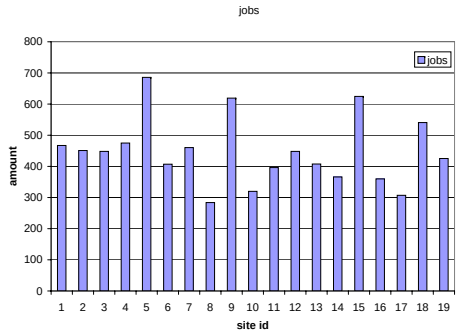

Figure 7.20: Test 3: Amount of jobs with many sites with few CPU's.

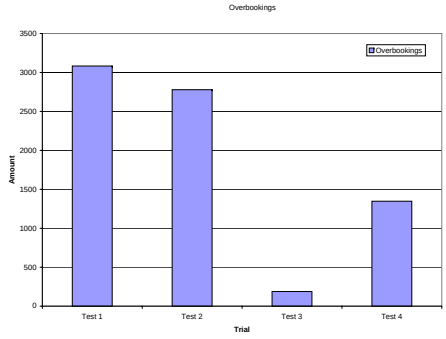

Figure 7.21: Amount of overbookings in the four subtest conducted.

600 CPU's more than the sites in test three, but the increase in work load is tripled.

7.6.4 Results from Experiment 9

- Aim: To show Users can only select few sites.

The test setup consisted of:

- 20 users and 20 sites. Sites have 600 CPU's. Different user pattern.

The different user pattern is instead of asking all the sites for a price, the user is only allowed to ask a random site which holds a position in the sites array and from that position and to the end of the sites array. The change in code was done in Resourcesite.cpp and consisted of the change in 7.2.

Listing 7.2: Now the user have To select from the random id to end of site array.

```
1 for(rIter=rSites.find(tempsite);rIter!=rSites.end();rIter++)
```

The result of the test is shown in figure 7.22 on the following page and it resembles the pattern of test one and two in the previous section.

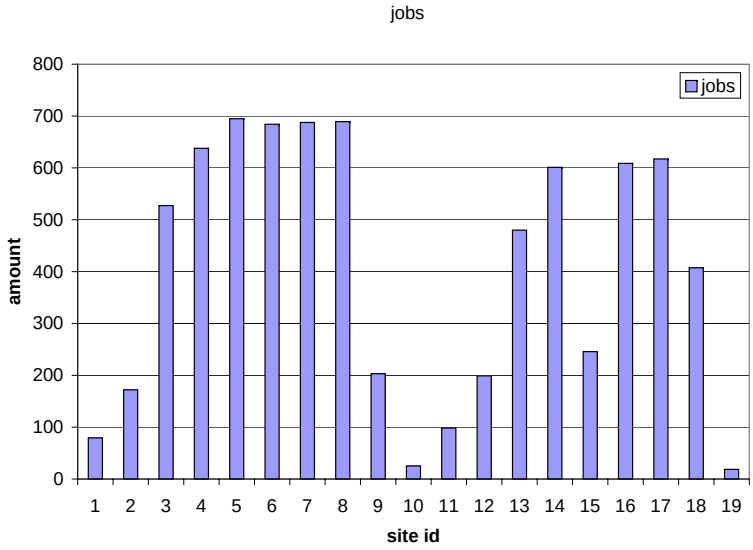

Figure 7.22: The distribution of jobs when users selection is limited.

7.7 Summary

In this chapter I start out with defining my test cases. These test cases are meant to show how prices fluctuates with the users trust and how the sites handles the capacity in a way that ensure an optimum. In the results I show how the different components interact according to specification.

Chapter 8

Discussion

8.1 Overview

In the first section of this chapter I summarize my results and discuss them. In the second section I will discuss possible future works for trust based accounting in grid systems.

8.2 Archived Results

I have archived to find a price for jobs in a Grid system. The price is dependent on many varying parameters which changes all the time due evolution of hardware. One of the parameters in the price was trust value derived from the system, which was mapped to an economic risk. This risk was set in the range of 0 to 20 percent, but it was not investigated what would happen if another mapping was used, say from range 0 to 40 percent. This mapping also assumed the trust score from the Grid system to be between -1 and 1. Another metric could have been used, but this was not analyzed.

The price was found with in relation to CPU requests, memory and storage. In

the price the CPU, memory and storage was assumed to have an equal weight of 1/3 for each. The cross relation was not simulated due to lack of relevant data and therefore it was difficult to give a qualified estimate of the use of memory and storage. The use of CPU time requests was found for the Nordugrid site, but there was no data of the amount of CPU's requested. I used a fraction of the CPU time request, since it followed normal distributed, to simulate the request of CPU's. I am confident that the request for CPU's follow the normal Distribution in a Grid system, but I do not have data to support my claim. Another issue with the relation between CPU's, memory and storage is that today storage is cheap compared to the price for CPU's and memory. The price for 2 CPU's compared to 4 GB of memory in a industrial server is almost the same and therefore it has to taken into account when calculating a just price. In my simulation this was not investigated, since I only looked at CPU's and CPU time.

In my assumptions I assume that the user always will buy. In the real world some users might want to wait to see if the prices decreases over time. The user is allowed to ask all sites in simulation, but in a Grid system with many sites this could turn out to be not such a good idea since it will take much time. Imaging the user talking to a American site which seems to be the cheapest, he will buy at that place since for the user it looks to be the best deal. However if there is network latency it not the best deal.The network latency have to be taken into consideration when pricing for the price to be more realistic.

The price was calculated according to the trust between the site and the user. If the site and user never had talked to each other before I used a default value. My strategy was that everybody is not bad and not good, but neutral. The after some transactions the site and user would have developed a trust between them, which indicates if you are behaving good or bad. Another strategy could have been that every user is guilty until otherwise proven. This could have been implemented instead in order for the site to have the highest possible prices at all time.

The usage of capacity management for resource sites proved very powerful. The sites that used discount rates where fully booked most of the time and they had the highest revenues. If the other sites, which did not have discount rates, were allowed to dynamically to change the prices to get more jobs, it would start a price war. It is not necessarily a bad thing with price war just as long as nobody can sell below their production cost, which is also called price dumping.

It was surprising to see that it was not the amount of CPU's at each site com-

pared to the amount of sites which had an impact on how many overbookings that occurred. When I looked a the average amount of jobs per site it was clear that an increase in CPU's only gave an a small deduction average job per site whereas in the case with more site there was an significant deduction. This lead to an triple increase of workload for each site, but they only had the double capacity. Also the time the CPU's are occupied plays a role and since there are more jobs, the CPU's stays occupied longer. If the arrival rate of CPU's request arrived with another Poisson distribution it could have resolved the problem, since the request arrive with a longer time interval.

The actual price I find is is realistic for one node in a Grid site. The reason for this is the model is build up with the cost for only one server, but can easily be expanded to include multiple nodes. The parameters that changes are the cost price for the hardware and amount of kWh used.

The price is only a price the variable cost and the fixed cost is not accounted for. The fixed cost of building the server room and the network infrastructure is not included. In order for this to be used in an enterprise this has be taken into account.

8.3 future work

My thesis investigates how to find a price and how to manage the resource sites. What is missing is how to handle jobs that run for many days. The Black-Scholes theorem for bond is introduced in the background sections on 4.5 on page 29 and seems to be a very interesting mechanism to calculate the price for a job that runs for more days. The idea is that the user and the site in a Grid system negotiates the time limits for the jobs. The user might want to run it within 5 days but is not sure when. The site needs to have a forecast function for the production cost over the next 5 days in order to calculate price for the bond and the price for performing the job. This way the total price to the user consist now of the price for the bond and the strike price, which is the price the user pays when the job is submitted.

Another way of using the Black-Scholes theorem is in business, where one firm can have call-options to users and by put-options of other sites. This way the site can have a fixed amount of capacity and when over capacity is needed they use put-options at other sites.

My simulation is build upon the notation of a bank. In the future there is a need for an introduction of a monetary currency on the Grid. There has be economic entities such as banks on the Grid who have high trust both from the users and from the sites. In order for the Grid to expand beyond the academic communities and into the commercial world this is a must.

Chapter 9

Conclusion

In my thesis I have investigated the different economic schemes for a Grid system and I have investigated how reputation trust can be utilized in calculating a price for a resource. Furthermore the field of revenue management with focus on capacity optimization have been investigated to control the resources at a Grid site.

I found that trust is a good way of controlling the price of resource to a user, since it awards a user with high trust with a low price compared to the price for a user with low trust.

Another discovery was that discount prices is a very powerful way of getting users to buy your resources since the resources utilizing this was fully booked. Discount pricing also lead to the highest earnings even though the average price was low compared to the sites, who did not have discount prices.

To find a just price for a Grid system it has to be just for both the user and the resource site and I believe that my model gives such a price.

Appendix A

Simulation flow chart

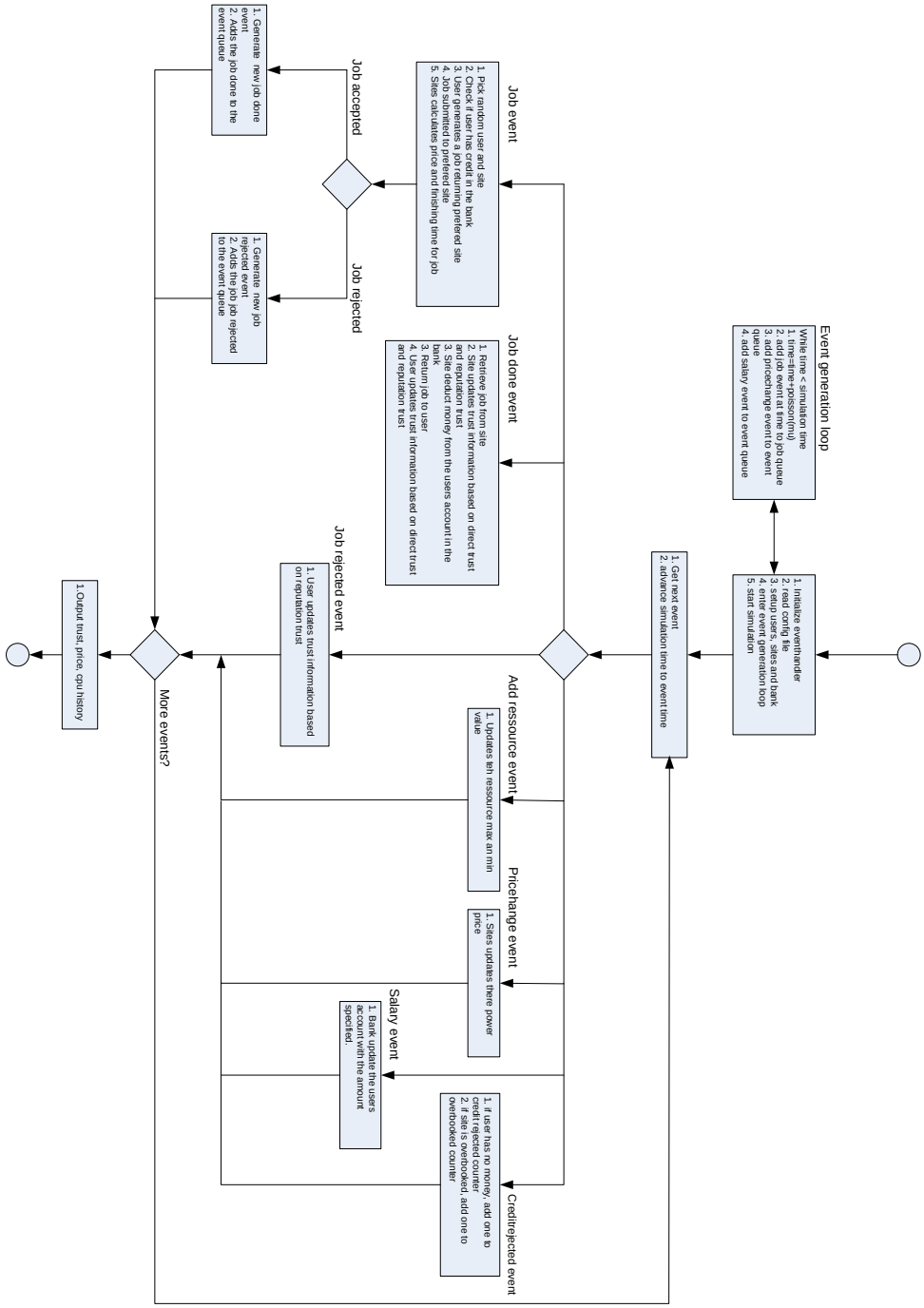

Figure A.1: The simulator flow chart

Appendix B

Contents on CD

The enclosed cd contains the following directories and files:

sourcecode Contains the source code for the simulation

experiments Contains the configuration files for the different experiments and theire figures.

scripts Different perl and bash scripts to handle the data output. There is also a gnuplot script.

report The master thesis as pdf.

Nordugrid log database Contains the logfiles from nordugrid from January 2005 to November same year. It is contained both as a Mysql database and a Microsoft access database. It also contains the different queries and output from this. It also contains figures from these queries.

Appendix C

Configuration files

In this chapter the different configuration files is specified. For experiment one all the items is specified, but for the rest of the experiments only the relevant parameters is specified.

C.1 Experiment 1

```
1  ; Configuration file for grid-simulator
2  ; a semicolon indicates a comment
3
4  [Config]
5  users=11
6  sites=10
7  time=180000
8  mu=10.73
9  normalmu=156.75
10 normalsigma=56.55
11
12 path=/s973487/masterthesis/simulation/simu1/exp4/
13
14 ; Various optional values for the [Config] section.
15 ; If not specified the the default is used.
16 ;
17 ; random number seed (might get overwritten at the commandline)
18 seed=100
```

```
19 ;
20 ; print a summary of the read config. This is very usefull when
21 ; debugging "strange" simulation output (default is no)
22 ;printconfigsummary=yes
23 ; prefix for the output dir (created in path). default is
24 ; first letter of trust_eval + first letter of rep_eval
25 ; for example with fuzzy-beta the prefix is fb
26 dirprefix=fbpricevstrust
27
28 ; write the the full output path to std.err
29 ; this can be useful when making scripts (default is no)
30 stderrpath=yes
31
32 ; reuse the jobevent objects (default is yes)
33 ;reuseevents=no
34
35 ; reuse the job objects (default is yes)
36 ;reusejobs=no
37
38 ; use references when calculating reputation (default is yes)
39 ;reputationrefs=yes
40
41 ; number of references to provide and use (default is 4)
42 ;numberofrefs=5
43
44
45 [Default]
46 i=1
47 j=1
48 groups = 0
49 threshold=-0.1
50 trust_eval=fuzzy
51 rep_eval=beta
52 direct_weight=0.6
53 smoothing=1.0
54 sigma=3
55 stranger=0.0
56 ;discrete_ratio=0.5
57 forgetting=0.4
58 base_weight=6
59 ;rep_ratio=0.0
60
61
62
63 [Economy]
64 markup=10
65 depreciation=6
66 discount=0
67 protectionlevel=0.0
68 hardwarecost=45000
69
70
71 ;(optional) section for specifying which resources save combined
       trust info
72 [CtSave]
```

```
default=no
s0=u0 u1
s1=u0 u1 u2 u3
s2=u0 u1 u3

[Resources]
<u0>
time=50000   i=20 j=20
time=110000  i=1  j=1
```

C.2 Experiment 2+3

Since it the power that changes no need for the configuration file to change.

```
; Configuration file for grid-simulator

[Config]
users=20
sites=10
time=180000
mu=10.73
normalmu=156.75
normalsigma=56.55


dirprefix=fbpricevstrust


[Default]
i=1
j=1

threshold=-0.1          ; default threshold
trust_eval=fuzzy
rep_eval=beta
direct_weight=0.6
smoothing=1.0
sigma=3
stranger=0.0
;discrete_ratio=0.5
forgetting=0.4
base_weight=6


[Economy]
markup=10
depreciation=6
discount=0
protectionlevel=0.0
hardwarecost=45000

section for specifying which resources save combined trust info
```

```
39 [CtSave]
40 default=no
41 s0=u0 u1
42 s1=u0 u1 u2 u3
43 s2=u0 u1 u3
44 u0=s0 s1 s2
45
46
47 [Resources]
48 <u0>
49 time=50000  i=20 j=20
50 time=110000 i=1  j=1
51 <s1>
52 time=80000 i=20 j=20
```

C.3 Experiment 4

C.3.1 Site ill behaves

```
1  ; Configuration file for grid-simulator
2
3  [Config]
4  users=15
5  sites=20
6  time=800714
7  mu=10.73
8  normalmu=155.75
9  normalsigma=56.56
10
11 ;
12 ; random number seed (might get overwritten at the commandline)
13 seed=100
14
15
16 [Default]
17
18 threshold=-0.2          ; default threshold
19
20
21 [Economy]
22 cpus=3000               ; The capacity of the site in terms of cpus
23 <s1>
24 discount=20
25
26
27
28 [Bank]
29 initialmoney=1000000
30
31
```

```
32
33
34 section for specifying which resources save combined trust info
35 [CtSave]
36 default=yes
37
38 [Resources]
39 <s1>
40 time=300000 i=10 j=10
```

C.3.2 User ill behaves

```
1  ; Configuration file for grid-simulator
2
3  [Config]
4  users=15
5  sites=20
6  time=800714
7  mu=10.73
8  normalmu=155.75
9  normalsigma=56.56
10
11 ;
12 ; random number seed (might get overwritten at the commandline)
13 seed=100
14
15
16 [Default]
17
18 threshold=-0.2          ; default threshold
19
20
21 [Economy]
22 cpus=3000               ; The capacity of the site in terms of cpus
23 <s1>
24 discount=20
25
26
27
28 [Bank]
29 initialmoney=1000000
30
31
32 section for specifying which resources save combined trust info
33 [CtSave]
34 default=yes
35
36
37 [Resources]
38 <u1>
39 time=50000 i=10 j=10
```

C.4 Experiment 5

```
1  ; Configuration file for grid-simulator
2
3  [Config]
4  users=10
5  sites=10
6  time=100714
7  mu=10.73
8  normalmu=155.75
9  normalsigma=56.56
10
11
12 dirprefix=cpuusage
13
14
15 [Economy]
16 markup=10
17 depreciation=6
18 discount=0
19 protectionlevel=0.0
20 hardwarecost=45000
21 cpus=500
22 <s0>
23 hardwarecost=35000
24
25
26 [Bank]
27 initialmoney=100000
28
29
30 ;(optional) section for specifying which resources save combined
       trust info
31 [CtSave]
32 default=no
33 s0=u0 u1 u2 u3
34 s1=u0 u1 u2 u3
35 s2=u0 u1 u2 u3
36 s3=u0 u1 u2 u3
```

C.5 Experiment 6

```
1  ; Configuration file for grid-simulator
2
3  [Config]
4  users=50
5  sites=100
6  time=1007140
7  mu=10.73
8  normalmu=155.75
9  normalsigma=56.56
10
```

```
11
12  dirprefix=manydiscounts
13
14  threshold=-1.0
15
16
17
18  [Economy]
19  markup=10
20  depreciation=6
21  discount=0
22  protectionlevel=0
23  hardwarecost=45000
24  cpus=600
25  <s0>
26  discount=10
27  <s1>
28  discount=20
29  <s2>
30  discount=25
31  <s3>
32  discount=40
33
34  [Bank]
35  initialmoney=300000
36
37
38  section for specifying which resources save combined trust info
39  [CtSave]
40  default=no
41  s2=u4
42  s3=u4
```

C.6 Experiment 7

```
1   ; Configuration file for grid-simulator
2
3   [Config]
4   users=50
5   sites=100
6   time=1007140
7   mu=10.73
8   normalmu=155.75
9   normalsigma=56.56
10
11
12  dirprefix=manydiscounts
13
14  threshold=-1.0
15
16
17
18  [Economy]
```

```
19 markup=10
20 depreciation=6
21 discount=0
22 protectionlevel=0
23 hardwarecost=45000
24 cpus=600
25 <s0>
26 discount=10
27 <s1>
28 discount=20
29 <s2>
30 discount=25
31 <s3>
32 discount=40
33
34 [Bank]
35 initialmoney=300000
36
37
38 section for specifying which resources save combined trust info
39 [CtSave]
40 default=no
41 s2=u4
42 s3=u4
```

C.7 Experiment 8

C.7.1 Test 1

```
1  ; Configuration file for grid-simulator
2
3  [Config]
4  users=50
5  sites=10
6  time=100714
7  mu=10.73
8  normalmu=155.75
9  normalsigma=56.56
10
11 [Economy]
12 markup=10
13 depreciation=6
14 discount=0
15 protectionlevel=0
16 hardwarecost=45000
17
18 cpus=600
19
20 <s1>
21 discount=10
22 <s2>
```

```
23  discount=20
24  <s3>
25  discount=25
26  <s4>
27  discount=40
28
29
30
31  [Bank]
32  initialmoney=300000
33
34  default=yes
```

C.7.2 Test 2

```
1   ; Configuration file for grid-simulator
2
3   [Config]
4   users=50
5   sites=10
6   time=100714
7   mu=10.73
8   normalmu=155.75
9   normalsigma=56.56
10
11  [Economy]
12  markup=10
13  depreciation=6
14  discount=0
15  protectionlevel=0
16  hardwarecost=45000
17
18  cpus=1200
19
20  <s1>
21  discount=10
22  <s2>
23  discount=20
24  <s3>
25  discount=25
26  <s4>
27  discount=40
28
29
30
31  [Bank]
32  initialmoney=300000
33
34  default=yes
```

C.7.3 Test 3

```
; Configuration file for grid-simulator

[Config]
users=50
sites=20
time=100714
mu=10.73
normalmu=155.75
normalsigma=56.56

[Economy]
markup=10
depreciation=6
discount=0
protectionlevel=0
hardwarecost=45000

cpus=600

<s1>
discount=10
<s2>
discount=20
<s3>
discount=25
<s4>
discount=40



[Bank]
initialmoney=300000

default=yes
```

C.8 Experiment 9

```
; Configuration file for grid-simulator

[Config]
users=50
sites=20
time=100714
mu=10.73
normalmu=155.75
normalsigma=56.56

[Economy]
markup=10
depreciation=6
discount=0
protectionlevel=0
hardwarecost=45000
```

```
17
18  cpus=600
19
20  <s1>
21   discount=10
22  <s2>
23   discount=20
24  <s3>
25   discount=25
26  <s4>
27   discount=40
28
29
30
31   [Bank]
32   initialmoney=300000
33
34   default=yes
```

Appendix D

Source codes

D.1 Pricechange event

D.1.1 pricechangeevent.h

```
1  #if !defined(PRICECHANGEEVENT_H)
2  #define PRICECHANGEEVENT_H
3
4
5  #include <iostream>
6  #include <string>
7  #include "event.h"
8  #include "resourceuser.h"
9  #include "resourcesite.h"
10
11
12 using namespace std;
13 /** The purpose of the Pricechange event is to provide an event
14     where each site receives a new power price. The event will
15     read the price in the "price.txt" filen and distribute the
       current day price to the sites.
16      */
17 class PriceChangeEvent : public Event {
18  private:
19   double price;
20  public:
21   /** Constructs the PriceChangeEvent
```

```
22        \param time is Time of the PriceChangeEvent
23        \param p is the power price
24     */
25    PriceChangeEvent(unsigned int time,double p) : Event(time),price(
          p){}
26
27    bool action(rmap& rUsers,rmap& rSites);
28
29
30    Event* get_event();
31  };
32  #endif
```

D.1.2 pricechangeevent.cpp

```
1   #include "pricechangeevent.h"
2
3   bool PriceChangeEvent::action(rmap& rUsers,rmap& rSites){
4    rmap::iterator it;
5
6           ResourceSite* rs;
7
8                   for (it=rSites.begin();it!=rSites.end();it++)
9                   {
10                  rs=dynamic_cast<ResourceSite*>(rSites[it->first]);
11                  //Update price in the site
12                  rs->update_price(price);
13
14                  }
15
16          return false;
17  }
18
19
20  Event* PriceChangeEvent::get_event(){
21    throw string("Trying to get event from PriceChangeEvent");
22    return e;
23  }
```

D.2 salary event

D.2.1 salaryevent.h

```
1   #if !defined(SALARYEVENT_H)
2   #define SALARYEVENT_H
3
4
5   #include <iostream>
6   #include <string>
7   #include "event.h"
```

```
8  #include "resourceuser.h"
9  #include "resourcesite.h"
10 #include "bank.h"
11
12
13 using namespace std;
14 /** The purpose of the Salary event is to provide an event
15     where each site receives a new power price. The event will
16     read the price in the "salary.txt" filen and distribute the
       current day price to the sites.
17      */
18 class SalaryEvent : public Event {
19  private:
20   double salary;
21  public:
22   /** Constructs the PriceChangeEvent
23       \param time is Time of the SalaryEvent
24       \param p is the salary price
25    */
26   SalaryEvent(unsigned int time,double p) : Event(time),salary(p){}
27
28   bool action(rmap& rUsers,rmap& rSites);
29
30
31   Event* get_event();
32 };
33 #endif
```

D.2.2 salaryevent.cpp

```
1  #include "salaryevent.h"
2
3  bool SalaryEvent::action(rmap& rUsers,rmap& rSites){
4   rmap::iterator it;
5
6
7          ResourceUser* rs;
8
9                  for (it=rUsers.begin();it!=rUsers.end();it++)
10                 {
11                 rs=dynamic_cast<ResourceUser*>(rUsers[it->first]);
12                 //Update price in the site
13                 rs->salary(salary);
14                 }
15
16         return false;
17 }
18
19
20 Event* SalaryEvent::get_event(){
21   throw string("Trying to get event from SalaryEvent");
22   return e;
23 }
```

D.3 creditrejected event

D.3.1 creditrejectedevent.h

```
#if !defined(CREDITREJECTEDEVENT_H)
#define CREDITREJECTEDEVENT_H

#include <iostream>
#include <string>
#include "event.h"
#include "resourcesite.h"
#include "resourceuser.h"


using namespace std;

class CreditRejectedEvent: public Event{
 private:
  string uid;
  Configuration* c;
  bool condition;

 public:
  /** Constructs the CreditRejectedEvent.
      \param u The id of the user (job creator)
      \param t The time of the event
   */
 CreditRejectedEvent(string u,const int t,Configuration* conf,bool
     overbooked) : Event(t),uid(u),c(conf),condition(overbooked)
    {
}

  /** The method will extract the job from the site and
      return it to the job creator.
      \param rUsers A reference to the map containing pointer to
          the users
      \param rSites A reference to the map containing pointer to
          the sites */
  bool action(rmap& rUsers,rmap& rSites);

  Event* get_event();
};
#endif
```

D.3.2 creditrejectedevent.cpp

```
#include "creditrejectedevent.h"

bool CreditRejectedEvent::action(rmap& rUsers,rmap& rSites){
 //cout << "Creditcheck failed for : " << uid << endl;

```

```
6  if(condition)
7  c->countcredit(true);
8  else
9  c->countcredit(false);
10 return false;
11 }
12
13 Event* CreditRejectedEvent::get_event(){
14   throw string("Trying to get event from JobdoneEvent");
15   return e;
16 }
```

D.4 Bank

D.4.1 Bank.h

```
1  #if !defined(BANK_H)
2  #define BANK_H
3
4  #include <iostream>
5  #include <fstream>
6  #include <map>
7  #include <list>
8  #include <vector>
9  #include <string>
10 #include <sstream>
11 #include <iterator>
12
13 using namespace std;
14
15 class Bank{
16
17 private:
18 typedef map <string,double> bankaccounts;
19 map <string,int> jobsawarded;
20 map <string,int>::iterator countjobs;
21
22 bankaccounts accounts;
23
24 double iniusermoney;
25
26 int countrejections,countoverbooked;
27
28 public:
29 Bank();
30
31 //Destructor
32   ~Bank(){}
33
34 void PutSalaryOnAccount(string id, double amount);
35
```

```
36  void Transfer_money_user_to_site(string sid,string uid,double
        amount);
37
38  void Set_Accounts(string id, double amount, bool initial);
39
40  void Set_init_money(double inimoney);
41
42  bool Has_Credit(string uid);
43
44  void count_creditrejected(bool condition);
45
46  void Print_accounts();
47
48
49
50
51
52
53  };
54
55  #endif
```

D.4.2 Bank.cpp

```
1   #include "bank.h"
2
3   Bank::Bank(){}
4
5   void Bank::PutSalaryOnAccount(string id,double amount){
6           map <string,double>::iterator it;
7           //cout << "salary kaldt med amount " << amount << endl;
8           //search the map full of accounts
9           for(it=accounts.begin();it!=accounts.end();it++)
10          {
11          //Only use the user accounts
12          if((it->first)==id)
13                  {
14                  //Add the salary to the account
15                  it->second+=amount;
16                  //cout << "Salary put on : " << (it->first) << "
                        account with the amount " << amount << endl;
17  //cout << "Total amount on account " << (it->second) << endl;
18                  }
19          }
20  }
21
22  void Bank::Transfer_money_user_to_site(string sid,string uid,double
        amount){
23          accounts[uid]-=amount; //Deduct the amount from the user
                account
24          accounts[sid]+=amount; //Add the amount to the site account
25
26          countjobs=jobsawarded.find(sid);
27
```

```
28          if(countjobs==jobsawarded.end())
29          {
30          jobsawarded[sid]=1;
31          }
32          else
33          jobsawarded[sid]+=1;
34
35  }
36
37  void Bank::Set_init_money(double inimoney){
38          iniusermoney=inimoney;
39  }
40
41  void Bank::Set_Accounts(string id, double amount, bool initial){
42
43          //cout << "Set_Accounts kaldt med: " << id << " og " <<
                amount << " og "
44          //<< initial << endl;
45          if(initial)
46                  {
47                  //cout << "inde i initial " << endl;
48                  countrejections=0;
49                  countoverbooked=0;
50                          //Does accounts exist
51                          if(accounts.find(id)==accounts.end())
52                          {
53                                  //cout << "accounten findes i
                                        mappet " << endl;
54                                  //is a user account
55                                  if(id[0]=='u')
56                                          {
57                                          accounts[id]=iniusermoney;
58                                          //cout<<"No account for "
                                                << id <<" existed"<<
                                                endl;
59                                  //No accounts for id with money and
                                        the account is created with

                                                inimoney
60                                          }
61                                          //is a site account
62                                  if(id[0]=='s')
63                                          {
64                                          //cout<<"No account for "
                                                << id <<" existed"<<
                                                endl;
65                                  //No accounts for id with money and
                                        the account is created with

                                                zero, since it is a
                                            site
66                                          accounts[id]=0;
67                                          }
68                          }
69                  }
```

```
70          else
71          accounts[id]=amount;
72
73          //cout << "for  id : " << id << " står der "<< accounts[id]
                << endl;
74
75  }
76
77
78  bool Bank::Has_Credit(string uid)
79  {
80          if(accounts.find(uid)!=accounts.end() && uid[0]=='u'){
81                  //there is an account
82                  if(accounts[uid]>0)
83                  {
84                  //cout << "Credit check for user " << uid << "
                        passed" << endl;
85                  return true;
86                  }
87                  if(accounts[uid] < 0){
88                          //cout << "The current account " << uid <<
                                "has not enough money" << endl;
89                          return false;
90                  }
91          }
92          else
93          cout << "The current account " << uid << "does not exist"
                << endl;
94          return false;
95
96
97  }
98
99  void Bank::Print_accounts()
100 {
101 map <string,double>::iterator it;
102 int maxjobs=0;
103 int sum=0;
104 string winner;
105
106 countjobs=jobsawarded.begin();
107 //Returns the site with the most jobs earned
108 for(countjobs==jobsawarded.begin();countjobs!=jobsawarded.end();
        countjobs++)
109 {
110
111 cout << "Site : " << countjobs->first << " has had " << countjobs->
        second << " job" << endl;
112 sum=sum+jobsawarded[countjobs->first];
113 if(jobsawarded[countjobs->first]>maxjobs)
114         {
115         winner=countjobs->first;
116         maxjobs=jobsawarded[countjobs->first];
117
118         }
```

```
119 }
120
121 cout << endl;
122 cout << "The site : " << winner << " has won with most jobs with "
        << maxjobs << " jobs" << endl;
123 cout << endl;
124 cout << "The amount of credit rejections are : " << countrejections
        << endl;
125 cout << endl;
126 cout << "The amount of overbookings are : " << countoverbooked<<
        endl;
127 cout << endl;
128 cout << "The amount of jobs tracked in the bank : " << sum <<endl;
129 cout << endl;
130
131 for(it=accounts.begin();it!=accounts.end();it++)
132         {
133         cout << "Konto :" << (it->first) << " har indestående " <<
                (it->second) << endl;
134         double temp=(it->second);
135         string testid=(it->first);
136         if(testid[0]=='s')
137                 {
138                 double revenue=temp/jobsawarded[testid];
139                 cout << "Det giver et samlet gennemsnit pris på "
                        << revenue << endl;
140                 }
141         }
142
143
144
145
146 }
147
148 void Bank::count_creditrejected(bool condition)
149 {
150 if(condition)
151 countoverbooked++;
152 else
153 countrejections++;
154 }
```

D.5 resourceuser

D.5.1 resourceuser.h

```
1 #if   !defined(RESOURCEUSER_H)
2 #define RESOURCEUSER_H
3 #include <iostream>
4 #include <map>
5 #include <deque>
```

```
6  #include <string>
7
8  #include "job.h"
9  #include "entity.h"
10 #include "resource.h"
11 #include "resourcesite.h"
12
13 using namespace std;
14
15 /** This class contains the specific methods
16     needed by users. This class inherits the Resource class */
17 class ResourceUser : public Resource{
18  private:
19   /** Overwrites default assignment operator. Eliminates
20       the risk of accidently assigning a ResourceUser
21       object to another ResourceUser object */
22   ResourceUser& operator = (ResourceUser&);
23
24   /** Overwrites default copy constructor. Eliminates the
25       risk of accidently copying a ResourceUser object.
26       The ResourceUser objects must be unique. */
27   ResourceUser(ResourceUser&);
28
29   ///boolean the denotes if jobs should be reused.
30   bool reusejobs;
31  public:
32   /** Constructs a ResourceUser object. The constructor will also
         specify
33       the resource is of type user and check if job objects should
             be
34       re-used.
35       \param i The unique id of the user
36       \param c Pointer to the configuration object */
37   ResourceUser(string i,Configuration* c) : Resource(i,c)
38     {
39       myType=user;
40       reusejobs=true;
41       if(c->valid_key("[Config]","reusejobs")){
42         string reuse=c->get_parameter<string>("[Config]","reusejobs
               ");
43         if(reuse=="no" || reuse=="false"){
44           reusejobs=false;
45           //Only print once
46           if(i=="u0")
47             cout<<"Info: Not re-using job objects"<<endl;
48         }
49       }
50     }
51
52   /// Destructor
53   ~ResourceUser(){
54   }
55
56   /* specific resource user methods */
57
```

```
58    /** Creates a job and returns it. If there is a rejected
59        job in the job queue then this is used. Otherwise a new job
60        is created. If the
61        proposed site is below the trust threshold then it is decided
62        if the site should be retried. This depends on what the
63        current simulation time is. The actual CPU time for the job
              will
64        be a random number between 20 and 40. The informed CPU time
              will
65        depend on the behavioural values.
66        \param sid The proposed site
67        \param rSites The map of sites
68        \param ctime The current simulation time
69        \return Pointer to the job to process   */
70    Job* submit_job(string sid,rmap& rSites,const unsigned int ctime)
          ;
71
72
73    /** Returns the job to the user. The job is examined and
74        a trust calculation is performed. First it is checked that
75        the site is equal to the site that processed the job. Then
76        the direct trust value is calculated. Afterwards the users
77        are asked to rate the site leading to the calculation of a
              reputation
78        trust value. These trust values are combined and the
79        trust value for the site is updated. All the trust values are
               saved
80        in sequences.
81        \param rUsers A reference to the map containing pointers to
              users
82        \param rSites A reference to the map containing pointers to
              sites
83        \param sid The site that processed the job
84        \param j Pointer to the processed job
85     */
86    void job_done(rmap& rUsers,rmap& rSites,const string& sid,Job* j)
          ;
87
88
89    /** Returns a rejected job to the user. The job can later be
90        resubmittet to another site. First the other users are asked
91        to rate the rejecting site in order for a reputation trust
              value
92        to be calculated. If the user's combined trust value for the
93        site is higher than the reputation trust value, then the
              combined
94        trust value is set to the reputation value. Otherwise 0.1 is
95        subtracted from the combined trust value (unless it results
              in a value
96        below -1).
97        \param rUsers A reference to the map containing pointers to
              the users
98        \param rSites A reference to the map containing pointers to
              the sites
99        \param j Pointer to the rejected job
```

```
100         \param ctime The current simulation time
101      */
102     void receive_rejected(rmap& rUsers,rmap& rSites,Job* j,const
            unsigned int ctime);
103
104     void salary(double amount);
105
106 };
107
108 #endif
```

D.5.2 resourceuser.cpp

```
1  #include "resourceuser.h"
2
3  Job* ResourceUser::submit_job(string s,rmap& rSites,const unsigned
       int cTime){
4          double tempprice=0.0;
5          ResourceSite* rs;
6          rmap::iterator rIter;
7          mandatory* m=config->get_mandatory();
8
9  //ask 10 other
10 stringstream ss;
11
12         int cpuTime;
13         int cpus;
14         cpuTime=(int) rg.normal(m->normalmu, m->normalsigma);
15         cpus=(int) rg.normal((m->normalmu)/10,(m->normalsigma)/10);
16
17
18          //id of the purposed site
19         string sid= s;
20
21         //tempstring to locate cheapest site id
22         string tempsite=sid;
23
24
25
26         //Assume that current site sid is the cheapest
27         rs=dynamic_cast<ResourceSite*>(rSites[sid]);
28         tempprice=rs->calculatesiteprice(id,cpus);
29
30
31         //Now we search from the site ID to the end to locate the
               cheapest site
32         for(rIter=rSites.begin();rIter!=rSites.end();rIter++)
33                 {
34                         rs=dynamic_cast<ResourceSite*>(rSites[(
                               rIter->first)]);
35
36
37                                 if(!rs->overbooked(cpus))
38                                 {
```

```
39                                          //if the current site is
                                                cheapen then sid, we
                                                will use this site
40                                          if(rs->calculatesiteprice(
                                                id,cpus) < tempprice)
41                                          {
42                                                  float tmptrust;
43                                                  if(entities.find(
                                                        rIter->first)==
                                                        entities.end())
44                                                  {
45                                                  tmptrust=(config->
                                                        get_properties(
                                                        id))->stranger;
46                                                  }
47                                                  else
48                                                  {
49                                                  tmptrust=
                                                        get_trust_value
                                                        (rIter->first);
50                                                  }
51
52                                          if(tmptrust>=threshold)
53                                                  {
54                                                  tempsite=rIter->
                                                        first;
55                                                  tempprice=rs->
                                                        calculatesiteprice
                                                        (id,cpus);
56                                                  }
57                                          }
58                                  }
59
60                  }
61
62          /*if(entities.find(tempsite)==entities.end() )  {
63                  if(threshold>(config->get_properties(id))->stranger
                        )
64                  throw string("No cheap trustworhty sites");
65          }
66          else if(threshold>get_trust_value(tempsite))
67                  throw string(tempsite+" is not trustworthy and
                        nobody are");
68          */
69          //Setting the id of the cheapest site found
70          sid=tempsite;
71
72
73
74
75
76
77
78  //test if site is known. If not an entity object will be created
79    emap::iterator eit;
```

```
80    eit=entities.find(sid);
81    if(eit==entities.end()){
82      properties* p=config->get_properties(id);
83
84      // Get entity options
85      saveOptions* save=config->get_save_options(id,sid);
86      float sv=p->stranger;
87      float fv=0;
88      //forgetting value is only used with the beta reputation system
89      if(repEval==beta)
90        fv=p->forgetting;
91
92      //Create entity object
93      Entity *e = new Entity(sid,sv,fv,cTime,save);
94
95      //Insert into map
96      entities[sid]=e;
97
98      //Set boolean if entity saves some trust sequences
99      if(entities[sid]->saves())
100       savesSomething=true;
101
102
103   }
104
105
106
107   /* Make a job and submit it.
108    * Check if there is preferences */
109
110
111      Job* j;
112
113
114
115   // Check if an old job has to be re-submitted
116   if(!jobs.empty()){
117     j=jobs[0];
118     jobs.pop_front();
119     // The actual cpuTime needed
120     if(j->rejected())
121       cpuTime=j->get_cpu_time();
122     //else
123 //{
124       //cpuTime=rg.irand(20,40);
125 //      cpuTime=(int) rg.normal(m->normalmu, m->normalsigma);
126       //cout << "Cputime calculated by a normal demand function "
              <<cpuTime << endl;
127   // }
128     j->set_cputime(cpuTime);
129     j->set_rejected(false);
130     j->set_prefered_site(tempsite);
131   }else{
132     // The actual cpuTime needed. Random number between 20 and 40
133       //cpuTime=(int) rg.normal(m->normalmu, m->normalsigma);
```

```
134       j = new Job(cpuTime,id);
135     }
136     /* This determins the informed cputime from a laplace
137      * distribution. This is determins the users behaviour.
138      * The width of the laplace distribution is determined by
139      * abs(min)+abs(max). The number la returned from the laplace
140      * distribution is then shifted. informed =la-min+max */
141     int la=(int)rg.laplace(abs(min)+abs(max));
142     int icputime=cpuTime+la-min+max;
143
144     /* The minimum informed cputime is 1
145      * Anything below that would not be creadable */
146     if(icputime<=0)
147       icputime=1;
148     j->set_informed_cpu_time(icputime);
149
150     /* Check if the site meets the minimum trust
151      * level. If this is not the case then
152      * select another site or check if the site
153      * should be retried. This should only be done
154      * if the job is not a high security job meaning
155      * that there already is a prefered site.   */
156     if(get_trust_value(sid)<threshold && j->get_prefered_site()==”
           none”){
157       /* Finding out how many times the site's jobs has been
158        * rejected and when the last time the trust level
159        * was updated */
160       unsigned int rejected=entities[sid]->get_rejections();
161       unsigned int retryTime = entities[sid]->get_retry_time();
162       unsigned int retries =entities[sid]->get_retries();
163       /* If its the first time the site is rejected
164        * Set the time to retry the site to 2 * current time */
165       if(rejected==0){
166         entities[sid]->set_retry_time(2*cTime);
167         entities[sid]->set_rejections(rejected+1);
168       }
169       /* If the site alrady has been retried but
170        * did not improve set the next retrytime
171        * to (number of retries + 1) * retry time
172        * if its not ready again to retry */
173       if(retries>rejected && retryTime<cTime){
174         entities[sid]->set_rejections(rejected+1);
175         entities[sid]->set_retry_time((retries+1)*retryTime);
176       }
177       /* If its not time to retry the site
178        * pick a random site with a trust level
179        * above the threshold */
180       if(retryTime>cTime)
181         sid=random_entity(threshold,site);
182       else{
183         entities[sid]->set_retries(retries+1);
184       }
185
186       /* If no sites are above the threshold
187        * then pick the most trusted site */
```

```
188       if(sid==="none"){
189         sid=most_trusted(site);
190       j->set_prefered_site(sid);
191       }
192     }
193     // provide references
194     strset ref;
195     deque<string>::iterator it;
196     for(it=references.begin();it!=references.end();it++)
197       ref.insert(*it);
198
199     j->set_user_references(ref);
200     j->set_prefered_site(tempsite);
201
202   // Set the amount of requested cpus between 1 and 10
203   //cpus=cputime;
204   j->set_cpus(cpus);
205
206   //cout << "User: " << id << " has requested " << cpus << " cpus "
        <<endl;
207
208
209     // for debug
210     if(0){
211       cout<<id<<" submitted a job that takes "<<cpuTime<<endl;
212       cout<<" told that it would take "<<icputime<<endl;
213       cout<<"Told me to submit job to "<<sid
214           <<" who i trust: "<<get_trust_value(s)<<endl;
215       if(get_trust_value(s)<threshold)
216         cout<<"Below my threshold for sites!"<<endl;
217     }
218
219    //cout << "Job returned " << endl;
220     // Returning job
221     return j;
222   cout << endl;
223
224
225
226   }
227
228
229
230   void ResourceUser::salary(double amount)
231   {
232   config->putsalaryonaccount(id,amount);
233   }
234
235   void ResourceUser::job_done(rmap& rUsers,rmap& rSites,const string&
        s,Job* j){
236     if(s!=j->get_site()){
237       string error = id + " wrong site specified ";
238       throw(error);
239     }
240     //Find current time
```

```
241   unsigned int cTime=j->time_finished();
242   // Calculate the direct trust value
243   float dt=direct_trust(j);
244
245   // Calculate the reputation
246   float rep = reputation(rUsers,rSites,s,user,j);
247
248   // Finally calculated the trust value
249   float trust = combine_trust(dt,rep);
250
251     //Write trust values
252   entities[s]->update_values(trust,dt,rep,cTime);
253   if(id=="u0" && s=="s0" && 0){
254     cout<<"time: "<<j->time_finished()<<endl;
255     cout<<"ct: "<<trust<< " dt: "<<dt<<" rep: "<<rep<<endl;
256   }
257
258   //Delete job
259   //delete j;
260
261   //reuse job
262   if(reusejobs)
263     jobs.push_back(j);
264   else
265     delete j;
266
267   //Save references
268   deque<string>::iterator it;
269   it=find(references.begin(),references.end(),s);
270   if(it!=references.end())
271     references.erase(it);
272
273   //if more the numberofrefs are provided the remove oldest
274   if(references.size()>=numberofrefs)
275     references.pop_front();
276
277   references.push_back(s);
278
279   /*Debug
280   if(0 && id=="u0"){ //&& s=="s0"){
281     char c;
282     cout<<"I am "<<id<<" and I got job back from: "<<s<<endl;
283     cout<<"My behaviour is :["<<min<<","<<max<<"]"<<endl;
284     cout<<"Job parameters: Cputime informed: "<<j->
          get_informed_cpu_time()
285       <<" actual cpu time : "<<j->get_cpu_time()<<" time given "
286       <<j->get_given_cpu_time()<<endl;
287     cout<<"Direct trust was : "<<dt<<endl;
288     cout<<"Reputation was: "<<rep<<endl;
289     cout<<"combined trust was: "<<trust<<endl;
290     cout<<"My references are :";
291     copy(references.begin(), references.end(), ostream_iterator<
          string>(cout, " "));
292     cout<<endl;
293     strset* sref=j->get_site_references();
```

```
294       if(sref->size()>=4){
295         cout<<"Received the references from site "<<s<<endl;
296         copy((*sref).begin(), (*sref).end(), ostream_iterator<string
              >(cout, " "));
297         cout<<endl;
298       }
299       //cin>>c;
300       *
301       }*/
302   }
303
304   void ResourceUser::receive_rejected(rmap& rUsers,rmap& rSites,Job*
        j,const unsigned int cTime){
305     /* Update trust information
306      * When a job is rejected i cant calculate
307      * a direct trust value nor would I if i could.
308      * I will rely on the reputation of the site  */
309     string sid =j->get_site();
310     //float trust = reputation(rUsers,rSites,sid,user);
311     float trust = reputation(rUsers,rSites,sid,user,j);
312     /* If I trust the site more then everybody else
313      * then set my trust to what the surroundings
314      * believe. If I trust the site more then everybody
315      * else the I just decrease my trust with 0.1 */
316     if(trust<get_trust_value(sid)){
317       entities[sid]->update_values(trust,cTime);
318     }
319     else{
320       trust=get_trust_value(sid)-0.1;
321       if(trust<-0.8)
322         trust=-0.8;
323       entities[sid]->update_values(trust,cTime);
324     }
325     /* Put job back into queue and re-submit
326      * at next opportunity */
327     if(reusejobs)
328       jobs.push_front(j);
329     else
330       delete j;
331     // Or Don't reuse. Delete job
332     //delete j;
333   }
```

D.6 resourcesite

D.6.1 resourcesite.h

```
1  #if !defined(RESOURCESITE_H)
2  #define RESOURCESITE_H
3  #include <iostream>
4  #include <map>
```

```
5  #include <deque>
6  #include <string>
7
8  #include "job.h"
9  #include "entity.h"
10 #include "resource.h"
11 #include "resourceuser.h"
12
13
14 using namespace std;
15
16 /** This class contains the specific methods
17     needed by sites. This class inherits the Resource class.  */
18 class ResourceSite: public Resource{
19  private:
20
21         double siteprice,usersprice,powerprice,trustscore,
               internalprice;
22
23         int markup,depreciation,discount,hardwarecost,jobtime,
               protectionlevel, cpuasked;
24
25         int sitecpus,currentavaiblecpus,occupiedcpus,bookinglimit;
26
27 economy* ep;
28 mandatory* m;
29
30
31
32   /** Overwrites the default assignment operator. Eliminates
33       the risk of accidently assigning a ResourceSite
34       object to another ResourceSite object */
35   ResourceSite& operator = (ResourceSite&);
36
37   /** Overwrites the default copy constructor. Eliminates the
38       risk of accidently copying a ResourceSite object.
39       The ResourceSite objects must be unique. */
40   ResourceSite(ResourceSite&);
41
42   /// The queue is used to store the rejected jobs before they are
         returned to the user.
43   jobqueue rejectQueue;
44
45   //double calculatesiteprice(double trust);
46    double power(double number, int power);
47
48         //Calculates the protectionlevels
49         double calculateprotectionlevel(double disco);
50
51
52  public:
53  /** Constructs a ResourceSite object. The constructor will also
        specify that the resource
54      is of type site.
55      \param i The unique id of the site
```

```
56        \param c Pointer to the configuration object */
57    ResourceSite(string i,Configuration* c): Resource(i,c)
58      {
59        myType=site;
60        ep=c->get_economy(i);
61        markup=ep->markup;
62        depreciation=ep->depreciation;
63        hardwarecost=ep->hardwarecost;
64        sitecpus=ep->sitecpus;
65        protectionlevel=ep->protectionlevel;
66        discount=ep->discount;
67        currentavaiblecpus=sitecpus;
68
69          cout << "site id: " << id <<endl;
70        cout << "my discount is: " << discount <<endl;
71       cout << "my CPUS are is: " << sitecpus <<endl;
72          //cout << "my hardwarecost is: " << hardwarecost <<endl;
73
74
75        usersprice=0;
76        powerprice=0;
77        if(discount>0)
78        protectionlevel=calculateprotectionlevel(discount);
79        cout << "my name is " << id << " and my protectionlevel is: "
              << protectionlevel <<endl;
80  ;
81
82      }
83    /** Destructor. Checks that rejected job queue is empty
84     */
85    ~ResourceSite()
86      {
87        if(!rejectQueue.empty())
88          throw string(id+" had a job that was not returned to job
                creator");
89      }
90    /* specific site functions */
91
92    /** Will (maybe) add a job to the site's job queue. This
93        depends on job creators trust value. If the trust value is
94        below the threshold then the job creator might get
95        retried depending on the current time. If it is rejected
96        then 0 is returned. Otherwise the finishing time for the
97        job is returned.
98        \param j The job to be processed
99        \param rUsers The map of users
100       \param ctime The current simulation time
101       \return The time the job will finish. */
102   int add_job(Job* j,rmap& rUsers,const unsigned int ctime);
103
104   /** Returns a processed job to the job creator. First the
105       user id is checked to make sure it matches the creator of the
106       job. Then the direct trust value for the job is calculated.
              Afterwards
107       the reputation of the job creator is calculated. These are
```

```
108         combined and the trust value for the user is updated. Finally
                a
109         pointer to the finished job is returned.
110         \param rUsers A reference to the map containing pointers to
                the users
111         \param rSites A reference to the map containing pointers to
                the sites
112         \param uid The id of the user
113         \return Pointer to the processed job */
114     Job* job_done(rmap& rUsers,rmap& rSites,const string& uid);
115
116     /** Returns a rejected job. It is checked that the rejected
117         job in the queue actually is created by the specified user
                before returning
118         a pointer to the job.
119         \param uid Id of the job creator
120         \return The rejected job */
121     Job* get_rejected(const string& uid);
122
123     void update_price(double price);
124
125    double calculatesiteprice(string uid, int cpusreq);
126
127    bool ResourceSite::overbooked(int cpurequested);
128
129
130   };
131
132   #endif
```

D.6.2 resourcesite.cpp

```
1   #include "resourcesite.h"
2
3   int ResourceSite::add_job(Job* j,rmap& rUsers,const unsigned int
        cTime) {
4     //Variables used by addJob
5     int cputime,timeGiven,timeDone;
6     double usertrust;
7     unsigned int time;
8     string uid=j->get_creator();
9     cpuasked=j->get_cpus();
10
11
12
13    //test if user is known. If not an entity object will be created
14   emap::iterator eit;
15    eit=entities.find(uid);
16    if(eit==entities.end()){
17      properties* p=config->get_properties(id);
18      //Get entity options
19      saveOptions* save=config->get_save_options(id,uid);
20      float sv=p->stranger;
21      float fv=0;
```

```
22      //forgetting value is only used with the beta reputation system
23      if(repEval==beta)
24        fv=p->forgetting;
25
26      //Create entity object
27      Entity *e = new Entity(uid,sv,fv,cTime,save);
28      //cout << "Entity obejkt lavet for : " << uid << " i tid: " <<
            cTime << endl;
29     //string c;
30  //       cin>>c;
31           //Insert into map
32      entities[uid]=e;
33      //entities[uid]=new Entity(uid,sv,fv,cTime,save);
34
35
36      //Set boolean if entity saves some trust sequences
37      if(entities[uid]->saves())
38        savesSomething=true;
39    }
40
41
42    /* If the user misbehaved earlier but bettered
43     * himself the record of the misbehaviour
44     * should not be kept. The term "once a crook
45     * always a crook" does not apply! */
46    if(get_trust_value(uid)>threshold &&entities[uid]->get_rejections
          ()>0 ){
47      entities[uid]->set_retry_time(0);
48      entities[uid]->set_rejections(0);
49    }
50    /* Does the user have the required trust level
51     * or should it be tested again?
52     * If not send back timeDone=0. This means
53     * that the job was rejected!   */
54    if(get_trust_value(uid)<threshold){
55      /* Finding out how many times the user's jobs has been
56       * rejected and when the last time the trust level
57       * was updated */
58      unsigned int rejected=entities[uid]->get_rejections();
59      //unsigned int lastUpdate = entities[uid]->updatedAt();
60      unsigned int retryTime = entities[uid]->get_retry_time();
61      unsigned int retries =entities[uid]->get_retries();
62      /* If its the first time the user is rejected
63       * Set the time to retry the user to 2* current time */
64      if(rejected==0){
65        entities[uid]->set_retry_time(2*cTime);
66        entities[uid]->set_rejections(rejected+1);
67      }
68      /* If the user alrady has been retried but
69       * did not improve set the next retrytime
70       * to (number of retries + 1) * retry time
71       * if its not ready again to retry     */
72      if(retries>rejected && retryTime<cTime){
73        entities[uid]->set_rejections(rejected+1);
74        entities[uid]->set_retry_time((retries+1)*retryTime);
```

```
75      }
76      /* If its not time to retry the user
77       * simply drop the job. Meaning that 0 is returned  */
78      if(retryTime>cTime){
79        /* Not that it is given 0 cputime
80         * and that I was the site that rejected the job  */
81        j->set_given_cpu_time(0);
82        j->set_site(id);
83        // Storing the job temporarely
84        rejectQueue.push_back(j);
85        return 0;
86      }
87      else{
88        entities[uid]->set_retries(retries+1);
89      }
90    }
91
92
93
94  //Price is being calculated
95  calculatesiteprice(uid,cpuasked);
96
97
98
99    /* Finding out when job is going to start
100    * meaning finding out when the last job in
101    * the job queue finishes   */
102   time=0;
103   if(job_count()>0)
104     time=jobs[jobs.size()-1]->time_finished();
105
106   // job queue is empty start job now
107   if(time<cTime)
108     time=cTime;
109
110   //Needed cpu time
111   cputime=j->get_informed_cpu_time();
112   /* This determins the given cputime from a laplace
113    * distribution. This is determins the sites behaviour.
114    * The width of the laplace distribution is determined by
115    * abs(min)+abs(max). The number la returned from the laplace
116    * distribution is then shifted. timegiven =la-min+max  */
117   int la=(int)rg.laplace(abs(min)+abs(max));
118   timeGiven = cputime+la-min+max;
119   //Give job at least 1 cpu second
120   if(timeGiven<=0)
121     timeGiven=1;
122
123   //finish time
124   j->set_given_cpu_time(timeGiven);
125   timeDone =time+timeGiven;
126
127   jobtime=timeGiven;
128
129   //Adding site id to job
```

```
130   j->set_site(id);
131
132   //provide references
133   strset ref;
134   deque<string>::iterator it;
135   for(it=references.begin();it!=references.end();it++)
136     ref.insert(*it);
137
138   j->set_site_references(ref);
139
140   //Adding Job
141   j->start(time);
142   j->set_finish_time(timeDone);
143   jobs.push_back(j);
144
145
146   //for debug
147   if(0){
148     cout<<"---\nCurrent time is:"<<cTime<<endl;
149     cout<<"There are "<<jobs.size()<<" jobs in "<<id<<" queue"<<
          endl;cout << "called in jobsubmitted" << endl;
150 cout << "My name is " << id << " and I have currently " <<
      currentavaiblecpus << " free cpus out of " << sitecpus << endl;
151
152     cout<<"job will start at:"<<time<<endl;
153     cout<<"job needs "<<cputime<<" cputime"<<endl;
154     cout<<"will give it:"<<timeGiven<<" cputime"<<endl;
155     cout<<"Job will finish at:"<<timeDone<<endl;
156     cout<<"job was created by:"<<j->get_creator()<<endl;
157     cout<<id<<" share "<<config->compare_groups(id,uid)
          <<" with "<<j->get_creator()<<endl;
158     cout<<"my prefered user is:"<<most_trusted(user)<<endl;
159     cout<<"Got a job from:"<<j->get_creator()
          <<" who I trust:"<<get_trust_value(uid)<<endl;
160     if(get_trust_value(uid)<threshold)
161        cout<<"Below my threshold for users"<<endl;
162   }
163
164   if(!overbooked(cpuasked))
165   {
166
167         currentavaiblecpus=currentavaiblecpus-cpuasked; //removes
              the cpus from the capacity
168         occupiedcpus=occupiedcpus+cpuasked;             //adds the
              cpus from the capacity
169         //if(currentavaiblecpus<=sitecpus)
170         entities[uid]->update_cpus(currentavaiblecpus,cTime);
171
172   }
173
174   //returning the time when the job is done
175   return timeDone;
176 }
177
178
179
```

```
180 Job* ResourceSite::job_done(rmap& rUsers,rmap& rSites,const string&
        u){
181   if(jobs.empty()){
182     string error= id+"Job queue was empty when trying to get "+u+"
          job";
183     throw(error);
184   }
185
186   //get job creator
187   Job* j=jobs[0];
188   string uid=j->get_creator();
189   if(uid!=u){
190     string error=id+" job was created by "+uid
191       +" but jobdoneevent was for "+u;
192     throw(error);
193   }
194
195   unsigned int cTime=j->time_finished();
196
197   float dt=direct_trust(j);
198   // save dt
199   //entities[uid]->setDtUpdate(dt,cTime);
200   //float rep = reputation(rUsers,rSites,uid,site);
201   float rep = reputation(rUsers,rSites,uid,site,j);
202   // save rep
203   //entities[uid]->setRepUpdate(rep,cTime);
204
205   float trust = combine_trust(dt,rep);
206
207   //Write new trust value
208   //entities[uid]->setValue(trust,j->timeFinished());
209   //Write trust values
210   entities[uid]->update_values(trust,dt,rep,cTime);
211   entities[uid]->update_price(usersprice,cTime);
212
213
214 // Deduct amount bought from user....
215 config->deductmoney(id,uid,usersprice);
216 //cout << "actualprice " << testprice << " for id : " << dayratio
      << endl;
217
218   //remove finished job from queue
219   jobs.pop_front();
220
221         // Release all the cpus
222         //cout << "inde i jobdone" << endl;
223         //cout << "mit navn er site: " << id << " og jeg har " <<
              currentavaiblecpus << " cpus " << endl;
224         currentavaiblecpus=currentavaiblecpus+(j->get_cpus()); //
              removes the cpus from the capacity
225         occupiedcpus=occupiedcpus-(j->get_cpus());             //
              adds the cpus from the capacity
226         //if(currentavaiblecpus<=sitecpus)
227         entities[uid]->update_cpus(currentavaiblecpus,cTime);
228
```

```
229
230   //Update references
231   deque<string>::iterator it;
232   it=find(references.begin(),references.end(),uid);
233   if(it!=references.end())
234     references.erase(it);
235
236   if(references.size()>=numberofrefs)
237     references.pop_front();
238
239   references.push_back(uid);
240   //Debug
241   if(0 && id=="s0"){ // && uid=="u0"){
242     char c;
243     cout<<"I am "<<id<<" and I finished a job created by: "<<uid<<
          endl;
244     cout<<"My behaviour is : ["<<min<<","<<max<<"]"<<endl;
245     cout<<"Job parameters: Cputime informed: "<<j->
          get_informed_cpu_time()
246          <<" actual cpu time : "<<j->get_cpu_time()<<" time given "
247          <<j->get_given_cpu_time()<<endl;
248     cout<<"Direct trust was : "<<dt<<endl;
249     cout<<"Reputation was: "<<rep<<endl;
250     cout<<"combined trust was: "<<trust<<endl;
251     strset* sref=j->get_user_references();
252     if(sref->size()>=4){
253       cout<<"Received the references from user "<<uid<<endl;
254       copy((*sref).begin(), (*sref).end(), ostream_iterator<string
            >(cout, " "));
255       cout<<endl;
256     }
257     //cin>>c;
258   }
259
260
261
262   return j;
263
264 }
265
266
267
268 double ResourceSite::calculatesiteprice(string uid, int cpusreq){
269
270         double trust;
271         double risk,costs,totalcost,maxdiscount;
272
273
274         //cout << "Calculateprice kaldt af id :" << uid << endl;
275         emap::iterator pit;
276         pit=entities.find(uid);
277
278         //if there is no user entity use a default value
279         if(pit==entities.end()){
280         //cout << uid <<" er ikke funde i entity mappet" << endl;
```

```
281         properties* pi=config->get_properties(id);
282         trust=0; //pi->stranger;
283         //cout << "trust er lig med stranger i  calculatestring,
                hvor id er lig="<< id<< endl;
284         }
285         else
286         {
287         //cout << " inde i else stamenent i calculatesite hvor uid
                ="<< uid<< endl;
288         trust=get_trust_value(uid);
289
290         }
291
292                 risk=(1-trust)*10;
293
294         //Test to see if the uses trustscore is above the threshold
295         if(trust<threshold)
296 {
297         risk=80;
298
299 }
300
301                 costs=hardwarecost+(24*1.3*((double)powerprice/100)
                        )*1000;
302                 double tempdepre=(double) (100+depreciation)/100;
303                 double temprisk=(double) (100+risk)/100;
304                 double tempmarkup=(double) (100+markup)/100;
305
306                 totalcost=costs*power(tempdepre,2)*power(temprisk
                        ,2)*power(tempmarkup,2);
307
308
309
310                 usersprice=totalcost/1000;
311                 //cout << "lige foer return userprice " <<
                        usersprice<< endl;
312                 internalprice=hardwarecost/1000+(24*1.3*((double)
                        powerprice/100));
313
314                 //cout << "Usersprice er "<< usersprice << " og
                        Internalprice er " << internalprice << " med
                        trust " << risk << " og med power pris pa " <<
                        powerprice <<endl;
315
316                 //This is the maximum discount where there is a
                        breakeven between cost and price
317         maxdiscount=100*((usersprice-internalprice)/usersprice);
318
319
320         if(discount>0 && discount<maxdiscount)
321         {
322
323
324           if((occupiedcpus+cpusreq)<bookinglimit)
325                 {
```

```
326                     usersprice=(1-((double)discount/100))*usersprice;
327
328                     }
329             }
330             else
331             {
332             if(discount>maxdiscount)
333             usersprice=10*usersprice;
334
335             }
336
337             return usersprice;
338
339
340     }
341
342     void ResourceSite::update_price(double price)
343     {
344             powerprice=price;
345             //cout<< " powerprice i resource site kaldt af id " << id
                    << " med pris " << powerprice << endl;
346
347
348     }
349
350
351     Job* ResourceSite::get_rejected(const string& u){
352       /* Exit program it the queue of rejected jobs
353        * is empty. This should not happen. Throw exception */
354       if(rejectQueue.empty()){
355         string error= id + " is missing a rejected job created by  "+
356           u+". rejectQueue was empty";
357         throw(error);
358       }
359       jobqueue::iterator jit;
360       for (jit=rejectQueue.begin(); jit != rejectQueue.end(); jit++){
361         if((*jit)->get_creator()==u){
362           rejectQueue.erase(jit);
363           return (*jit);
364         }
365       }
366       /* Error. Job created by u was not found
367        * as a rejected job. Should not happen
368        * throw exception */
369       string error= id + " is missing a rejected job created by  "+u;
370       throw(error);
371     }
372
373
374     double ResourceSite::power(double x, int n)
375       {
376         double product = 1.0;
377         int i;
378       if ( n == 0)
379           return 1.0;
```

```
380       else
381         {
382         for (i = 1; i <= n; i++)
383 {
384             product= product*x;
385 }
386           return product;
387         }
388     } // end of power
389
390
391 double ResourceSite::calculateprotectionlevel(double disco)
392 {
393 m=config->get_mandatory();
394 double normalmu=m->normalmu;
395 double z;
396 double normalsigma=m->normalsigma;
397 z=rg.inversecdf(disco/100);
398
399 //cout << " z er lig :" << z << endl;
400
401 return normalmu/5+z*(normalsigma)/5;
402 }
403
404 bool ResourceSite::overbooked(int cpurequested)
405 {
406
407 if((currentavaiblecpus-cpurequested)<0)
408 return true;
409 else
410 return false;
411 }
```

D.7 configuration

D.7.1 configuration.h

```
1  #if !defined(CONFIGURATION_H)
2  #define CONFIGURATION_H
3  extern "C" {
4    double parse_this(char *);
5    void setup();
6    int error_flag();
7    //double parseresult;
8    //int ERRORFLAG;
9  }
10
11 #include <iostream>
12 #include <fstream>
13 #include <queue>
14 #include <map>
```

```
15 #include <list>
16 #include <vector>
17 #include <set>
18 #include <string>
19 #include <sstream>
20 #include <iterator>
21 #include <ctime>
22 #include "saveoption.h"
23 #include "bank.h"
24
25 using namespace std;
26
27
28 typedef set<string> strset;
29 typedef list<int> intlist;
30 typedef set<int > intset;
31 typedef vector<string> strvec;
32 typedef map<string, strvec> submap;
33 typedef map<string,submap> cmap;
34 typedef map<string,string> strmap;
35
36 struct properties{
37   int i;
38   int j;
39   intset groups;
40   double threshold;
41   string trust_eval;
42   string rep_eval;
43   double direct_weight;
44   double sigma;
45   double smoothing;
46   double stranger;
47   //Only used if discrete combination
48   double discrete_ratio;
49   //Only used if beta
50   double forgetting;
51   double base_weight;
52   //Only used if discrete reputation
53   double rep_ratio;
54 };
55 //Map for resource options
56 typedef map<string,properties> romap;
57
58 struct mandatory{
59   unsigned int users;
60   unsigned int sites;
61   unsigned int time;
62   double mu;
63   double normalmu;
64   double normalsigma;
65   string path;
66   unsigned long int seed;
67   properties rDefaults;
68 };
69
```

```
70  //Parameters to calculate prices for the sites
71  struct economy{
72          unsigned int markup;
73          unsigned int depreciation;
74          unsigned int discount;
75          unsigned int hardwarecost;
76          double protectionlevel;
77          unsigned int sitecpus;
78  };
79
80  //map of parameters and sites.
81  typedef map<string,economy> ecmap;
82
83
84  ///Structure containing id and time where a resource is to be added
85  struct addition{
86    string id;
87    unsigned int time;
88  };
89  //Structure used to order addition according to event time
90  struct addcomparison
91  {
92    bool operator () (addition * left, addition * right)
93    {
94      int l=(*left).time;
95      int r=(*right).time;
96      return l > r;
97    }
98  };
99
100 typedef priority_queue<addition*, vector< addition*,allocator<
        addition*> >,
101                        addcomparison > addqueue;
102 typedef list<addition*> addlist;
103
104 //Structure containing the id and time where the resource will
        change behavior
105 struct change{
106   string id;
107   unsigned int time;
108   int i;
109   int j;
110 };
111 //Structure used to order changes according to event time
112 struct changecomparison
113 {
114   bool operator () (change * left, change * right)
115   {
116     int l=(*left).time;
117     int r=(*right).time;
118     return l > r;
119   }
120 };
121
```

```
122 typedef priority_queue<change*, vector< change*,allocator<change*>
    >,
123                       changecomparison > changequeue;
124 typedef list<change*> changelist;
125
126
127 /** The purpose of this class
128     is to read the .conf file and set up the simulation.
129     The objects can query the configuration object for
130     the specifics of the simulation.*/
131 class Configuration{
132
133 private:
134   //map containing the config file as strings
135   cmap cfile;
136
137   economy ep;
138
139   //structure the contain the bare minimum of options
140   mandatory simOptions;
141   //map that contain properties for individual resources
142   romap resources;
143   /* flag that denotes that the simulation uses a
144      commandline specified random number seed */
145
146   ecmap economyressources;
147
148   Bank nationaltreasure;
149
150
151   bool commandlineseed;
152
153   //Queue of additions
154   addqueue additions;
155
156   //list of additions (used for summary )
157   addlist listofadds;
158
159   //Time(s) where additions will happen
160   intset addtimes;
161
162   //Queue of changes
163   changequeue changes;
164
165   //list of changes (used for summary)
166   changelist listofchanges;
167
168   //Time(s) where changes will happen
169   intset changetimes;
170
171   ///Default saving options object
172   SaveOption* sOptions;
173
174   //the heuristics map (might not be used)
175   strmap heuristic;
```

```
176     //bool that denotes if custom heuristics are used
177     bool customHeuris;
178
179     /** Method that reads the entire configuration
180         file into a map. Lines beginning with ;
181         are skipped. When a line starts with [something]
182         is is a section. When a line starts with
183         <something> is is a subsection (in the section).
184         \param file The configuration file name
185      */
186     void read_whole_config(const string& file);
187
188     /* Boolean functions for checking various inputs */
189
190     ///Checks if line starts with '[' and has a ']' in it
191     bool valid_section(string& line);
192
193     ///Checks if line starts with '<' and has a '>' in it
194     bool valid_subsection(string& line);
195
196     ///Checks if line appears to be an id (must start with 'u' or 's'
            and have a number
197     bool valid_id(const string& line);
198
199     /** Gets the next elemet (from the iterator position) from a
200         string vector. It checks that end of the vector has not
201         been reached.   */
202     string& get_next_token(const strvec&,strvec::iterator& );
203
204     /** Tokenize a string. The string str is tokenized, using
205         the delimiters, and the result is inserted into the
206         vector.
207         \param str The string to be parsed
208         \param tokens The result vector
209         \param delimiters The delimiters to use  */
210     void tokenize(const string& str,
211                   vector<string>& tokens,
212                   const string& delimiters = "␣;=\t");
213
214     ///Removes begining and ending chars from a string
215     string trim(string& s,const string& drop = "␣");
216
217     ///Check if to properties structures contain equal values
218     bool compare_properties(const properties&,const properties&);
219
220     /** Read the mandatory options from the cmap. Also the default
221         properties are read.   */
222     void set_mandatory_options();
223
224     /** Set the properties from a section and subsetion. If
225         not the default properties cmap["[Default]"]["none"]
226         then only the values that are there are read
227         \param p The properties struct where the read values will be
                saved
228         \param section The section to read from
```

```
229         \param subsection The subsection to read from   */
230     void set_properties(properties& p,
231                         const string& section = "[Default]",
232                         const string& subsection = "none");
233
234   /*set the different economic parameters for sites and subsites*/
235   void set_economies();
236
237   // set the parameters for the economy
238   void set_economy(economy& test, const string& section="[Economy]",
          const string&
239   subsection="none");
240
241
242
243   //set up the bank
244   void set_bank();
245
246
247
248   bool compare_economy(const economy& e1,const economy& e2);
249
250
251
252     /** Reads the section cmap[section] for subsections.
253         If a subsection is there then the set_properties method
254         is called to check for custom properties. It is assumed
255         that a subsection is a <resource_id>.   */
256     void set_resources(const string& section = "[Resources]");
257
258     /** Reads the section cmap[section] for possible additions.
259         Meaning it looks for a format of:
260         time=some_unsigned_int some_resource_id
261         Example: time=15000 u50
262
263         The behavior can also be specified (optional)
264         Example: time=15000 u50 i=10 j=10
265
266         If no behavior is specified then the default values
267         are used. It does not matter if its written as i,j or j,i*/
268     void set_additions(const string& section = "[Add]");
269
270     /** Reads the section cmap[section] for subsections
271         These subsections are searched for possible changes
272         in the resource's behavior.
273         Meaning it looks for a format of:
274         time=some_unsigned_int i=some_int j=some_int
275         Example: time=15000 i=10 j=5
276         It does not matter if its written as i,j or j,i */
277     void set_changes(const string& section = "[Resources]");
278
279     /** The method scans the sections [CtSave],[DtSave] and [RtSave]
280         (in the cmap) for possible saving info. If a section is
281         left out the default action is save nothing  */
282     void set_saveoptions();
```

```
283
284   /** Helper function for set_saveoptions.   */
285   void set_savetype(const string& section,const string& type);
286
287   /** Checks if there is a section [Heuristics] in the cmap
288       If its there then a heuristics map is created
289       with the values specified.  */
290   void set_heuristic();
291
292   /** Finds key in cmap[section][subsection] and returns
293       the tokenized result as a string vector using the
294       delimiters specified. This method is used by the
295       template functions.
296       \param section The section to search for
297       \param key The key to search for
298       \param subsection The subsection to look in
299       \param delimiters The delimiters to use when tokenizing   */
300   strvec find_and_tokenize(const string& section,
301                            const string& key,
302                            const string& subsection="none",
303                            const string& delimiters= " =;\t");
304   void init();
305  public:
306   /** Constructs and automatically loads the configuration
307       file specified.
308       \param file The configuration filename.
309    */
310   Configuration(const string& file);
311   /** Constructs and automatically loads the configuration
312       file specified. Also the value s overwrites any randomnumber
313       seeds specified in the config file
314       \param file The configuration filename.
315       \param s The random number seed to use in the simulation
316    */
317   Configuration(const string& file,unsigned long int s);
318
319   /// Destructor
320   ~Configuration(){
321     //Check for unprocessed additions
322     if(!additions.empty()){
323       cout<<"There are additions that was not processed!"<<endl;
324       while(!additions.empty()){
325         addition* a= additions.top();
326         delete a;
327         additions.pop();
328       }
329     }
330     //Check for unprocessed changes
331     if(!changes.empty()){
332       cout<<"There are changes that was not processed!"<<endl;
333       while(!changes.empty()){
334         change* c= changes.top();
335         delete c;
336         changes.pop();
337       }
```

```
338       }
339       //Delete saveOptions
340       delete sOptions;
341
342     }
343
344
345      /** Template function that returns the value of a key in a
346         particular section. The type should explicitly defined
347         when using function. For example :
348         double value=configuration.get_parameter<double>("section","
              key");
349         The function is defined in the header file for portability
350         between different compilers.
351         \param section The section to search in (cmap[section])
352         \param key The key to look for (in cmap[section][subsection])
353         \param subsection The subsection to search in.
354     */
355     template <class T>
356       T get_parameter(const string& section,
357                       const string& key,
358                       const string& subsection="none"){
359       //Get the tokenized line in configuration file.
360       strvec tokenized=find_and_tokenize(section,key,subsection);
361       stringstream ss;
362       T result;
363       //The first string in tokenized must be key
364       if(tokenized[0]!=key)
365         throw string("lost keyword: "+key+" when tokenizing line");
366       //vs[1] is the value of type T that should be returned
367       ss<<tokenized[1];
368       if(!(ss>>result))
369         throw string(key+" has invalid type: "+tokenized[1]);
370       //clear stringstream
371       ss.clear();
372       ss.str("");
373
374       return result;
375     }
376
377     /** Template function that returns a set of values from a key in
              a
378         particular section. The type should explicitly defined
379         when using function. For example :
380
381         set<string> stringset=configuration.get_parameterset<string
              >("section","key");
382
383         The function is defined in the header file for portability
384         between different compilers.
385         \param section The section to search in (cmap[section])
386         \param key The key to look for (in cmap[section][subsection])
387         \param subsection The subsection to search in. */
388     template <class T>
389       set<T> get_parameterset(const string& section,
```

```
390                                 const string& key,
391                                 const string& subsection="none"){
392     set<T> result;
393     //Get the tokenized line in configuration file.
394     strvec tokenized=find_and_tokenize(section,key,subsection);
395     strvec::iterator it;
396     stringstream ss;
397
398     //First element in the list should be key
399     if(tokenized[0]!=key)
400       throw string("An error occured when getting the parameter
              list for "+key);
401
402     //Erasing first element
403     tokenized.erase(tokenized.begin());
404
405     //The rest of the element should be of type T
406     for(it=tokenized.begin();it!=tokenized.end();it++){
407       ss<<*it;
408       T tmp;
409       if(!(ss>>tmp))
410         throw string(*it+" is of the wrong type. Noticed when
              getting parameter set for "+key);
411       result.insert(tmp);
412       ss.clear();
413       ss.str("");
414     }
415     return result;
416   }
417
418   /** Template function that will read a file that
419       contain pairs. Example content of file:
420
421       string1 int1
422       string2 int2
423       ....
424
425       The function will return a list of pointers
426       to pairs. For example for the above file
427       list<pair<string,int>* > will be returned.
428
429       Important: Remember to delete the pairs
430       you are done using them since this function
431       allocates memory to the pairs but it does
432       not free the memory when you are have finished
433       using the pairs. Another thing to remember
434       is that the reading will stop when one of the
435       values had an incorrect type. For example
436       when reading the file :
437
438       string1 int1
439       string2 int2
440       string3 string4
441
442       only the first two pairs will be read. Modify
```

```
443         the function to throw an exception it its needed. */
444     template <class A, class B>
445       list<pair<A,B>* > read_file(const string& file){
446       //open file
447       ifstream inFile(file.c_str());
448       //if file cant be opened
449       if(!inFile)
450         throw string("Invalid file: "+ file);
451       //the list of pairs returned
452       list<pair<A,B>* > result;
453       //while there is still data
454       while(inFile){
455         A a;
456         B b;
457         if(inFile>>a && inFile>>b){
458           pair<A,B>* p= new pair<A,B>;
459
460           p->first=a;
461           p->second=b;
462           //push back the pointer to the pair
463           result.push_back(p);
464         }
465       }
466       //close file
467       inFile.close();
468
469       return result;
470     }
471     /** Check if the key is valid. It looks in the config map
472         (the read config file) and checks if the key is there
473         \param section The section to search in (cmap[section])
474         \param key The key to look for (in cmap[section][subsection])
475         \param subsection The subsection to search in   */
476     bool valid_key(const string& section,
477                    const string& key,
478                    const string& subsection="none");
479
480     ///Return a pointer to the default properties structure
481     properties* get_default_properties(){
482       return &simOptions.rDefaults;
483     }
484     ///Check if there is custom heuristics
485     bool custom_heuristics(){
486       return customHeuris;
487     }
488     ///Return a pointer to the heuristics map
489     strmap* get_heuristics(){
490       if(!customHeuris)
491         throw string("trying to get custom heuristics but none have
              been read");
492       return &heuristic;
493     }
494     /** Return a pointer to the resource's properties structure.
495         If the resource does not have a unique properties
496         structure associated with it then a pointer to the
```

```
497         default properties structure is returned.
498         \param id The resource id of the properties to search for
                */
499     properties* get_properties(const string& id);
500     /** Return a pointer to the saveOptions structed associated
501         between the two resources. The method returns what id1
502         save about id2.
503         \param id1 The resource that want to know what it should save
504         \param id2 The resource who's information might get saved by
                id1    */
505
506      economy* get_economy(const string& id);
507      /*
508        Returns a pointer to the economy properties
509        */
510     saveOptions* get_save_options(const string& id1,const string& id2
            );
511
512     /** Returns a pointer to a string set contain the resource id's
513         that a resource (id) should use as its (initial) reputation
514         group. The reputation goups is defined as:
515
516         if there are 100 resource users in the simulation,
                numberofrefs=4
517         and id = u54 then the string set containing u55, u56, u57,
                u58 is returned.
518
519         if there are 100 resource if numberofrefs=5 and id = u98 then
                 the
520         string set containing u99, u0, u1, u2, u3 is returned.
521
522         The string set is created (using new) so remember to
523         delete it when done using it.
524         \param id The resource whom then reputation group is intented
                 for. */
525     strset* get_rep_group(const string& id);
526
527     /** Returns a pointer to the mandatory structure. This stucture
528         number users, sites, save path ect. */
529     mandatory* get_mandatory(){
530       return &simOptions;
531     }
532
533     /** The Bank Manipulating functions  */
534
535
536   void Configuration::countcredit(bool check);
537
538   void deductmoney(string sid,string uid,double amount);
539
540
541   void setaccounts(string uid, double amount, bool initial);
542
543   //Manipulate the bank object and in particular the users accounts
544   void putsalaryonaccount(string id, double amount);
```

```
545
546 bool hascredit(string userid);
547
548 //printsout the accounts
549 void print_accounts();
550
551   /** Frontend function to the expression parser. This function
552       adds a newline to the string '\n' (for the C flex/bison calc.
              l/calc.y)
553       and creates a char[] that is submitted to the C expression
              parser.
554       Check the file calc.y - function setup() - for the math
              functions
555       available as default. The result is then returned as a double
556       IMPORTANT : math.h uses the name log to the natural logarithm
              (ln)
557       and log10 to the 10 base version. This is done to avoid
              confusion with
558       the *nix program ln.
559       For example :
560
561       if string exp="ln(10)";
562       then parse_exp(exp)==log(10);
563
564       if string exp ="log(10)";
565       then parse_exp(exp)==log10(10);
566
567       Make sure that you only use log when you mean log10
568       \param exp The expression to parse    */
569   double parse_exp(string exp);
570
571   /** Calculated the number of shared groups between resource id1
572       and resource id2.   */
573   int compare_groups(const string& id1,const string& id2);
574
575
576   /* methods for changing and adding resources to the simulation */
577   ///Check if there are more resources to be added at the current
          time
578   bool more_additions(unsigned int ctime);
579
580   ///Return a pointer to the next addition structure(remember to
          delete it)
581   addition* get_next_add();
582
583   ///Return a set of times where resources should be added
584   intset& get_addtimes(){
585     return addtimes;
586   }
587
588   ///Check if there are more resources to be change at the current
          time
589   bool more_changes(unsigned int ctime);
590
```

```
591   ///Return a pointer to the next change structure(remember to
          delete it)
592   change* get_next_change();
593
594   ///Return a set of times where resources should change
595   intset& get_changetimes(){
596     return changetimes;
597   }
598
599   /*Various test functions*/
600
601
602   ///Prints a summary of the read config file
603   void print_summary();
604   ///print the entire read config (the cmap - used for testing)
605   void print_config();
606   ///print the content of a properties structure (used for testing)
607   void print_properties(const properties* p);
608   ///print the content of the resources map (containing the
          properties structures)
609   void print_resources();
610   ///print a string vector
611   void print_strvec(const strvec&);
612   ///print the difference between resource (id) properties p and
          the default properties
613   void print_difference(const string& id,const properties* p);
614 };
615 #endif
```

D.7.2 configuration.cpp

```
1  #include "configuration.h"
2
3  Configuration::Configuration(const string& file){
4    commandlineseed=false;
5    read_whole_config(file);
6    init();
7  }
8
9  Configuration::Configuration(const string& file,unsigned long int s
       ){
10   //A commandline seed is being used
11   commandlineseed=true;
12   //Setting seed
13   simOptions.seed=s;
14   read_whole_config(file);
15   init();
16 }
17 void Configuration::init(){
18   set_mandatory_options();
19
20   set_resources();
21
22   set_economies();
```

```
23
24    set_additions();
25
26    set_changes();
27
28    set_saveoptions();
29
30    set_bank();
31
32    //if there is specified custom heuristics then used those
33    set_heuristic();
34    //Setup exp parser
35    setup();
36
37    //print summary
38    if(valid_key("[Config]","printconfigsummary")){
39      string printsum=get_parameter<string>("[Config]","
            printconfigsummary");
40      if(printsum=="yes" || printsum=="true"){
41        print_summary();
42      }
43    }
44  }
45
46  void Configuration::read_whole_config(const string& file){
47    string line;
48    string section="none";
49    string subsection="none";
50
51    //Open config file
52    ifstream inFile(file.c_str());
53
54    //Throw exception if can open file
55    if(!inFile)
56      throw string("Invalid config file: "+ file);
57
58    //Read whole config into configfile map.
59    while(! inFile.eof() ){
60      //Read line
61      getline(inFile,line);
62      //First trim the line
63      line=trim(line);
64
65      //if line is not a comment
66      if(line[0]!=';'){
67        if(valid_section(line)){
68          //line is a [Section]
69          section=line.substr(line.find_first_of('[',0),
70                              line.find_first_of(']',0)+1);
71          //Reset subsection to "none"
72          subsection ="none";
73        }
74        else if(valid_subsection(line)){
75          //line is a <Subsection>
76          subsection=line.substr(line.find_first_of('<',0),
```

```
77                                  line.find_first_of('>',0)+1);
78          //cout<<"subsection: "<<subsection<<endl;
79        }
80        else if(line.size()>0){
81          //Push back line if size>0. No reason to store empty lines
82          cfile[section][subsection].push_back(line);
83        }
84      }
85    }
86    //close file
87    inFile.close();
88  }
89  bool Configuration::valid_subsection(string& s){
90    string line=s;
91    line=trim(line);
92    /* The requirements for a valid subsection
93       is the it starts with the char '<'
94       and that there is a ending '>' somewhere. */
95    if(line[0]!='<')
96      return false;
97    if(line.find(">",0)!= string::npos)
98      return true;
99    else
100     return false;
101 }
102
103 bool Configuration::valid_section(string& s){
104   string line=s;
105   //remove whitespace
106   line=trim(line);
107   /* The requirements for a valid section
108      is the it starts with the char '['
109      and that there is a ending ']' somewhere. */
110   if(line[0]!='[')
111     return false;
112   if(line.find("]",0)!= string::npos)
113     return true;
114   else
115     return false;
116 }
117
118 bool Configuration::valid_key(const string& section,
119                               const string& key,
120                               const string& subsection){
121   //Check if section is defined in cfile map
122   if(cfile.find(section)==cfile.end())
123     return false;
124
125   //Check if subsection is defined in cfile map
126   if(cfile[section].find(subsection)==cfile[section].end())
127     return false;
128
129   //Find key in the string vector at cfile[section][subsection]
130   strvec::iterator it=cfile[section][subsection].begin();
131   while(it!=cfile[section][subsection].end()){
```

```
132     //if the line *it starts with key.
133     string line=*it;
134     if(trim(line).substr(0,key.size())==key)
135       return true;
136 
137     it++;
138   }
139   return false;
140 }
141 
142 bool Configuration::valid_id(const string& id){
143   //id must start with u or s
144   if(id[0]!='u' && id[0]!='s')
145     return false;
146   //the rest of the id must be a number
147   string rnumber=id.substr(1,id.size());
148   //Check number
149   int number;
150   stringstream ss;
151   ss<<rnumber;
152   if((ss>>number)){
153     return true;
154   }
155   //Clear stringstream
156   ss.clear();
157   ss.str("");
158   return false;
159 }
160 
161 
162 string& Configuration::get_next_token(const strvec& v,strvec::
      iterator& it){
163   /* Simple function that advances the iterator one step and checks
164      if the end is reached. If it is then a exception is thrown.
           Else
165      a reference to the element is returned.
166 
167      (The reason for this function is just to avoid explicitly
          checking
168      if the end is reached each time.)
169   */
170   it++;
171   if(it==v.end())
172     throw string("trying to get next token, but there are no more")
          ;
173 
174   return *it;
175 }
176 
177 void Configuration::set_economies(){
178 
179         //Setting default
180         set_economy(ep);
181         string section="[Economy]";
182         // cout << "inde i set_economies" << endl;
```

```
183
184         for(submap::iterator it=cfile[section].begin();it!=cfile[
                section].end();it++)
185           if(it->first!="none")
186           {
187
188                   economy test=ep;
189                   set_economy(test,section,it->first);
190
191                   if(compare_economy(test,ep)!=true)
192                   {
193                           string eid = (it->first).substr((it->first)
                                .find("<",0)+1,
194                                        (it->first).find(">",0)-1);
195
196                           //save the properties struct for
                                economyressource eid
197                           economyressources[eid]=test;
198
199                   }
200          }
201 }
202
203
204 void Configuration::set_economy(economy& e, const string& section,
        const string& subsection){
205         if(section!="[Economy]")
206                 {
207                 throw string ("Economy parameters was not specified
                        in section "+section+" (Might not be a default
                        value to use either)");
208                 }
209
210         if(subsection=="none"){
211                 e.markup=get_parameter<int>(section,"markup",
                        subsection);
212                 e.depreciation=get_parameter<int>(section,"
                        depreciation",subsection);
213                 e.discount=get_parameter<int>(section,"discount",
                        subsection);
214                 e.protectionlevel=get_parameter<double>(section,"
                        protectionlevel",subsection);
215                 e.hardwarecost=get_parameter<int>(section,"
                        hardwarecost",subsection);
216                 e.sitecpus=get_parameter<int>(section,"cpus",
                        subsection);
217                 //cout<<"Economy parameters set";
218                 }
219         else
220         {
221         //Setting properties for a specific resources
222         if(valid_key(section,"markup",subsection))
223         {
224          e.markup=get_parameter<int>(section,"markup",subsection);
```

```
225           cout << "inde i " << section << " med " << subsection << "
                  hvor special markup kaldt" << endl;
226           }
227           if(valid_key(section,"depreciation",subsection))
228            e.depreciation=get_parameter<int>(section,"depreciation",
                  subsection);
229
230           if(valid_key(section,"discount",subsection))
231            e.discount=get_parameter<int>(section,"discount",
                  subsection);
232
233           if(valid_key(section,"protectionlevel",subsection))
234            e.protectionlevel=get_parameter<double>(section,"
                  protectionlevel",subsection);
235
236            if(valid_key(section,"hardwarecost",subsection))
237                   {
238                    e.hardwarecost=get_parameter<int>(section,"
                         hardwarecost",subsection);
239                   cout << "inde i " << section << " med " <<
                        subsection << "hvor special hardwarecost kaldt"
                         << endl;
240                   }
241
242           if(valid_key(section,"cpus",subsection))
243            e.sitecpus=get_parameter<int>(section,"cpus",subsection);
244           }
245
246
247
248   }
249
250   void Configuration::set_properties(properties& p,
251                                      const string& section,
252                                      const string& subsection){
253     //sets property struct p with the values in section,subsection
254
255     if(section=="[Default]" && subsection=="none"){
256       //Setting default properties
257       p.i=get_parameter<int>(section,"i");
258       p.j=get_parameter<int>(section,"j");
259       p.groups=get_parameterset<int>(section,"groups");
260       p.threshold=get_parameter<double>(section,"threshold");
261       p.trust_eval=get_parameter<string>(section,"trust_eval");
262       p.rep_eval=get_parameter<string>(section,"rep_eval");
263       p.direct_weight=get_parameter<double>(section,"direct_weight");
264       p.smoothing=get_parameter<double>(section,"smoothing");
265       p.sigma=get_parameter<double>(section,"sigma");
266       p.stranger=get_parameter<double>(section,"stranger");
267       //Only used if discrete
268       if(p.trust_eval=="discrete")
269         p.discrete_ratio=get_parameter<double>(section,"
               discrete_ratio");
270       //Only used if beta
271       if(p.rep_eval=="beta"){
```

```
272         p.forgetting=get_parameter<double>(section,"forgetting");
273         p.base_weight=get_parameter<double>(section,"base_weight");
274       }
275       if(p.rep_eval=="discrete")
276         p.rep_ratio=get_parameter<double>(section,"rep_ratio");
277     }
278     else{
279       //Setting properties for a specific resources
280       if(valid_key(section,"i",subsection))
281         p.i=get_parameter<int>(section,"i",subsection);
282
283       if(valid_key(section,"j",subsection))
284         p.j=get_parameter<int>(section,"j",subsection);
285
286       if(valid_key(section,"groups",subsection)){
287         p.groups=get_parameterset<int>(section,"groups",subsection);
288       }
289
290       if(valid_key(section,"threshold",subsection))
291         p.threshold=get_parameter<double>(section,"threshold",
              subsection);
292
293       if(valid_key(section,"trust_eval",subsection))
294         p.trust_eval=get_parameter<string>(section,"trust_eval",
              subsection);
295
296       if(valid_key(section,"rep_eval",subsection))
297         p.rep_eval=get_parameter<string>(section,"rep_eval",
              subsection);
298
299       if(valid_key(section,"direct_weight",subsection))
300         p.direct_weight=get_parameter<double>(section,"direct_weight"
              ,subsection);
301
302       if(valid_key(section,"smoothing",subsection))
303         p.smoothing=get_parameter<double>(section,"smoothing",
              subsection);
304
305       if(valid_key(section,"sigma",subsection))
306         p.sigma=get_parameter<double>(section,"sigma",subsection);
307
308       if(valid_key(section,"stranger",subsection))
309         p.stranger=get_parameter<double>(section,"stranger");
310
311       //Only used if discrete combination
312       if(p.trust_eval=="discrete")
313         if(valid_key(section,"discrete_ratio",subsection))
314           p.discrete_ratio=get_parameter<double>(section,"
                discrete_ratio",subsection);
315         else if(simOptions.rDefaults.trust_eval!="discrete")
316           throw string("discrete_ratio was not set in subsection "+
                subsection+ " (Might not be a default value to use
                either)" );
317       //Only used if beta
318       if(p.rep_eval=="beta")
```

```
319         if(valid_key(section,"forgetting",subsection))
320           p.forgetting=get_parameter<double>(section,"forgetting",
                  subsection);
321         else if(simOptions.rDefaults.rep_eval!="beta")
322           throw string("forgetting was not specified in subsection "+
                  subsection+" (Might not be a default value to use
                  either)");
323
324       //Only used if beta
325       if(p.rep_eval=="beta")
326         if(valid_key(section,"base_weight",subsection))
327           p.base_weight=get_parameter<double>(section,"base_weight",
                  subsection);
328         else if(simOptions.rDefaults.rep_eval!="beta")
329           throw string("base_weight was not specified in subsection "
                  +subsection+" (Might not be a default value to use
                  either)");
330
331       //Only used if discrete reputation
332       if(p.rep_eval=="discrete")
333         if(valid_key(section,"rep_ratio",subsection))
334           p.rep_ratio=get_parameter<double>(section,"rep_ratio",
                  subsection);
335         else if(simOptions.rDefaults.rep_eval!="discrete")
336           throw string("rep_ratio was not set in subsection "+
                  subsection+ " (Might not be a default value to use
                  either)" );
337
338     }
339   }
340   void Configuration::set_resources(const string& section){
341     if(section!="[Resources]")
342       throw string("Trying to set resources from section: " +section)
              ;
343
344     for(submap::iterator it=cfile[section].begin();it!=cfile[section
          ].end();it++)
345       if(it->first!="none"){
346         //Setting properties for <it->first>
347         properties p= simOptions.rDefaults;
348         set_properties(p,section,it->first);
349         if(!compare_properties(p,simOptions.rDefaults)){
350           //<id> has custom properties. Finding id
351           string rid = (it->first).substr((it->first).find("<",0)+1,
352                                           (it->first).find(">",0)-1);
353           //save the properties struct for resource rid
354           resources[rid]=p;
355         }
356       }
357   }
358   void Configuration::set_changes(const string& section ){
359     if(section!="[Resources]")
360       throw string("Trying to set resources from section: " +section)
              ;
361
```

```
362     string key ="time";
363     strvec tokenized;
364     stringstream ss;
365     submap::iterator subit;
366     strvec::iterator vit;
367     for(subit=cfile[section].begin();subit!=cfile[section].end();
            subit++)
368       if(subit->first!="none"){
369         /* Processing subsection <it->first>. Will check if the
370            key ("time") is there. If the key is found then i and j
371            is also mandatory. I and j are the change in behaviour. */
372         for(vit=subit->second.begin();vit!=subit->second.end();vit++)
373           if(trim(*vit).substr(0,key.size())==key){
374             //resource id
375             string rid=(subit->first).substr((subit->first).find("<"
                    ,0)+1,
376                                              (subit->first).find(">"
                                                   ,0)-1);
377             //Set up for tokenizing
378             strvec tokenized;
379             string::size_type pos = vit->find_first_of(';');
380             string tokenizeme= *vit;
381             if(pos != string::npos)
382               tokenizeme=vit->substr(0,pos);
383
384             //tokenizing
385             tokenize(tokenizeme,tokenized);
386
387             //tokenize iterator
388             strvec::iterator tokit;
389             /* Find the time of the change  */
390             tokit=find(tokenized.begin(),tokenized.end(),"time");
391             if(tokit==tokenized.end())
392               throw string("The␣key␣time␣was␣not␣found␣in␣the␣line:␣"
                      +tokenizeme);
393             /* The location tokit+1 must contain a unsigned int that
                    denotes
394                the time of the change. */
395
396             unsigned int changeTime;
397             ss<<get_next_token(tokenized,tokit);
398             if(!(ss>>changeTime))
399               throw string(*tokit+"␣is␣not␣of␣type␣unsigned␣int␣in␣
                      line␣"+tokenizeme);
400
401             //Clear stringstream
402             ss.clear();
403             ss.str("");
404
405             /* Find i and j values */
406             // i
407             int r_i;
408             tokit=find(tokenized.begin(),tokenized.end(),"i");
409             if(tokit!=tokenized.end()){
410               //the i key was found.
```

```
411               ss<<get_next_token(tokenized,tokit);
412               if(!(ss>>r_i))
413                 throw string("No valid i value was found in "+*tokit)
                        ;
414             }
415             else
416               throw string("Missing i value was in "+tokenizeme);
417             //Clear stringstream
418             ss.clear();
419             ss.str("");
420
421             // j
422             int r_j;
423             tokit=find(tokenized.begin(),tokenized.end(),"j");
424             if(tokit!=tokenized.end()){
425               //the j key was found.
426               ss<<get_next_token(tokenized,tokit);
427               if(!(ss>>r_j))
428                 throw string("No valid i value was found in "+*tokit)
                        ;
429             }
430             else
431               throw string("Missing j value was in "+tokenizeme);
432             //Clear stringstream
433             ss.clear();
434             ss.str("");
435
436             //Making change struct and add it to change queue
437             change* c=new change;
438             (*c).id=rid;
439             (*c).time=changeTime;
440             (*c).i=r_i;
441             (*c).j=r_j;
442
443             changes.push(c);
444             changetimes.insert((*c).time);
445
446             //Add pointer to listofchanges (used for summary)
447             listofchanges.push_back(c);
448           }
449       }
450   }
451
452   void Configuration::set_additions(const string& section){
453     if(section!="[Add]")
454       throw string("Trying to set additions from section: " +section)
            ;
455
456     stringstream ss;
457     submap::iterator sit;
458     strvec::iterator it;
459     for(sit=cfile[section].begin();sit!=cfile[section].end();sit++){
460       //for each subsection
461       for(it=sit->second.begin();it!=sit->second.end();it++){
```

```
462         /* Each *it should be a line containing information of a
                addition.
463            This line will be tokenized and then addition structs will
                    be created
464            and inserted into the addition queue.  */
465         strvec tokenized;
466         string::size_type pos = it->find_first_of(';');
467         string tokenizeme= *it;
468         if(pos != string::npos)
469           tokenizeme=it->substr(0,pos);
470
471         //tokenizing
472         tokenize(tokenizeme,tokenized);
473
474         strvec::iterator tokit;
475         /* Some keys must be present in the line (in the string
                vector
476            containing the tokenized line). The first mandatory key is
                    "time".
477            This key denotes when the addition will happen.  */
478         tokit=find(tokenized.begin(),tokenized.end(),"time");
479         if(tokit==tokenized.end())
480           throw string("The key time was not found in the line: "+
                   tokenizeme);
481         /* The location tokit+1 must contain a unsigned int that
                denotes
482            the time of the addition. */
483
484         unsigned int addTime;
485         ss<<get_next_token(tokenized,tokit);
486         if(!(ss>>addTime))
487           throw string("Expected "+*tokit+" to be of type unsigned
                   int when proccessing line "+tokenizeme);
488
489         //Clear stringstream
490         ss.clear();
491         ss.str("");
492         /* A mandatory value that also must be there is a valid
                resource id.
493            This is required simple to remind the user to keep track
                   of who joins
494            the simulation (for saving purposes). Here we will iterate
                    through
495            the strvec and se if a string begins with 'u' or 's'.*/
496         bool foundrid=false;
497         string rid;
498         for(tokit=tokenized.begin();tokit!=tokenized.end() && !
                foundrid;tokit++)
499           if(valid_id(*tokit)){
500             rid=*tokit;
501             foundrid=true;
502           }
503
504         //Making addition struct and adding to addqueue and adding
                time to addtimes
```

```
505         addition* a=new addition;
506         (*a).id=rid;
507         (*a).time=addTime;
508         additions.push(a);
509         addtimes.insert((*a).time);
510
511         /* Now the time of addition has been found and the id of the
512            resource. Now its time to check for optional values.
513            It is however recommended (purely for consistency of the
514            configuration file) to specify optional values in the
515            [Resource] section. The method here is just for shorthand.
516
517            The only optional values allowed are i & j. Meaning that
518            only the behavioural values can be specified.
519
520            REMEMBER : Any option values specified in the [Add]
                section
521            will override values from the [Resource] section. */
522
523         //Check for i value
524         bool specified_i=false;
525         int r_i;
526         tokit=find(tokenized.begin(),tokenized.end(),"i");
527         if(tokit!=tokenized.end()){
528           //the i key was found.
529           ss<<get_next_token(tokenized,tokit);
530           if(!(ss>>r_i))
531             throw string("No valid i value was found when in "+*tokit
                  );
532           specified_i=true;
533         }
534         //Clear stringstream
535         ss.clear();
536         ss.str("");
537
538         //Check for j value
539         bool specified_j=false;
540         int r_j=-1;
541         tokit=find(tokenized.begin(),tokenized.end(),"j");
542         if(tokit!=tokenized.end()){
543           //the i key was found.
544           ss<<get_next_token(tokenized,tokit);
545           if(!(ss>>r_j))
546             throw string("No valid i value was found when in "+*tokit
                  );
547           specified_j=true;
548         }
549         //Clear stringstream
550         ss.clear();
551         ss.str("");
552
553         /* Update the resources map if i and j was specified.
554            Here a warning will be printed if it overwrites
555            previously defined values.
556         */
```

```
557         if(specified_i || specified_j){
558           //i or j was specifed.
559           if(resources.find(rid)==resources.end()){
560             /*No values have previously been specified
561               Now we can just make a new properties struct
562               and insert. Base is still rDefaults  */
563             properties p =simOptions.rDefaults;
564             if(specified_i)
565               p.i=r_i;
566             if(specified_j)
567               p.j=r_j;
568
569             resources[rid]=p;
570           }
571           else{
572             /* properties struct already exist for resource rid.
573                We need to change it. (overwrite i and j)
574                Will print a warning if different from default values
575              */
576             if(r_i!=simOptions.rDefaults.i || r_j!=simOptions.
                   rDefaults.j)
577               if(resources[rid].i!=simOptions.rDefaults.i ||
578                  resources[rid].j!=simOptions.rDefaults.j)
579                 cout<<"Warning: Overwriting i and/or j values for "<<
                        rid<<endl;
580             if(specified_i)
581               resources[rid].i=r_i;
582             if(specified_j)
583               resources[rid].j=r_j;
584           }
585
586         }
587         //Add addition* to listofadds (when printing the summary)
588         listofadds.push_back(a);
589
590       }
591     }
592
593     //cout<<"set_additions ([Add]) success"<<endl;
594   }
595
596   void Configuration::set_saveoptions(){
597     /* This method sets the saveoptions. It will
598        look for 3 sections in the cfile. The sections
599        are [CtSave], [DtSave] and [RtSave], [PrSave].
600
601        IMPORTANT: If a section is missing the default action
602        is to save nothing.
603      */
604     bool ct=false,dt=false,rt=false, pr=false;
605     //Find default saving options
606     if(valid_key("[CtSave]","default")){
607       string save=get_parameter<string>("[CtSave]","default");
608       if(save=="yes"|| save=="true")
609         ct=true;
```

```
610     }
611     if(valid_key("[DtSave]","default")){
612       string save=get_parameter<string>("[DtSave]","default");
613       if(save=="yes"|| save=="true")
614         dt=true;
615     }
616     if(valid_key("[RtSave]","default")){
617       string save=get_parameter<string>("[RtSave]","default");
618       if(save=="yes"|| save=="true")
619         rt=true;
620     }
621       if(valid_key("[PrSave]","default")){
622       string save=get_parameter<string>("[PrSave]","default");
623       if(save=="yes"|| save=="true")
624         pr=true;
625     }
626     if(ct||dt||rt|pr)
627       cout<<"Something is saved by default"<<endl;
628     //Create saveOptions object
629     sOptions=new SaveOption(ct,dt,rt);
630     //scan for [CtSave],[DtSave] and [RtSave] sections
631     set_savetype("[CtSave]","ct");
632     set_savetype("[DtSave]","dt");
633     set_savetype("[RtSave]","rt");
634   }
635
636   void Configuration::set_savetype(const string& section,const string
        & type){
637     /* Helper function to set_saveoptions */
638     if(type!="ct" && type!="dt" && type!="rt" && type!="pr")
639       throw string("using set_savetype with invalid type: "+type);
640
641     string subsection="none";
642     string nokey="default";
643     if(cfile.find(section)!=cfile.end()){
644       //There is a [savetype] section.
645       for(strvec::iterator it=cfile[section][subsection].begin();
646           it!=cfile[section][subsection].end();it++)
647         if(trim(*it).substr(0,nokey.size())!=nokey){
648           /* *it might contain a string of a key (a resource id)
649              and a list of resource id's. These resource id's are
650              the resources that the resource with id = key will
651              save combined trust info on.
652           */
653           string::size_type pos = it->find_first_of(';');
654           string tokenizeme= *it;
655           if(pos != string::npos)
656             tokenizeme=it->substr(0,pos);
657
658           //tokenizing
659           strvec tokenized;
660           tokenize(tokenizeme,tokenized);
661
662           //id is the resource that will save information
663           string id;
```

```
664           if(valid_id(tokenized[0]))
665             id=tokenized[0];
666           else
667             throw string(id+" appears to be an invalid id key ( [
                  CtSave] )");
668
669           //Check that every token appears as a valid resource id
670           for(strvec::iterator tokit=tokenized.begin();tokit!=
                  tokenized.end();tokit++){
671             if(*tokit=="full"){
672               //Saves everything
673               if(type=="ct")
674                 sOptions->add_full_save(id,type);
675               else if(type=="dt")
676                 sOptions->add_full_save(id,type);
677               else if(type=="rt")
678                 sOptions->add_full_save(id,type);
679                else if(type=="pr")
680                 sOptions->add_full_save(id,type);
681               else
682                 throw string("Unknown type: "+type);
683             }
684             else if(!valid_id(*tokit))
685               throw string (*tokit+" appears invalid in "+id+"'s
                    ctsave list");
686             else if(id!=*tokit){
687               //Add *tokit at saveoptions.
688               //cout<<id<<" saves "<<*tokit<<" of type"<<type<<endl;
689               if(type=="ct")
690                 sOptions->save_ct(id,*tokit);
691               else if(type=="dt")
692                 sOptions->save_dt(id,*tokit);
693               else if(type=="rt")
694                 sOptions->save_rt(id,*tokit);
695                else if(type=="pr")
696                 sOptions->save_rt(id,*tokit);
697               else
698                 throw string("Unknown type: "+type);
699             }
700           }
701
702         }
703     }
704   }
705
706
707   void Configuration::set_mandatory_options(){
708     /* get and set the mandatory options.
709        These options are the bare minimum needed to
710        run a simulation.
711     */
712     try{
713       simOptions.users=get_parameter<unsigned int>("[Config]","users"
              );
```

```
714       simOptions.sites=get_parameter<unsigned int>("[Config]","sites"
              );
715       simOptions.time=get_parameter<unsigned int>("[Config]","time");
716       simOptions.mu=get_parameter<double>("[Config]","mu");
717       simOptions.normalmu=get_parameter<double>("[Config]","normalmu"
              );
718       simOptions.normalsigma=get_parameter<double>("[Config]","
              normalsigma");
719       simOptions.path=get_parameter<string>("[Config]","path");
720       //If there is no seed specifed at commandline seed
721       if(!commandlineseed){
722         //Check if there is a seed specified in cmap. If not
723         //current time will be used as seed.
724         if(valid_key("[Config]","seed"))
725           simOptions.seed=get_parameter<unsigned long int>("[Config]"
                  ,"seed");
726         else
727           simOptions.seed=time(0);
728       }
729       else if(valid_key("[Config]","seed")){
730         cout<<"Warning: the seed "<<simOptions.seed
731             <<" was specified at the commandline, but the config file
                  "
732             <<" suggests the seed "
733             <<get_parameter<unsigned long int>("[Config]","seed")<<
                  endl;
734       }
735       //Default behaviour for the resources
736       set_properties(simOptions.rDefaults);
737
738       //cout<<"set mandatory options success"<<endl;
739     }
740     catch(string e){
741       string exc;
742       exc="When trying to set the mandatory options";
743       exc+=" the following error occured\n"+e;
744       throw string(exc);
745     }
746
747   }
748   void Configuration::set_heuristic(){
749     customHeuris=false;
750
751     if(valid_key("[Heuristics]","experience")){
752       heuristic["experience"]=get_parameter<string>("[Heuristics]","
              experience");
753       cout<<"Info: Custom experience heuristics specified"<<endl;
754       customHeuris=true;
755     }
756     if(valid_key("[Heuristics]","oldvalue")){
757       heuristic["oldvalue"]=get_parameter<string>("[Heuristics]","
              oldvalue");
758       cout<<"Info: Custom oldvalue heuristics specified"<<endl;
759       customHeuris=true;
760     }
```

```
761   if(valid_key("[Heuristics]","newvalue")){
762     heuristic["newvalue"]=get_parameter<string>("[Heuristics]","
          newvalue");
763     cout<<"Info: Custom newvalue heuristics specified"<<endl;
764     customHeuris=true;
765   }
766
767 }
768
769
770 void Configuration::tokenize(const string& str,
771                              vector<string>& tokens,
772                              const string& delimiters){
773   // Skip delimiters at beginning.
774   string::size_type lastPos = str.find_first_not_of(delimiters, 0);
775   // Find first "non-delimiter".
776   string::size_type pos     = str.find_first_of(delimiters, lastPos
        );
777
778   while (string::npos != pos || string::npos != lastPos){
779     // Found a token, add it to the vector.
780     tokens.push_back(str.substr(lastPos, pos - lastPos));
781     // Skip delimiters.  Note the "not_of"
782     lastPos = str.find_first_not_of(delimiters, pos);
783     // Find next "non-delimiter"
784     pos = str.find_first_of(delimiters, lastPos);
785   }
786 }
787
788 string Configuration::trim(string& s,const string& drop)
789 {
790   string r=s.erase(s.find_last_not_of(drop)+1);
791   return r.erase(0,r.find_first_not_of(drop));
792 }
793 bool Configuration::compare_properties(const properties& p1,const
      properties& p2){
794   if(p1.i!=p2.i)
795     return false;
796   if(p1.j!=p2.j)
797     return false;
798   if(p1.threshold!=p2.threshold)
799     return false;
800   if(p1.trust_eval!=p2.trust_eval)
801     return false;
802   if(p1.rep_eval!=p2.rep_eval)
803     return false;
804   if(p1.smoothing!=p2.smoothing)
805     return false;
806   if(p1.sigma!=p2.sigma)
807     return false;
808   if(p1.stranger!=p2.stranger)
809     return false;
810   if(p1.trust_eval=="discrete")
811     if(p1.discrete_ratio!=p2.discrete_ratio)
812       return false;
```

```
813   if(p1.rep_eval==”beta”)
814     if(p1.forgetting!=p2.forgetting)
815       return false;
816     else if(p1.base_weight!=p2.base_weight)
817       return false;
818   if(p1.rep_eval==”discrete”)
819     if(p1.rep_ratio!=p2.rep_ratio)
820       return false;
821   if(p1.groups!=p2.groups)
822     return false;
823
824   //Properties seem equal
825   return true;
826 }
827
828 bool Configuration::compare_economy(const economy& e1,const economy
      & e2){
829   if(e1.markup!=e2.markup)
830     return false;
831   if(e1.depreciation!=e2.depreciation)
832     return false;
833   if(e1.discount!=e2.discount)
834     return false;
835   if(e1.hardwarecost!=e2.hardwarecost)
836     return false;
837   if(e1.protectionlevel!=e2.protectionlevel)
838     return false;
839
840   //Properties seem equal
841   return true;
842 }
843
844
845 strvec Configuration::find_and_tokenize(const string& section,
846                                         const string& key,
847                                         const string& subsection,
848                                         const string& delim){
849   //String vector for the tokenized result
850   strvec result;
851   //Iterators
852   strvec::iterator it;
853   //Flag indicated if key is found
854   bool foundKey= false;
855
856   //Check if section is defined in cfile map
857   if(cfile.find(section)==cfile.end())
858     throw string(section + ”␣section␣was␣not␣found␣in␣config␣file”)
          ;
859
860   //Check if subsection is defined in cfile map
861   if(cfile[section].find(subsection)==cfile[section].end())
862     throw string(subsection + ”␣subsection␣was␣not␣found␣in␣config␣
          file”);
863
864   //Find key in the string vector at cfile[section][subsection]
```

```
865     it=cfile[section][subsection].begin();
866     while(!foundKey && it!=cfile[section][subsection].end()){
867       // *it must start with key
868       if(trim(*it).substr(0,key.size())==key){
869         /* Found keyword. Tokenizing the line. Only the line
870            upto the the first comment, denoted by ; is parsed. */
871         string::size_type pos = it->find_first_of(';');
872         string tokenizeme= *it;
873         if(pos != string::npos)
874           tokenizeme=it->substr(0,pos);
875
876         //tokenizing
877         tokenize(tokenizeme,result,delim);
878         //Indicating that the key was found
879         foundKey=true;
880       }
881       it++;
882     }
883     if(!foundKey)
884       throw string(key+" was not found in section "+section);
885
886     //Return tokenized string as a vector of strings
887     return result;
888
889   }
890   properties* Configuration::get_properties(const string& id){
891     if(resources.find(id)==resources.end())
892       return &simOptions.rDefaults;
893     else
894       return &resources[id];
895   }
896
897   economy* Configuration::get_economy(const string& id){
                //const string& id){
898     if(economyressources.find(id)!=economyressources.end())
899      return &economyressources[id];
900       else
901     return &ep;
902
903   }
904
905
906
907   void Configuration::set_bank(){
908           double startmoney=0.0;
909
910
911           for(submap::iterator it=cfile["[Bank]"].begin();it!=cfile["
                  [Bank]"].end();it++)
912           {
913           if(it->first!="none")
914                   {
915                   string bid = (it->first).substr((it->first).find("<
                        ",0)+1,
```

```
916                                                    (it->first).find(">
                                                           ",0)-1);
917
918                 startmoney=get_parameter<double>("[Bank]","
                        initialmoney",(it->first));
919
920                 nationaltreasure.Set_Accounts(bid, startmoney,false
                        );
921
922                 }
923
924         }
925
926         startmoney=get_parameter<double>("[Bank]","initialmoney");

              nationaltreasure.Set_init_money(startmoney);
927         cout << "BANK_OPRETTET" << endl;
928
929
930
931
932 }
933
934
935 void Configuration::putsalaryonaccount(string id, double amount)
936 {
937
938 nationaltreasure.PutSalaryOnAccount(id,amount);
939
940 }
941
942 bool Configuration::hascredit(string userid){
943 if(nationaltreasure.Has_Credit(userid))
944 return true;
945 else
946 return false;
947 }
948
949 saveOptions* Configuration::get_save_options(const string& id1,
950                                              const string& id2){
951   return sOptions->get_save_options(id1,id2);
952 }
953
954 strset* Configuration::get_rep_group(const string& rId){
955   //Check valid rId
956   if(rId[0]!='u' && rId[0]!='s')
957     throw string("Tried to get a reputation group for "+rId);
958
959   strset* res= new strset();
960
961   stringstream ss;
962
963   //get rId's number
964   int myNr;
965   ss<<rId.substr(1,rId.length()-1);
```

```
966     if(!(ss>>myNr))
967       throw string("Invalid resource id "+rId);
968
969     //Clear stringstream
970     ss.clear();
971     ss.str("");
972
973     //number of reputation group members
974     int numberofrefs=4;
975     if(valid_key("[Config]","numberofrefs"))
976       numberofrefs=get_parameter<int>("[Config]","numberofrefs");
977
978     //find max amount of resources
979     int max;
980     if(rId[0]=='u')
981       max=simOptions.users-1;
982     else
983       max=simOptions.sites-1;
984
985     //counter
986     int c=myNr;
987     vector<int> nrs;
988     if(c==numberofrefs)
989       while(nrs.size()<numberofrefs && nrs.size()<=max){
990         c--;
991         if(c!=myNr)
992           nrs.push_back(c);
993       }
994     else if(c>numberofrefs)
995       while(nrs.size()<numberofrefs && nrs.size()<=max){
996         c--;
997         if(c!=myNr)
998           nrs.push_back(c);
999     }
1000    else{
1001      for(int i=0;i<=numberofrefs && i<=max ;i++)
1002        if(i!=myNr)
1003          nrs.push_back(i);
1004    }
1005
1006    //create id's and insert
1007    for(int i=0;i<nrs.size();i++){
1008      ss<<rId[0]<<nrs[i];
1009      res->insert(ss.str());
1010
1011      ss.clear();
1012      ss.str("");
1013    }
1014    //print
1015    //cout<<"For "<<rId<<" - ";
1016    //copy(res->begin(), res->end(), ostream_iterator<string>(cout, "
           "));
1017    //cout<<endl;
1018    return res;
1019
```

```
1020 }
1021
1022 int Configuration::compare_groups(const string& id1,const string&
       id2){
1023   //if(1)
1024   //return 1;
1025
1026   if(resources.find(id1)==resources.end() && resources.find(id2)==
         resources.end())
1027     return simOptions.rDefaults.groups.size();
1028   else{
1029     intset* g1;
1030     intset* g2;
1031     if(resources.find(id1)!=resources.end())
1032       g1=&resources[id1].groups;
1033     else
1034       g1=&simOptions.rDefaults.groups;
1035     if(resources.find(id2)!=resources.end())
1036       g2=&resources[id2].groups;
1037     else
1038       g2=&simOptions.rDefaults.groups;
1039     //make the set intersection for find the number of share groups
1040     intset r;
1041     set_intersection(g1->begin(),g1->end(),
1042                      g2->begin(),g2->end(),
1043                      inserter(r,r.begin()));
1044
1045     //Comparing
1046     return r.size();
1047   }
1048 }
1049 double Configuration::parse_exp(string e){
1050   string exp=e;
1051   //The expression has to end with a newline
1052   exp+="\n";
1053   //make char*
1054   char* expression=new char[exp.size()+1];
1055   //copy chars to expression
1056   strcpy(expression,exp.c_str());
1057   double result= parse_this(expression);
1058   //delete space allocate to expression
1059   delete[] expression;
1060   //Check if the parsing went ok.
1061   if(error_flag()!=0)
1062     throw string("Error␣parsing␣expression:" +e);
1063   return result;
1064 }
1065 bool Configuration::more_additions(unsigned int ctime){
1066   if(additions.empty())
1067     return false;
1068   if(additions.top()->time!=ctime)
1069     return false;
1070   else
1071     return true;
1072 }
```

```
1073
1074  addition* Configuration::get_next_add(){
1075    if(additions.empty())
1076      throw string("Asked for one more addition but none existed in
          queue");
1077
1078    addition* a;
1079    a=additions.top();
1080    additions.pop();
1081    //increment users or sites counter
1082    if((*a).id[0]=='u')
1083      simOptions.users++;
1084    else if((*a).id[0]=='s')
1085      simOptions.sites++;
1086
1087    return a;
1088  }
1089
1090  bool Configuration::more_changes(unsigned int ctime){
1091    if(changes.empty())
1092      return false;
1093    if(changes.top()->time!=ctime)
1094      return false;
1095    else
1096      return true;
1097  }
1098
1099  change* Configuration::get_next_change(){
1100    if(changes.empty())
1101      throw string("Asked for one more addition but none existed in
          queue");
1102
1103    change* c;
1104    c=changes.top();
1105    changes.pop();
1106    return c;
1107  }
1108
1109
1110  /* ---------- Test functions ---------- */
1111
1112
1113  void Configuration::print_config(){
1114    cmap::iterator it;
1115    submap::iterator subit;
1116    strvec::iterator vit;
1117    for(it=cfile.begin();it!=cfile.end();it++)
1118      for(subit=(it->second).begin();subit!=(it->second).end();subit
            ++){
1119        cout<<"printing section: "<<it->first
1120            <<" and subsection "<<subit->first<<endl;
1121        for(vit=(subit->second).begin();vit!=(subit->second).end();
              vit++)
1122          cout<<*vit<<endl;
1123      }
```

```
1124 }
1125
1126 void Configuration::print_properties(const properties* p){
1127
1128   cout<<"printing properties"<<endl;
1129   cout<<"i="<<p->i<<endl;
1130   cout<<"j="<<p->j<<endl;
1131   cout<<"groups=";
1132   copy(p->groups.begin(), p->groups.end(), ostream_iterator<int>(
          cout, " "));
1133   cout<<endl;
1134   cout<<"threshold="<<p->threshold<<endl;
1135   cout<<"trust_eval="<<p->trust_eval<<endl;
1136   cout<<"rep_eval="<<p->rep_eval<<endl;
1137   cout<<"smoothing="<<p->smoothing<<endl;
1138   cout<<"sigma="<<p->sigma<<endl;
1139   cout<<"stranger="<<p->stranger<<endl;
1140   if(p->trust_eval=="discrete")
1141     cout<<"discrete_ratio="<<p->discrete_ratio<<endl;
1142   if(p->rep_eval=="beta"){
1143     cout<<"forgetting="<<p->forgetting<<endl;
1144     cout<<"base_weight"<<p->base_weight<<endl;
1145   }
1146   if(p->rep_eval=="discrete")
1147     cout<<"rep_ratio="<<p->rep_ratio<<endl;
1148 }
1149
1150 void Configuration::print_resources(){
1151   cout<<"Default properties:"<<endl;
1152   print_properties(&simOptions.rDefaults);
1153   for(romap::iterator it=resources.begin();it!=resources.end();it
          ++){
1154     cout<<"resource "<<it->first<<" has the properties:"<<endl;
1155     print_properties(&it->second);
1156   }
1157 }
1158
1159 void Configuration::print_strvec(const strvec& v){
1160   cout<<"Content of strvec:";
1161   copy(v.begin(), v.end(), ostream_iterator<string>(cout, ","));
1162   cout<<endl;
1163 }
1164
1165 void Configuration::print_difference(const string& id,const
        properties* p){
1166   if(!compare_properties(*p,simOptions.rDefaults)){
1167     cout<<id<<" has special properties"<<endl;
1168     if(p->i!=simOptions.rDefaults.i)
1169       cout<<"i is "<<p->i<<" and the default is "
1170           <<simOptions.rDefaults.i<<endl;
1171     if(p->j!=simOptions.rDefaults.j)
1172       cout<<"j is "<<p->j<<" and the default is "
1173           <<simOptions.rDefaults.j<<endl;
1174     if(p->threshold!=simOptions.rDefaults.threshold)
1175       cout<<"threshold is "<<p->threshold<<" and the default is "
```

```
1176            <<simOptions.rDefaults.threshold<<endl;
1177        if(p->trust_eval!=simOptions.rDefaults.trust_eval)
1178          cout<<"trust_eval is "<<p->trust_eval<<" and the default is "
1179              <<simOptions.rDefaults.trust_eval<<endl;
1180        if(p->rep_eval!=simOptions.rDefaults.rep_eval)
1181          cout<<"rep_eval is "<<p->rep_eval<<" and the default is "
1182              <<simOptions.rDefaults.rep_eval<<endl;
1183        if(p->smoothing!=simOptions.rDefaults.smoothing)
1184          cout<<"smoothing is "<<p->smoothing<<" and the default is "
1185              <<simOptions.rDefaults.smoothing<<endl;
1186        if(p->sigma!=simOptions.rDefaults.sigma)
1187          cout<<"sigma is "<<p->sigma<<" and the default is "
1188              <<simOptions.rDefaults.sigma<<endl;
1189        if(p->stranger!=simOptions.rDefaults.stranger)
1190          cout<<"stranger is "<<p->smoothing<<" and the default is "
1191              <<simOptions.rDefaults.smoothing<<endl;
1192        if(p->trust_eval=="discrete")
1193          if(simOptions.rDefaults.trust_eval=="discrete"){
1194            if(p->discrete_ratio!=simOptions.rDefaults.discrete_ratio)
1195              cout<<"discrete_ratio is "<<p->discrete_ratio<<" and the
                      default is "
1196                  <<simOptions.rDefaults.discrete_ratio<<endl;
1197          }
1198          else
1199            cout<<"discrete_ratio is "<<p->discrete_ratio
1200                <<" and there is no default"<<endl;
1201
1202        if(p->rep_eval=="beta")
1203          if(simOptions.rDefaults.rep_eval=="beta"){
1204            if(p->forgetting!=simOptions.rDefaults.forgetting)
1205              cout<<"forgetting is "<<p->forgetting<<" and the default
                      is "
1206                  <<simOptions.rDefaults.forgetting<<endl;
1207          }
1208          else
1209            cout<<"forgetting is "<<p->forgetting
1210                <<" and there is no default"<<endl;
1211
1212        if(p->rep_eval=="beta")
1213          if(simOptions.rDefaults.rep_eval=="beta"){
1214            if(p->base_weight!=simOptions.rDefaults.base_weight)
1215              cout<<"forgetting is "<<p->base_weight<<" and the default
                      is "
1216                  <<simOptions.rDefaults.base_weight<<endl;
1217          }
1218          else
1219            cout<<"base_weight is "<<p->base_weight
1220                <<" and there is no default"<<endl;
1221
1222        if(p->rep_eval=="discrete")
1223          if(simOptions.rDefaults.rep_eval=="discrete"){
1224            if(p->rep_ratio!=simOptions.rDefaults.rep_ratio)
1225              cout<<"rep_ratio is "<<p->rep_ratio<<" and the default is
                      "
1226                  <<simOptions.rDefaults.rep_ratio<<endl;
```

```
1227        }
1228        else
1229          cout<<"rep_ratio is "<<p->rep_ratio
1230              <<" and there is no default"<<endl;
1231
1232      //compare groups
1233      if(p->groups!=simOptions.rDefaults.groups){
1234        cout<<"Groups are: ";
1235        copy(p->groups.begin(), p->groups.end(),
1236             ostream_iterator<int>(cout, " "));
1237        cout<<endl;
1238        cout<<"Default groups are: ";
1239        copy(simOptions.rDefaults.groups.begin(),
1240             simOptions.rDefaults.groups.end(),
1241             ostream_iterator<int>(cout, " "));
1242        cout<<endl;
1243      }
1244    }
1245  }
1246
1247  void Configuration::countcredit(bool check){
1248  //cout << " det virker" << endl;
1249  nationaltreasure.count_creditrejected(check);
1250  }
1251
1252  void Configuration::deductmoney(string sid,string uid,double amount
       )
1253   {
1254   nationaltreasure.Transfer_money_user_to_site(sid,uid,amount);
1255   }
1256
1257   void Configuration::setaccounts(string uid, double amount, bool
         initial)
1258   {
1259
1260          nationaltreasure.Set_Accounts(uid, amount,initial);
1261          //cout << "account " << uid << " sat" << endl;
1262
1263   }
1264
1265  void Configuration::print_accounts()
1266  {
1267  nationaltreasure.Print_accounts();
1268  }
1269  void Configuration::print_summary(){
1270    /* This function prints the result of the read config file
1271       It is highly recommened that you print this information
1272       and save as a log file. It prints alot of information
1273       but it is usefull when debugging and/or investigating
1274       strange simulation results.
1275    */
1276    cout<<"----------Simulation summary----------"<<endl;
1277    //print the main simulation setup
1278    cout<<"## Main simulation setup. ##"<<endl;
1279    mandatory* m=get_mandatory();
```

```
1280    cout<<"The simulation will start out with having "<<m->users
1281        <<" users and "<<m->sites<<" sites."<<endl;
1282    cout<<"The simulation time is set to "<<m->time<<
1283      " using a mu value of "<<m->mu<<endl;
1284    cout<<"for the poisson distribution (approximately "
1285        <<m->time/m->mu<<" job events)"<<endl;
1286    cout<<"The results will be saved to a subdirectory in "<<m->path
            <<endl;
1287    cout<<"The random number seed is "<<m->seed<<endl;
1288    cout<<"--"<<endl;
1289    cout<<"The default properties for resources are:"<<endl;
1290    cout<<"Combination for direct and reputation trust using the "
1291        <<m->rDefaults.trust_eval<<" approach"<<endl;
1292    cout<<"and reputation calculation using the "<<m->rDefaults.
            rep_eval
1293        <<" approach."<<endl;
1294    if(m->rDefaults.trust_eval=="discrete")
1295      cout<<"The discrete ratio between direct and reputation trust
              is "
1296          <<m->rDefaults.discrete_ratio<<endl;
1297    if(m->rDefaults.rep_eval=="beta"){
1298      cout<<"The forgetting factor for the beta reputation is "
1299          <<m->rDefaults.forgetting<<endl;
1300      cout<<"The base weight for the beta reputation is "
1301          <<m->rDefaults.base_weight<<endl;
1302    }
1303    if(m->rDefaults.rep_eval=="discrete")
1304      cout<<"The rep ratio between old and new reputation trust is "
1305          <<m->rDefaults.rep_ratio<<endl;
1306
1307    cout<<"Behaviour [i,j]=["<<m->rDefaults.i<<","<<m->rDefaults.j<<"
            ]"<<endl;
1308    cout<<"Groups=";
1309    copy(m->rDefaults.groups.begin(),
1310         m->rDefaults.groups.end(),
1311         ostream_iterator<int>(cout, " "));
1312    cout<<"(each number is a group)."<<endl;
1313    cout<<"The trust threshold is "<<m->rDefaults.threshold<<endl;
1314    cout<<"The direct weight (when calculating direct trust) is "
1315        <<m->rDefaults.direct_weight<<endl
1316        <<" using a smoothing factor of "<<m->rDefaults.smoothing<<
              endl
1317        <<"and sigma value "<<m->rDefaults.sigma<<endl;
1318    cout<<"The default trust value that is assigned to strangers is "
1319        <<m->rDefaults.stranger<<endl;
1320    cout<<"--"<<endl;
1321
1322    cout<<"## Main simulation setup end. ##"<<endl<<endl;
1323
1324    //list changes details (listofchanges)
1325    cout<<"## list of changes. ##"<<endl;
1326    changelist::iterator cit;
1327    for(cit=listofchanges.begin();cit!=listofchanges.end();cit++){
1328      properties* p=get_properties((*cit)->id);
```

```
1329        cout<<(*cit)->id<<" will at time="<<(*cit)->time<<" change [i,j
                ] from ["
1330             <<p->i<<","<<p->j<<"] to ["
1331             <<(*cit)->i<<","<<(*cit)->j<<"]"<<endl;
1332        print_difference((*cit)->id,p);
1333      }
1334      cout<<endl;
1335
1336      //print additions details (listofadds)
1337      cout<<"## list of additions. ##"<<endl;
1338      addlist::iterator ait;
1339      for(ait=listofadds.begin();ait!=listofadds.end();ait++){
1340        cout<<(*ait)->id<<" will be added at time="<<(*ait)->time<<endl
                ;
1341        properties* p=get_properties((*ait)->id);
1342        print_difference((*ait)->id,p);
1343      }
1344      cout<<endl;
1345
1346      //print saving options
1347      sOptions->print_save_info();
1348    };
```

Bibliography

[Audun Jøsang and Kinaeter, 2005] Audun Jøsang, E. G. and Kinaeter, M. (2005). Simplification and ananlysis of transitive trust networks. . *Web Intelligence and Agent Systems Journal*.

[Audun Jøsang and Faccer, 2003] Audun Jøsang, S. H. and Faccer, E. (2003). Simulating the effect of reputation systems on e-markets. Technical report, DSTC, Queensland University of Technology, Brisbane Qld 4001.

[Brinkløv, 2005] Brinkløv, M. H. (2005). Incremental trust in grid systems. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby". Supervised by Prof. Robin Sharp.

[Brinkløv and Sharp, 2006] Brinkløv, M. and Sharp, R. (2006). Incremental trust in grid computing. page 20.

[Buyya, 2002] Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne. Aviable at http://www.buyya.com/thesis/thesis.pdf.

[Buyya et al., 2001] Buyya, R., Abramson, D., and Giddy, J. (2001). A case for economy grid architecture for service oriented grid computing. In *IPDPS '01: Proceedings of the 10th Heterogeneous Computing Workshop HCW 2001 (Workshop 1)*, page 20083.1, Washington, DC, USA. IEEE Computer Society.

[Debreu, 1959] Debreu, G. (1959). *Theory of value*. Yale University Press.

[Eck, 2005] Eck, C. (2005). Draft lcg tier 0 and 1 resource draft.

[Frey, 1997] Frey, R. (1997). Derivative asset analysis in models with level-dependent and stochastic volatility. Discussion Paper Serie B 401, University of Bonn, Germany. available at http://ideas.repec.org/p/bon/bonsfb/401.html.

[Gambetta, 1990] Gambetta, D. (1990). Can we trust trust? *Trust: Making and Breaking Cooperative relations*, pages 213–238.

[J.D. Petrucelli and M.Chen, 1999] J.D. Petrucelli, B. N. and M.Chen (1999). *Applied Statistics for Engineers and Scientists*. Prentice-Hall, 1. edition.

[Jøsang and Ismail, 2002] Jøsang, A. and Ismail, R. (2002). The beta reputation system. In *e-Reality:Construting the e-Economy*. Avaible at http://security.dstc.edu.au/papers/JI2002-Bled.pdf.

[Knobloch and L.Robertson, 2005] Knobloch, J. and L.Robertson (2005). Lhc computing grid technical design report. Design version 1.04, CERN-LHCC.

[Mcknight and Chervany, 1996] Mcknight, D. and Chervany, N. (1996). The meanings of trust. Technical report MIRSC Working Paper Series.

[S. Holm-Rasmussen, 2005] S. Holm-Rasmussen, J. J. a. L. H. (2005). *Virksomhedsøkonomi til videregående uddannelse*. Systime academic.

[S.Ø. Jensen and Viegand, 2004] S.Ø. Jensen, C. S. Poulsen, P. T. and Viegand, J. (2004). Elforbrug i serverrum. Technical report, Elsparefonden Teknologisk institut.

[Talluri and Ryzin, 2005] Talluri, K. T. and Ryzin, G. J. V. (2005). *The theory and practice of revenue management*. Springer.

[Weatherford and Bodily, 1992] Weatherford, L. R. and Bodily, S. E. (1992). A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking and pricing. *Operations Research*, 40(5):831–844. Avaible at http://portal.acm.org/citation.cfm?id=146890&jmp=cit&dl=GUIDE&dl=ACM.

[Wolski et al., 2001] Wolski, R., Plank, J. S., Brevik, J., and Bryan, T. (2001). G-commerce: Market formulations controlling resource allocation on the computational grid. In *International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco. IEEEE. Avaible at http://www.cs.utk.edu/ plank/plank/papers/IPDPS-01.html.