Table of contents

1.1 INTRODUCTION - SGAS

1.1.1 Background

The SweGrid Accounting System [SGAS], started in August 2003, is, as implied, a coordinated accounting system for SweGrid clusters. It is designed to be non-intrusive on site deployment and has a uniform architecture, built on open standard-based grid protocols and existing toolkits. An alpha release is scheduled to be ready this month. As SGAS is still very much in early development (current version is 0.0.1a), the information below is mostly based on the available SGAS documentation.

1.1.2 Definitions

JARM	Job Account Reservation Manager (2.3)
LUTS	Logging and Usage Tracking Service (2.2)
OGSI	Open Grid Services Infrastructure
SGAS	SweGrid Accounting System

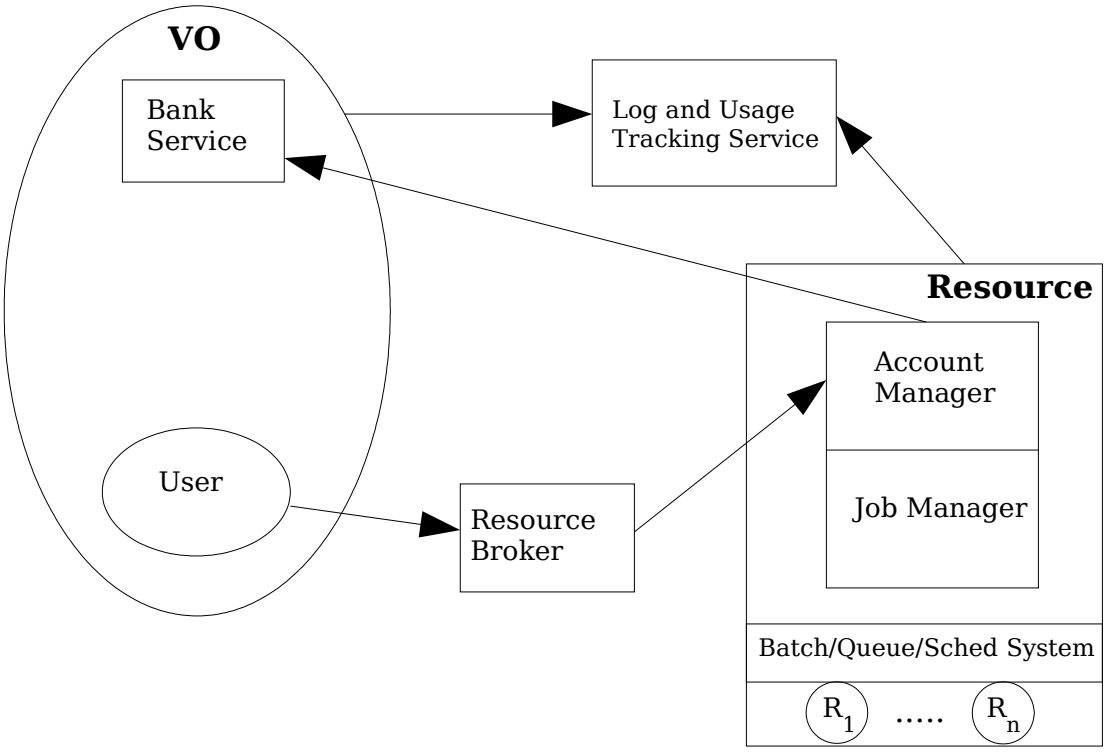
1.1.3 Functional Requirements
(as defined in architecture proposal)

From resource perspective
- Provide cost information to users
- Grant/deny users access based on account balance
- Get a guarantee that user funds will be available to pay for resource usage
- Track resource usage
- Charge the user for resource usage
- Get resource usage information for each submitted job

From user perspective
- Specify which VO/project should be charged for the resource usage
- Get the cost associated with using a resource
- Get account balance
- Get information on completed transactions
- Get usage information on completed jobs

1.2 ARCHITECTURE

The major components in the SGAS architecture consists of VO (User and Bank service), Resource broker, Resource (Account and Job Manager), Logging and Usage Tracking Service (LUTS).



1.2.1 Bank Service

The primary purpose of the SGAS Bank is to keep track of the resource usage for the individual projects/users. The Bank component is based on OGSI. All bank related information is stored in an XML database as Xindice is already incorporated into Globus Toolkit 3. Features of the banking component include bank account administration, chargeable account (multiple account holders can withdraw/create holds from the same account), transaction history, logging service (retrieve detailed info about account entries), security based on delegated user credentials, and soft state account holds.

In SweGrid the bank accounts are assigned to projects and not to the individual project members, even though the members have individual user accounts on the Grid. Within each VO, a user might participate in multiple projects (with different bank accounts). A user can also be a member of projects in several VOs.

The bank account supports a dynamic set of account holders that are modifiable at runtime. The account is only accessible by the account holders and each bank client is authenticated and authorized before access to the account is granted. Also, the account holders have different privileges, controlling their access rights.

Account holders are able to request time-limited reservations (holds) on account funds. A client may reset the expiry time of a hold after its creation. On hold expiry, the reserved amount is returned to the account.

A transaction history is maintained for each account and account holders are able to pose

non-trivial queries against the transaction log.

All transactions are performed using grid credits (unit less currency). While only node hours are accounted for in the initial SweGrid setting, additional types of resource usage can be incorporated into the accounting system (by using a function that converts the desired resource into grid credits).

1.2.2 Logging and Usage Tracking Service

LUTS [4] allow resources to log usage data and users to query the data in a consistent way. It is standardized around the GGF Usage Record XML format and uses standard Xpath-based Query languages. It includes batch publication, federated databases, and embedded local database support which together adds scalability and performance. When users submit their jobs, the job broker sends the job onward to an available resource for execution. When the job is done, the resource can push the usage record into (based on GGF UR standard) a local database (XML) or use the LUTS to publish the data in a service on a remote host. In both scenarios the users talk to a well-known LUTS service (that may be specified when the job is submitted) to query the log and usage data. The usage records can be pushed to the service in batch mode for better scalability, and there is hence no time guarantees on when the records will be available in LUTS.

1.2.3 Job Account Reservation Manager

JARM [5] provides an integration point for the SGAS system into various Grid middleware (i.e. NorduGrid) with minimal intrusion. It interacts with the job submission infrastructure and creates a hold (soft-state fund reservation), or a reservation of a certain amount of available funds. A hold will either be granted or denied depending on the availability of requested funds. The calculation of the expected allocation and the actual allocation required to run the job is customizable, and based on run time properties as well as the client job specification. The account to be charged can be specified in the job submission (RSL), otherwise JARM will attempt to find the targeted user in the bank. In both cases, the security proxy of the user submitting the job must be used when authenticating and authorizing with the bank account. Once the job has completed, JARM calculates the cost and charges it against the hold that was used to reserve the allocation. It is possible to extend the reservation, if the job has not completed within the estimated run time.

1.3 SCALABILITY

The SGAS architecture does not assume any centralized account handling entity. Each VO has an associated bank that handles the accounts of the VO users. Should the VO grow too large (thus affecting scalability), the VO can either be split up into two VOs with two separate bank services, or deploy a second bank. While in current practice, users are either mapped 1:1 or n:1 to local resource accounts, SGAS is aiming for a more dynamic creation of user accounts (so-called template accounts) at submission time.

1.4 REDUNDANCY AND RECOVERY

The system should run smoothly even after component failure. In the case of a bank being unreachable, the user should still be able to access a resource, but with less strict consistency guarantees. Replication will be included. The system will try to mask failures by proceeding as if nothing happened, resolving inconsistencies at a later time.

All information such as account state and transaction logs is stored in an XML database, to assure recoverability in the event of a server crash. It also allows users to query banking information.

1.5 SECURITY

Authentication is based on the Public Key Infrastructure (PKI), and SSL like handshakes over SOAP in compliance with WS-Security, WS-SecureConversation, XML-Encryption, XML-Signature, and GSS-APIs (to allow future integration of Kerberos) [2].

Authorization comprises two parts: policy management and policy enforcement. Management is done through a WDSL interface while policy enforcement is implemented as a callout from a message handler to deny or permit a request.

Accounting information is only exchanged between trusted entities and accessed by trusted entities.

WS-SecureConversation handshakes ensure that account reserves as well as the account charge operations are secured against replay attacks and snooping.

1.6 SOURCE CODE

The source code is available via AFS and the cell pdc.kth.se. A nightly checkout of the code can also be downloaded from http://www.pdc.kth.se/grid/sgas/source/

1.7 REFERENCES

[SGAS] SGAS Design documents
http://www.pdc.kth.se/grid/sgas/
[1] SweGrid Accounting System Architectural Proposal
http://www.pdc.kth.se/grid/sgas/docs/SGAS-0.1.3.pdf
[2] SweGrid Accounting System Security Design
http://www.pdc.kth.se/grid/sgas/docs/SGAS-SEC-DD-0.1.pdf
[3] SweGrid Accounting System Bank Design
http://www.pdc.kth.se/grid/sgas/docs/SGAS-BANK-DD-0.1.pdf
[4] SweGrid Logging and Usage Tracking Service Design
http://www.pdc.kth.se/grid/sgas/docs/SGAS-LUTS-DD-0.1.pdf
[5] Job Account Reservation Manager Design
http://www.pdc.kth.se/grid/sgas/docs/SGAS-JARM-DD-0.1.pdf

2.1 INTRODUCTION - GRIDBANK

2.1.1 Background

GridBank (GB)[1], developed in Australia by Gridbus[2], has been designed to meet new grid requirements as an accounting and payment handling system. This makes it possible for the resource owners to profit from the resources that are used by other people. Until now, those who had free resources had to make a local account for anyone who needed resource access. In the future, this will be impossible because it will be thousands of computers in the same grid. GridBank has a different infrastructure. Everyone has to register themselves on a central server; hence the resource owners don't have to make accounts for every resource user.

2.2 ARCHITECTURE

2.2.1 Overview

Maintenance and modifications is easy in this system. GB's server architecture consists of many modules which can be modified or replaced without disturbing the other modules. These are organized in three levels: the account layer, payment protocol layer and security layer. The account layer has the responsibility for databases and operations towards the accounts. The payment protocol layer defines payment schemes, message formats and communication protocols. It's easy to add payment systems in the future without modifying DB accounts and DB security modules. The security layer ensures that everyone is authorized and everything is authenticated.
GB's structure makes it easy to have many users. In other middleware technologies such as Globus, the local resource owner has to create and manage local accounts for each user. This will be impossible if thousands of computers are connected to the grid. When we use GB, a negotiator connects the buyer and the seller together.

2.2.2 Job submitting

The Grid Service Provider (GSP) and Grid Service Consumer (GSC) opens an account in the GB server. The user (GRC) submits a job with the requirements it has to the resources, costs, deadline and so on. This is the QoS-requirements. Grid Resource Broker (GRB) get this information and communicate with GSP's Grid Trading Service (GTS) to find a GSP which has the optimal resources available for this job. GRB negotiates with GTS and Grid Agent (GA) get the responsibility to make a GSP computer ready and download the job.
But before the job can start it has to be a local account on the GSP. The GSP's have some accounts in a pool which is not dedicated to any users. If the GSC get authorized, one of the accounts will be used by the GSP. GSCs certificate name connects temporary to the account to indicate the relationship between the account and the user.
GRB keeps contact with GridBank Payment Module (GBPM) to manage the use of available funds to help GSC avoid overspending. GBPM forwards payment details to GridBank Charging Module (GBCM).
GSP's Grid Resource Meter (GRM) gathers information about which resources that has

been used for processing the job when it's finished. GRM forwards this to GSP's GBCM which estimates the total cost. They find out which GSC who has the account, and then the relationship get removed and put the account back to the pool, and GBCM forwards the payment details together with Resource Usage Record (RUR) to GB server with a request to charge the user account. The total cost estimates out of CPU-time, network load and other metrics.

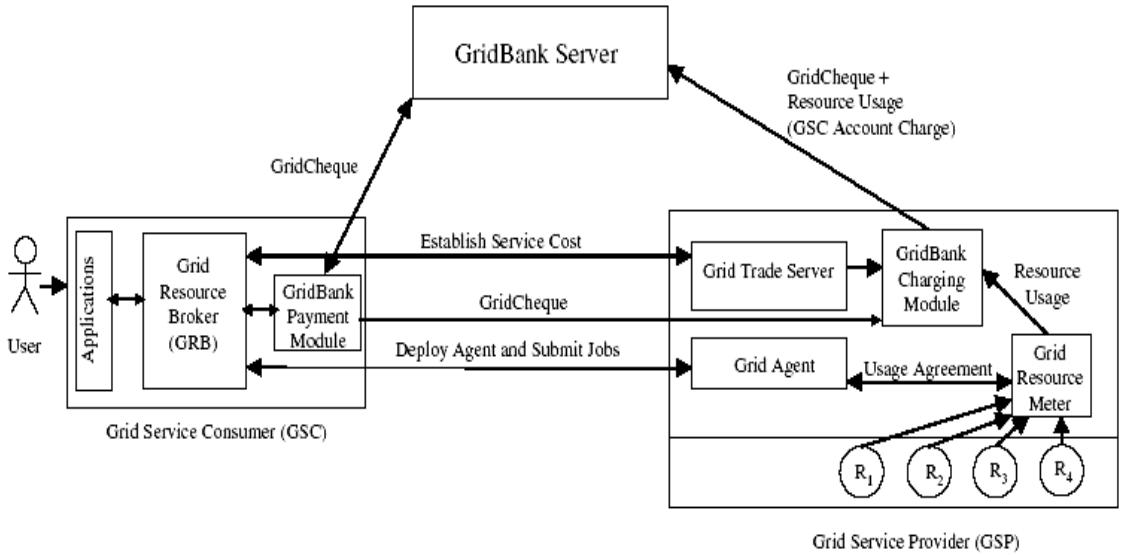


Figure from the GridBank documentation

When GRM shall pick up information about used resources, it will communicate with the local systems, e.g. cluster scheduler. It filters relevant information and forwards it to the conversion unit, which generates a RUR, which is a standard XML-document. When we use different protocols it can be necessary with different stats for the resource usage. R1 – R4 in the figure forwards individual stats over the resource usage GRM. GB saves this information so they have proof that the job has been done. RUR is forwarded to GBCM, which get information about from GTS to estimate the cost. Every chargeable item in GTS must have a corresponding item in RUR. This will avoid cheating in terms of getting it cheaper or more expensive.

2.3 USER INTERFACE

Before we can install GB, Globus toolkit with Globus server bundle 2 or later must be installed. We also need MySQL. The users can check the state of the account, and the administrators can add new administrators, open and close accounts and deposit and withdraw funds from the accounts. You can also use a grid monitor, a web-based GUI to control your account [3].

2.4 ECONOMICAL HANDLING

It's the owner of the resources (GSP) who decides the price. The users can choose what resource they want to use, but if it's too expensive, they will probably look elsewhere. The price will be regulated as economic systems in real life. If the demand is high, the price will be high; with low demand the price will be low. GSP can send a description of the resources to GB which estimates a price from similar jobs. This makes a reasonable price for the job.
To avoid overspending, we can use a budget. But if you pay with a credit card after the job is done, you can overspend. This is possible even if a credit limit is set. However, you can also pay prior to a job submission, which will limit the usage to the amount paid.

To be sure that a user can pay for the jobs when he uses grid check, GB can lock some funds before the check is validated. Every GSP gets a check with some locked funds. All payment handling happens through secure connections. It's the administrators who handle payments, change the credit limit, close accounts and other things regarding accounts and payments. They transfer real money, but here it's possible to automate things in the future with e.g. PayPal.
GB supports the use of grid credits as payment. Co-operative groups that use resources from each other can pay with credits instead of real money. It's important that everybody gives and takes resources or some few get all the credits and the rest of them have none.

2.5 REDUNDANCY AND RECOVERY

If we want to have a redundant system we need to take backup of the GB server. How often this is done, is up to the administrators. Though, it is very important since everything happens through GB. All the grid user accounts are there, so none can use the grid if the GB server is down. It can be necessary to have an extra server if the first one breaks down. In this system they use MySQL which makes data recoverable.

2.6 SECURITY

Security is important in every grid because we don't want anyone to break into the grid, nor do we want the grid users to abuse the system. It's the security protocol which authorizes and authenticates the users of the grid. The Globus toolkit is based on PKI which use x509v3 certificates. The users have to be listed in a central file to get access to the grid. If they aren't listed, they won't be able to submit any jobs. This method of authorization and authentication prevents DoS-attacks.
When you have logged in, you don't have to type the password anymore. The system uses the certificates to authenticate you when it has to. It's difficult for GSC and GSP to abuse the system because when a job is done, used resources have to match the requirements. Additionally, DB saves the RUR document to have proof of what has been used.

2.7 REFERENCES

[1] http://www.gridbus.org/papers/gridbank.pdf

[2] http://www.gridbus.org/

[3] http://www.gridbus.org/papers/gmonitor.pdf

3.1 INTRODUCTION – DGAS

3.1.1 Background

DGAS was developed for use in the European DataGrid project [1], which have been in the works in since 2001 and up to now, March 2004. With DGAS [2] they have tried to make an approach to a more economical brokering in data grids, in contradiction to the other brokering methods used a lot today. The idea is that users who contribute with resources should get something back, in this case they will earn GridCredits when they contribute with resources. These GridCredits can be spent later by using other resources on the grid. Initially, DGAS can only handle GridCredits, but for the future there is a possibility that these economic grids could have a way of transforming GridCredits into other currencies. Small companies that need computational power can then buy the resources they need and resource-providers can benefit from giving others access to the resources.

3.1.2 Definitions

EDG	European DataGrid, the project group that have been developing DGAS as a system for the larger EDG project.
Globus	Globus Toolkit, a tool for grid computing used in EDG and NorduGrid.
GridCredits	A currency used to describe the value of computational energy in the grid.
HLR	Home Location Register, a part of DGAS that contains the user database, their resources, and takes care of economic transactions between different HLRs.
IS	Information Service, a part of EDG that contains the information concerning the resource clusters and their attributes.
JDL	Job Description Language, a language used to define the requirements required by a job to run on a node.
PA	Price Authority, a part of DGAS that decides the price of the resources at different clusters.
RB	Resource Broker, a part of the WMS that collect information and decides which clusters are best suited for executing a specified job.
Resource	Computational energy needed for executing a job.
VO	Virtual Organization, an organization that administratively groups a set of users and/or resources.
WMS	Workload Management System, a system that receives a job and decides if it can be executed, where it should be executed and sends the job to the cluster.

3.2 ARCHITECTURE

3.2.1 HLR and PA

HLR – Home Location Register
The HLR is a database that contains the different user accounts on the grid, and their account information. It also takes care of the communication between different HLRs and credits/debits the different user/resources owners for the amount that their job uses. EDG has suggested a model where each VO has their own HLR for their own members.

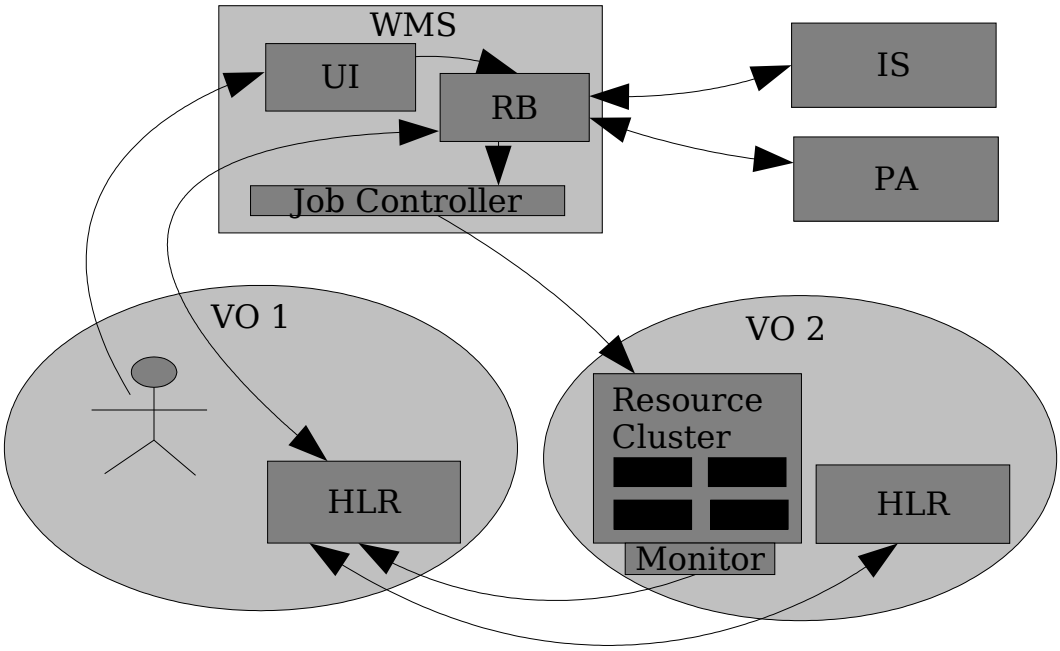
PA – Price Authority
The PA is a component in DGAS that decides the prices for all the different resources in the grid. The PA can assign different prices to all the different clusters, based on predetermined factors.

3.2.2 Job submitting

When a user wants a job executed, the job goes through different phases in the system. Here we will look very short on how the accounting system works in general.

At first, the user submits the job to a central WMS, using a language for describing the requirements from the system that will execute the job, JDL. The RB will then decide which places that are appropriate to execute the job on; the places that fulfill all the requirements are defined in the JDL message from the user. The information needed for taking this decision is gathered from the IS.
At this stage, the RB should gather an estimate of the price, but this is not implemented yet, due to the difficulty of estimating the running cost of the job. A "cost estimation algorithm" could then optimalize the cost of the job based the the prerequisites of the job.
The RB then checks the user's bank account, and reserves an amount of money if the user is able to pay for the job. If the user can't pay for the job, e.g. negative balance on the account, the RB will not send the job any further. If the user is able to pay, the job will be submitted to the cluster decided to do the job.

When the job is finished, the cluster will message the job submitter's HLR and the HLR will calculate the job cost based on the resources used. Then it will communicate with the resource owner's HLR and they transfer the GridCredits. Ideally, the system should be monitored while running the job. For instance, if the job is taking a lot longer time than expected, it could be suspended if the job owner runs out of GridCredits. This is not implemented now since there is no way of monitoring the resources used before the job is finished at the moment.



Even if NorduGrid and EDG base their systems on the same software at the lower level, the Globus toolkit, there are 2 major differences in their structure. NorduGrid uses a decentralized resource brokering system, where each resource broker resides in the job submitter's computer. In EDG, however, they have a central resource broker. The DGAS will have to be modified to handle this difference, but if you look at DGAS itself, it is very much like NorduGrid, decentralized with each VO having its own HLR. DGAS itself should be scalable to larger system with the “bank-branch” like structure, spreading the user accounts out over a large net rather than having them all on one place.
With all these HLRs spread out on the net, there must be some kind centralized place to list all the approved HLRs, so users can only use these approved HLRs.

3.3 REDUNDANCY AND RECOVERY AFTER FAILURE

None of the redundancy and recovery after failure issues is specially mentioned in the DGAS architecture, but the possibility of having only one PA could be a problem. This could, as mentioned in the DGAS architecture, be divided into several physical instances, that share a common dataset, making the system more redundant if one of the PAs should fail. Since there are several HLRs in the network, a job will only be dependant on the two HLRs involved in the job (The resource consumer's HLR and the resource provider's HLR). Recovery after system failure will have to be addressed in the database used for storage, eg. MySQL.

3.4 SECURITY

DGAS is based on the Globus Grid Security Infrastrcture (GSI) [3], a system that is already in use in NordGrid. GSI uses X.509 PKI certificates and communicates using

SSL. Authentication is mutual between hosts to ensure that both parts really are talking to the on they think they are talking to, preventing spoofing. Since DGAS bases its communication around XML messages, they use a unique identification of different XML messages to prevent repetitive registration of XML message and replay/playback of messages. A list of approved HLRs has to be created to ensure that no one can create his own fake HLR to obtain false GridCredits for use in the grid.

3.5 FUNCTIONALITY AND ECONOMIC MODELS

3.5.1 Metrics to measure used resources

A good system should have the capability to use many different metrics for measuring used resources while processing the job. DGAS only supports used CPU-time and used wall-clock-time now, with CPU-time mandatory and wall-clock-time optional. Other measurable metrics like used memory, network traffic, and used disk space should be implemented at some time, but is not currently supported by DGAS. The total cost of the job will then be calculated from these factors.

3.5.2 Job payment

The payment of a job is only done when the job is finished. If the job terminates or crashes the resource owner will not get paid. While this encourages resource owner to have stable systems, it doesn't really help them if the job owner terminates the job before it is finished. The resource owner will not get paid even if the job occupied his resources while running. A scheme with payment “as you go” here will solve this, with some kind of sensor telling the bank not to pay if the cluster goes down. (This is of course not implemented)

3.5.3 Economic models

The DGAS system does not focus on any specified economic model. The system has been designed with an eye on the fact that they do not really know what economic model they want to use, therefore they have used the system with PAs. With the PA, the way resources are priced can be decided later. Implementation of the price scheme can be implemented in a specified file on the PA server. At the moment the PA decides the resource price based on the different queues on the different clusters, but this algorithm can easily be changed into other pricing schemes.
By regulating prices on different clusters, you can partially do job brokering on an economical level, since user tend to lean towards the cheaper places if they know they get the same service. Regulating prices this way, and at the same time regulating grid load, is the basics behind “Economic Brokering”.

Currently, there is no way of reserving resources ahead of time. These will have to base themselves on the queues at the local clusters.

3.5.4 User account administration

Account administration is at a very primitive level at the moment. You can create accounts and give them GridCredits, but the user cannot control the account in any way, e.g. limit how much is spent each day. A user can overdraw his account, but this really depends on what kind of economic policy you want on the accounts. By enforcing a strict economic policy, users might be allowed to overdraw the account only if they already have positive funds and if the job is estimated to use less than what the user actually have in his account. A less strict policy could allow users to submit jobs regardless of their funds.

3.6. INFORMATION LOGGING

Since all the executed jobs go through the central WMS server in EDG, logging of everything is fairly simple. With NorduGrid you would have to decide if all the logged information about job execution should reside on each HLR or if it should be collected to a central server and stored there. For the information to be gathered on a centralized server, there has to be some system for either collecting it at given intervals or sending the information when the job is finished. This information is useful for analyzing data about the grid, and on local HLRs users would probably like to have a log of their activity on the grid.

3.7. SOURCE CODE

All the source code from the DGAS system and the rest of the EDG project is available at the EDG website [http://datagrid.in2p3.fr/cvsweb/workload/] and can be freely used and modified as long as you state that parts of the code come from the EDG project.

3.8. DGAS/EDG IN THE FUTURE

The EDG project is scheduled for final evaluation in March 2004. A new European grid-project is already moving, the EGEE [4]. According to their plans, they are supposed to take advantage of the experience gained in other grid projects and take these plans further. As they have stated, this will also include the EDG project. This might open a possibility for further development of DGAS, but there is no guarantee for this, they might even decide to ditch the whole system.

3.9. REFERENCES

[1] EU-DataGrid project website
http://eu-datagrid.web.cern.ch/eu-datagrid/

[2] DataGrid Accounting System Architecture
http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf

[3] Globus Toolkit Grid Security Infrastructure
http://www-unix.globus.org/security/

[4] EGEE: Enabling Grids for E-science in Europe
http://www.eu-egee.org

3 CONCLUSION

Out of all the three system we have looked at, Gridbank is the only complete and finished system. DGAS is “finished” in the sense that the EDG project is finished, but the software itself have many things that are proposed, but not yet implemented. SGAS is still in early development and as of now only nightly builds are availible (no announced releases).

While DGAS's distributed model with HLRs is very similar to NorduGrids structure, SGAS is also aiming for a dencentralized bank-structure with several banks, maybe one for each VO. GridBank hasnt migrated to a distributed structure yet, but as stated in their architecture, this might come in the future. Apart from this, Gridbank is a very throughly planned structure with a hard focus on economic structure and economic brokering. While DGAS and Gridbank have a more traditional approach to bank accounts, SGAS have a system which makes it possible to define accounts to different projects, and then assign users to these projects, thus making it possible to have many users on a single project, then all these users can access the resources assigned to the project. SGAS also seem to aim a little more against universities and similar institutes, with their model, they dont seem to have that much focus on the cost brokering and market values.

Payment of resources is also very throughly adressed in GridBank with different payment schemes, based on both gridcredits and real money.

Implementation will probably require some work and modification with Gridbank, while SGAS is alread being developed for use with the NorduGrid software. DGAS is using somewhat the same structure as SGAS, but it would still have to be modified to be usable in NorduGrid. Gridbank already support a large variety of different measurable metrics for calculating job cost, things like used network traffic, used memory, used cpu/wallclock-time have their own price, so they will be included in the final resource price. At the moment NorduGrid only provides information about used time, but in the future in a full scale production grid you may want to calculate the price with a background from many different factors.

We believe that GridBank, when properly adopted and modified, would pose as the overall best system for NorduGrid with commercial economical brokering in our mind. If the grid is more focused against scientific applications used by non-commercial institutes, SGAS also seem to be a good scheme, as an overall simpler system, with less focus on the economic brokering.