A Formal Model for Incorporating End-to-End Performance Guarantees in Grid Based Systems

Anish Desai Nishit Shah

anish_desai@dell.com nishit@cs.utexas.edu

Abstract: Grid systems are moving from the Academic and Research domain to a Commercial environment. Such an environment demands more than just 'Best Effort' delivery. In fact what is needed by the participants is a guarantee of performance. This requires addressing the issue of ***incentive*** *for resources to operate at higher reliability. In this paper we propose a System Engineering Approach to introduce guarantees within Grid Systems. Our goal is to design a model that could be easily incorporated in an OGSA Architecture based Grid. One important factor we keep in mind while designing this system is to make sure that the key component in our design has the ability to remain distributed. The key component viz., the* ***Guarantee and Quality of Service Engine (GQoSE)*** *takes care of resource discovery, reliability, pricing and most importantly guaranteed delivery. We shall describe various components to build such a system and discuss the subsequent concepts of reliability, pricing models and techniques to* ***guarantee delivery****.*

I Introduction

As we move from the academic and research environment to a commercialized one, there is a need to do better than just *'Best Effort'* delivery. Here we have to consider issues relating to pricing model and mechanisms to guarantee delivery. Related works [2,3,4] have proposed solutions to the issues relating to this field known as '***Grid Economics***'. We work towards extending this field to incorporate the feature of ***guarantees along with pricing*****.** Here we take a System Engineering approach and define the architectural components needed to provide *performance guarantees using Distributed Components called the GQoSE.*

Within the context of a Grid System, *Computation, Storage, Network & Applications* could be considered as primary resource offerings [5]. Secondary resource offerings could be any specialized units that are shared between peers in a Grid. The resource offerings are abstracted away from the individual nodes and absorbed into a "Common Grid Resource Pool" to be made available seamlessly on demand. The collective resource is managed through an abstraction layer that collects these resources into a Virtual Organization (VO)[6]. The VO binds the resources and presents them through a designated interface. In such a model the resource structure is **opaque** to the Grid Resource Consumer (GRC). Thus introducing a limitation that forces the GRC to rely on a "*best-effort*" paradigm. Such Grid systems therefore need to rely on Prediction models and complex resource allocation schemes to achieve an expected level of performance.

In this paper we discuss a Resource Management layer that is ***partially opaque****. In the proposed model we offer the GRC the option to choose the level of performance and enforce a grid protocol that responds with a guarantee for deliverance of said performance.* We begin by defining Performance in the context of a Grid system.

II Performance for Grid Systems

The concept of ***Quality of Service*** *(QoS)[12,13,14]* is closely related to ***Performance***. While QoS is defined in terms of metrics that are indicative of performance, performance itself is a broader concept than QoS since

extends to metrics such as viability, availability & assurance. Especially in case of commercial grid systems, the question of performance guarantees plays a decisive role in various business goals. Thus, we begin by identifying the performance components for each grid resource.

A few performance components that would be common to all four Resources [5] could be:

Availability, Security, Cost, Reliability

Some of the more resource specific components are:

1. **Computation**: *MTOPS*
2. **Storage**: *Throughput*, *Response Time*, *Capacity*
3. **Network**: *Throughput*, *Latency*, *Jitter*
4. **Application**: *Space/Time Complexity, Custom (e.g. : Accuracy, Efficiency, manageability)*

Each of the above mentioned metrics carry the well known industry definitions. While our model attempts to call out the major components that would define the performance index of a particular resource. We propose an extensible performance model that will scale as per the users definition.

Our proposal is not centered around the specific nature of performance; rather it focuses on the adaptability of the model as per the needs of the Grid Users. This ability allows the user to introduce the grid to the custom notion of performance. The performance metrics are laid out in the ***Resource Discovery Engine[23]***. This engine is responsible for collating and tracking the various resources. In order to accept a resource in the VO of the Grid; each resource must provide the performance signature that fits the boiler plate description of the Grid Performance Model as defined by the user. Once the individual resource entities agree to speak using the same performance *signatures [consistent protocol]*; the Resource Discovery Engine assimilates the resource into the VO. Later in the paper we shall give the protocol that will be followed for `join` and `leave` of resource.

In this section we have defined the notion of performance within grid systems; while keeping it scalable. The metrics listed in this section will suffice well for any basic interpretation of performance within a grid system. We now move on to better define the terms *reliability index* [7] and *guarantee factor*.

III Reliability Index

Each resource in the Grid pool of resources has an index associated with it, what we call the ***reliability index*** (`R`) [related to 7]. This index is a measure of the past performance of the resource. (`R`) varies from 0 to 1 in accordance with the commitment to perform according to the agreements of a transaction. Initially, when the resource has no previous record, the Grid Administrator can set the Reliability index as per choice. A recommended value for this would be 0.5.

The change in the Reliability index can be defined by the Grid Administrator. We suggest a ***'step-wise'*** approach to the changing of this index. E.g. in fig [i] the gradient is bell shaped. It can also be implemented as logarithmic, incremental, linear or flat.

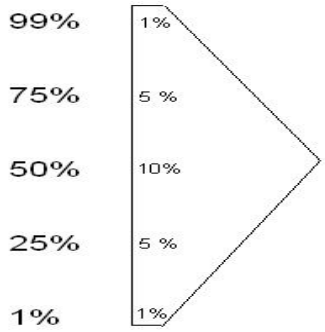

Fig i

IV The 'Guarantee' Factor

The reliability of individual components goes on to form the ***Guarantee Factor*** with which the Grid will deliver the results as per the negotiated performance. A greater Guarantee will necessitate more reliable components or redundancy in execution of the jobs.

We define guarantee as the degree of assurance with regards to a particular service level and performance factors as described in the previous section. However, the degree of assurance can only be quantified as a probability factor. Therefore we state **guarantee** as the '**probability of deliverance**'. There are several standardized reliability models which can be used to provide the guarantee [7]. E.g., given a *simple serial*

reliability model we can mathematically define the Guarantee as:

Probability of deliverance
= Π (Ri) over resources 1 to n

As one can quickly deduce, such a model of guarantee within a grid system relies on availability of each individual component. There can be *several other models which incorporate redundancy for assurance*.
On the other end of the spectrum for reliability would be a *complex system* of serial and parallel sub-systems. The individual resources could be identical or non-identical [7].

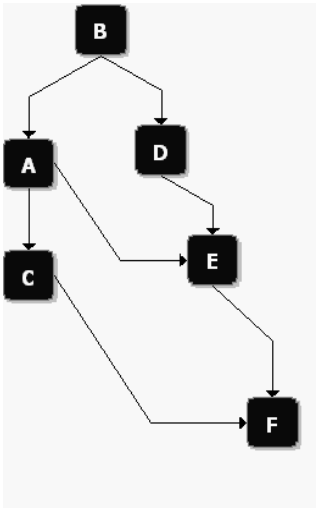

Fig ii

The overall guarantee in such a case can be calculated using *Decomposition, Event Space or Path Tracing* [7]. *E.g. in fig* [ii] *using key element 'A' the Guarantee Factor can be given as:*

$$G = P(s|A)P(A) + P(s|\bar{A})P(\bar{A})$$
$$= R_B R_F [1-(1-R_C)(1-R_B)] R_A + R_B R_D R_E R_F (1-R_A)$$

The discussion of Guarantee on such a model would not be complete without the factor of ***cost***. In any grid system, redundancy directly translates into cost. However, we expect that combined with a historical tracker and prediction based engine; the amortized cost of such a model shall be minimal. Cost nevertheless will continue to remain a dictating factor for negotiating the appropriate performance guarantee as we shall illustrate in section VII.

As mentioned above, in order to have greater reliability, the GQoSE may opt to have redundancy to meet a particular Guarantee. These components will be in one of two states: ***Active*** or ***Quiescent***. When a redundant resource is used as a 'backup' to the primary resource in the transaction, it is said to be in *Quiescent* state and is called a *cold swap. However,* if the computation is being performed on the redundant resource in parallel with the other resources, it is said to be in *Active state* and it is called a *Hot Swap.* It is important to note that the probability of success of a resource is different in these two states.

V Challenges in applying QoS Guarantee within a Grid Paradigm

Our guarantee model revolves around the fact that resources can be either 'immediately allocated' to meet the requested guarantee factor or the resource requester must await 'future allocation' until the grid can provide the requested assurance. While in the later scenario, where the requestor waits for the right resource, may seem easier, both conditions offer unique challenges within a grid environment. Namely:

1.**Resource Flux**: A Grids resources are in continuous flux. Resource must therefore be discovered/relinquished & accounted for dynamically from a complex set.

2.**Resource Flavors**: A Grid could be composed from a heterogeneous resource set. Even identical resources could offer themselves to the Virtual Organization of a Grid with different properties. We refer to these properties as flavors. These flavors are characteristics that cause the Virtual Organization to access/use the resource only in a particular manner. For example, a resource could offer itself only through particular security restrictions, or only for particular job roles and guarantee expectations.

The flavors are not necessarily restrictive. For example a computation unit that offers a high FLOPS/second could be reserved for long-running scientific calculations only. This way the smaller and less CPU intensive jobs would not cause the blocking of such a computational resource in turn, requiring less resource management and less resource switching when a truly large scientific job requests the resource.

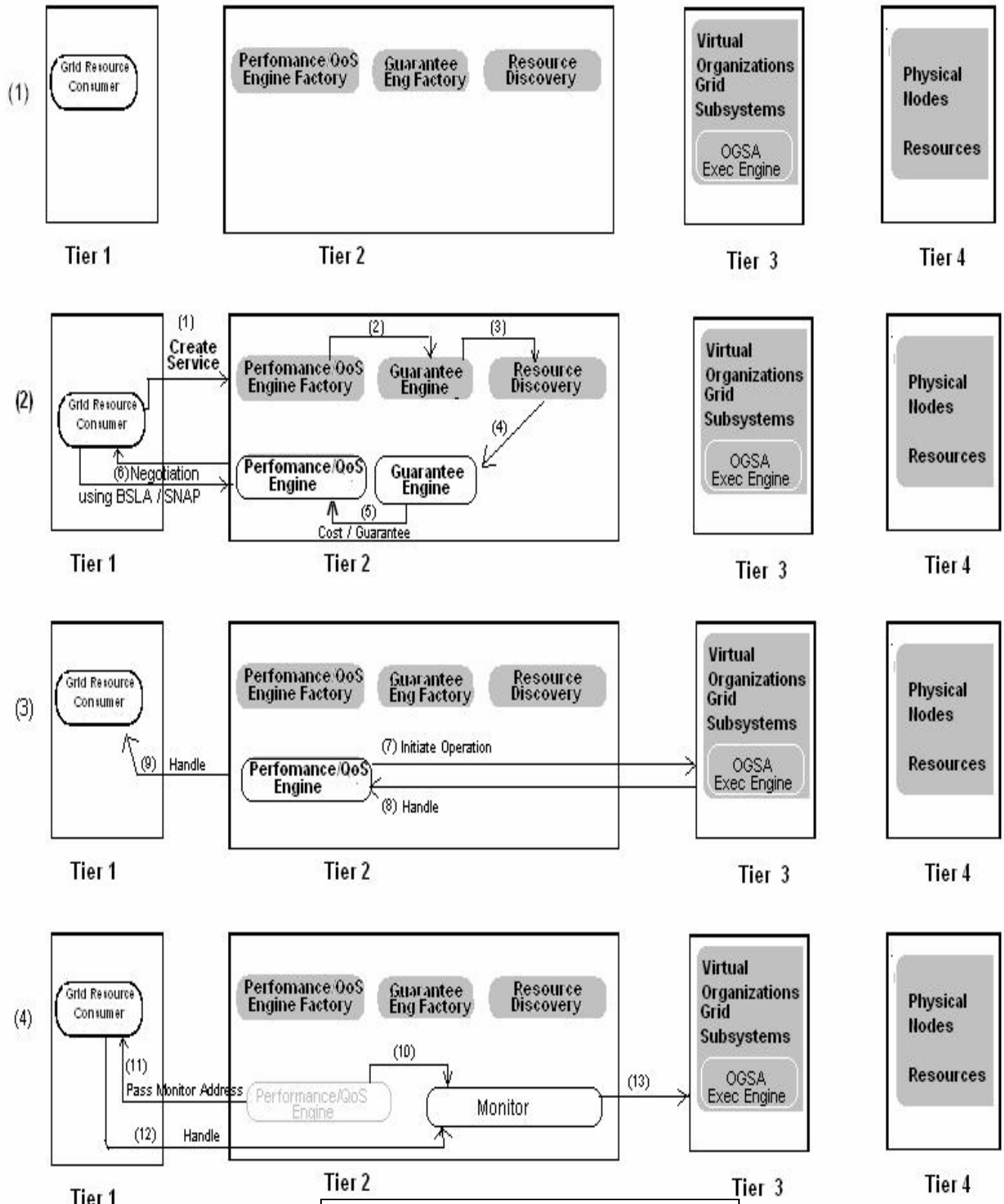

Fig iii. Example of Instantiation

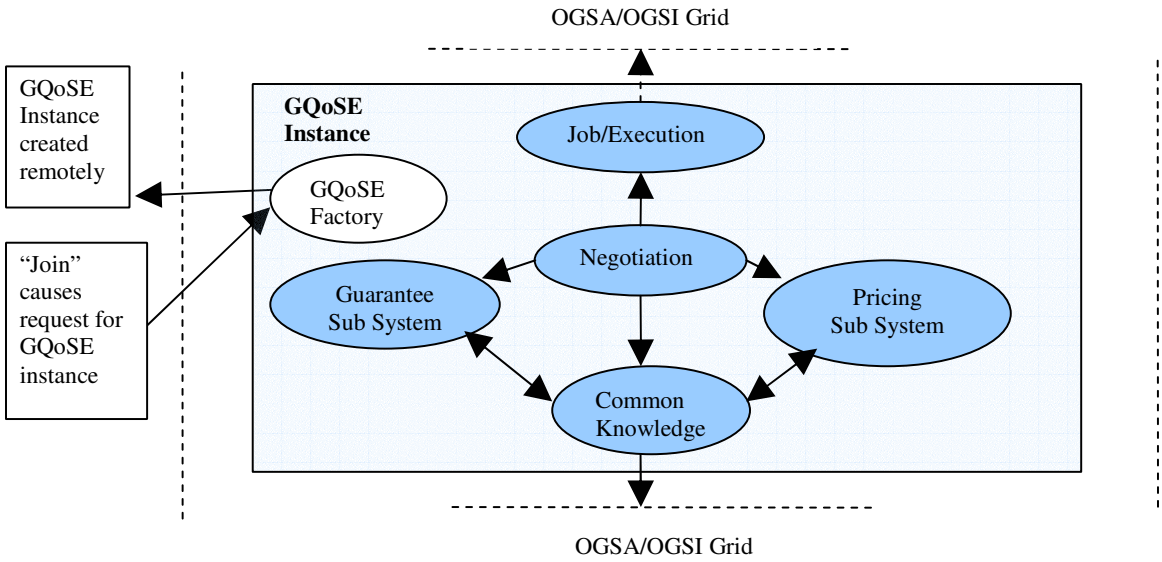

Fig iv. Components

We define two resource flavors, *internal* and *external* ***Internal Flavors*** are those characteristics which are *advertised* by the resource e.g. speed, latency, capacity. ***External Flavors*** are those characteristics which are determined by the Grid, and GQoSE, e.g. Reliability, Pricing. ***While modules such as GARA exist within system like Globus for the actual reservation of resource, they are not designed to cater for the consideration of resource flavors.***

3. **Resource Performance-Guarantee sub-system**: Grid resources do not offer any inherent guarantee to the reservation mechanisms. This means the roles of resource arbitration and management now must be extended through the "performance-guarantee sub system". Potentially building upon & extending, any default grid resource management sub-system that may exist such as GRAM[22] in case of Globus.

VI Architectural Components

1. **GQoSE Factory/Instantiation**
   - Triggered with every Join and registered with a Directory Service of the Grid Sub-system.
   - Responsible for creating a node-local instance of the GQoSE engine.
   - Binds the instance to the distributed GQoSE sub-system.
2. **GQoSE Common-Knowledge**
   - Serves as the ledger of all sub-components.
   - Modelled along the lines of distributed databases that can retrieve and store information over a distributed environment.
3. **Guarantee Sub-system**
   - Monitors and issues Reliability Index for each individual resource.
   - Determines the guarantee for a job request.
4. **Pricing Sub-system**
   - Serves as the distributed market maker for the shared resource.
   - Modelled as per the needs of the Pricing structure that is practised within the grid; it works in conjunction with the Guarantee sub-system to determine appropriate pricing.
5. **Negotiation Sub-system**
   - Primary responsibility is to arbitrate on behalf of the GRC for Price-Guarantee negotiations.
6. **Job Execution/Monitor Sub-system**
   - Responsible for the final committal of resources to a job along with the negotiated guarantee.
   - Monitors and ensures the guarantee for the job, while providing a feedback to the other sub-systems in case of any post negotiation discrepancies.

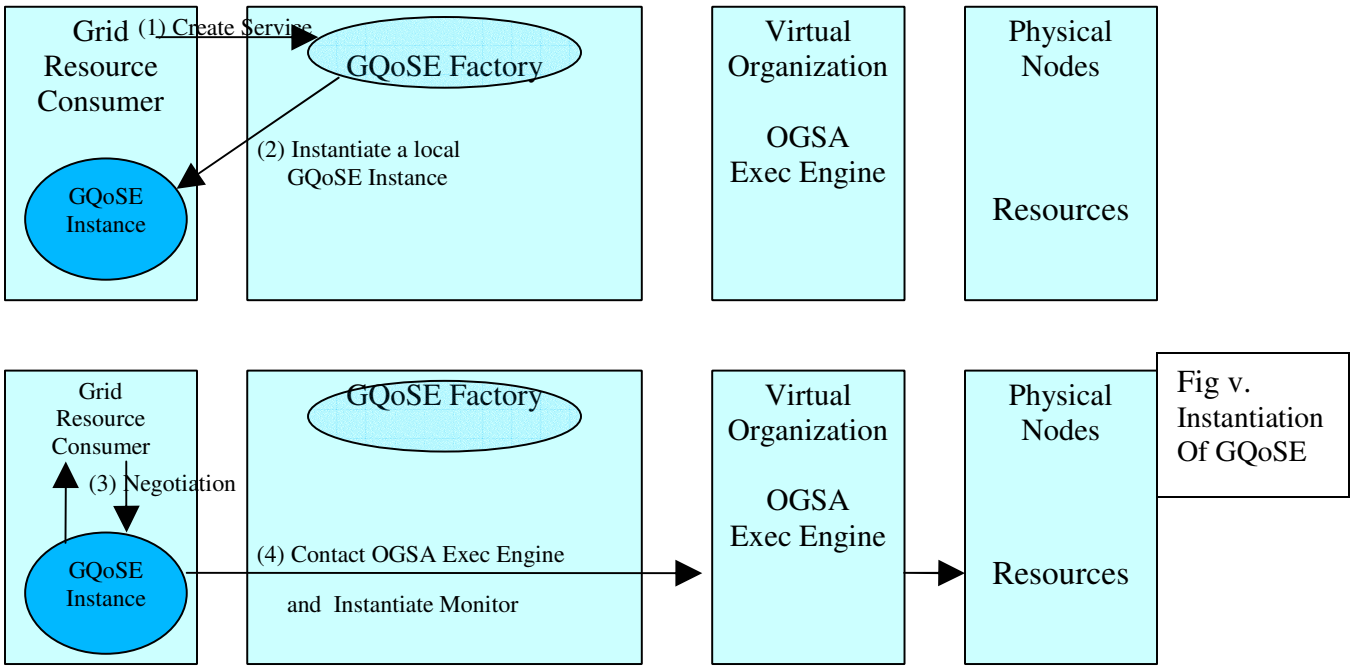

Fig v. Instantiation Of GQoSE

VII Protocol Specifics

In our model GRC requests are made in terms of desired level of guarantee rather than relying on Best Effort. There is inherent "*cost-based*" weighted grading of requests. In a truly commercialized grid this would be the appropriate assumption. The "cost" payment is not necessarily expressed in terms of any traditional currency amount. It may very well be a credit exchange system. In which the seller of the resource receives credits for the sharing within a Virtual Organization. The physical resource owner could later act as a GRC and utilize the accumulated credits to buy into resource guarantees for a job it needs to complete. Such a model offers incentive for sharing resources in return for benefits at a later point of need.

Resource Negotiation could be accomplished through protocols such as SNAP, which fits very well into a competitive space of Grid Resource Sharing. The SNAP protocol calls for continuous negotiation for the acquisition and binding of the resources. If in a later competing bid for the same resource from a different GRC is negotiated to be higher, it will allow for a pre-emptive expiration of the said binding. The protocol does allow the previous GRC to re-negotiate in order to retain the binding with the particular resource. Let us discuss the SNAP protocol a little further to understand some of the overall behaviors expected from the model.

SNAP[8] : Agreement State Transitions

The association of submitted activities with acquired resources in a utilization agreement is not always a protocol operation, but is nonetheless an observable *transition of the service state*, as depicted in Fig [vi]. In essence there are four states through which planning progresses:

S1: Submitted activities or acquired resources are not matched with each other.

S2: Submitted activities are matched with appropriate re-source acquisitions, and this grouping represents a re-source utilization agreement meant to resolve the activity.

S3: Acquired resources are being utilized for a submitted activity and can still be controlled or changed.

S4: The agreements have been resolved either by successful completion of the activity, or by expiration or cancellation of the agreements.

These states have a relationship to protocol messages in that client messages can only create or operate on active agreements. Inactive or finalized agreements can only occur as a result of client messages and the

passing of time. However, the figure simplifies the situation in that the state changes are not actually synchronized between submission and assignment agreements. Related submission or acquisition agreements may in fact move through these planning states at different rates, because activities may share the same resource capability but also may consume different capabilities sequentially.

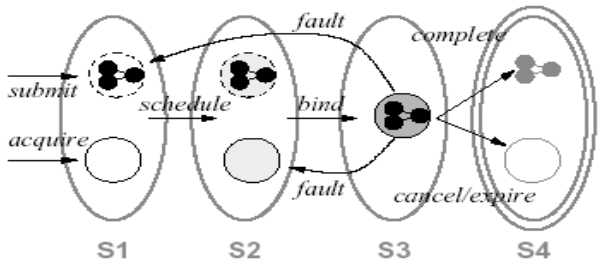

Fig vi [*Source: SNAP[7]*]

Major Operations within this system: Join, Leave, Negotiate, and Monitor. We shall discuss each of these operations within the context of the proposed model to better understand the interactions between the components of the grid.

A. Join

To begin with let us, consider a node joining the Grid system. This node would contribute its resource to the virtual organization or pool of resources already within the grid. In order to do so it must first contact the architectural component that plays the role of directory service. And request for the GQoSE instance. Assuming that the directory is capable of locating the GQoSE factory it will bind a GQoSE instance with the new node.

At this point it is important to note that the performance sub-system (GQoSE) is distributed. Which is to say, that unlike a master-worker paradigm the GQoSE would be a peer to peer system, that relies on common knowledge to be shared between each of the nodes. This shared knowledge could be installed in a indexed retrieval mechanism such as distributed hash tables or filling-curves that keep this knowledge in synch across the Grid.

Once the node has received the GQoSE instance, it now declares the resources it intends to share through this sub-system. GQoSE in turn makes this information available to the Grid transparently. The offer to share a particular resource is made through an Offers Interface defined as follows.

```
Format of Offers Interface (OI):
1. Grid Unique ID for Resource.
      Mandatory
```

This will act as the resource locator across the grid. It is created by the local GQoSE instance before offering it to the Grid.

```
2. Usage policy for resource
      Mandatory
```

This field specifies the usage policies such as security and domain restrictions etc.

```
3. Minimal Price expectation
      Optional
```

This is the minimum price at which the node chooses to offer the particular resource.

The last but most important index i.e. the guarantee factor must be associated with each resource upon joining. There are two scenarios to assign this. Supposing that the resource offering has no prior history GQoSE shall assign the mean value of the guarantee factor associated with that particular resource type within the Grid System. On the other hand, if that resource type has a history with the Grid, the GQoSE sub-system would have the pre-calculated Guarantee Index. This concludes the Join Operation.

B. Leave

A node may leave the Grid system either as a result of a failure or voluntarily. In either case, this event will trigger a set of operations:

Case i: Node resources were not in use.

The distributed GQoSE subsystem shall detect this absence and initiate the appropriate corrections in the common knowledge such as marking the resources unavailable and recalculating the guarantee factor due to lack of availability which is then stored in the history associated with the particular resource.

Case ii: Node resources were in use.

This scenario is of particular interest to our discussion since this model offers Guarantee through redundancy. This requires redundant instances of the GQoSE to exist through similar resource offerings. As a conglomerate the GQoSE subsystem shall detect the

abrupt departure of a peer GQoSE instance. This triggers the redundant GQoSE instances to take over the responsibility, for availability of lost resources and thus maintaining the Guarantee factor to the GRC. Subsequently performing the housekeeping operations as discussed in Case i.

C. Reservation-Negotiate

Reservation is done through redundant resource. The resources offer themselves in one of two states:

1. Active Resources *(Hot Swap)*
2. Passive Resources *(Cold Swap)*

This is a characteristic with respect to the GRC and not a characteristic of the resource itself. The Passive resource offers itself to the Grid with a Guarantee Index of zero.

D. Usage Monitor

The GRC prepares a Requests Interface (RI) based on the application it wants to run.

Format of RI:

```
1. Application type:
2. Estimated Resources Required:
3. Desired Level of Security:
4. Required Band Width:
5. Required Custom Characteristic
   (E.g. Maximum Jitter, Maximum
    allowed delay)
6. Desired level of Guarantee
```

Messages could be exchanged in the following manner

1. GRC to GQoSE

The GRC(U1) gives his RI to his GQoSE. The GQoSE looks up the DHT to determine the availability of the resources and tries to find the closest match to the RI. Since there will be several resources with similar Offers Interface, the one with the least cost that matches the RI will be chosen.

2. GQoSE to GRC

GQoSE replies to GRC with the available set of resources and the bidding starts. Since there could be multiple GQoSE instances competing for the same set, the one with the highest bid shall be granted access.

3. GRC to GQoSE

The GRC then may increase its bid in order to gain access to the resource.

4. GQoSE to GRC

The resource is 'Allocated' to the highest bidder and the corresponding information is stored as part of the common knowledge.

5. GQoSE instantiates the Performance Monitor

The monitor ensures that the GRC and the Resource are 'Well-Behaved'. The details of the current transaction get added to the Historical Information associated with the resource and becomes common knowledge.

6. If the Resource provider is unable to deliver

The Guarantee associated with the Provider is reduced and the corresponding new value is made available through common knowledge. At the same time, the current transaction is broken and the monitor hands the control to the GQoSE.

7. When the processing completes, the resource is marked as free.

VIII Incentive and Pricing Models [9,10]

For the successful adoption and implementation of the proposed Architectural Model in this paper it is important that the appropriate economic *incentive* be provided. Grid systems assume an implicit *sharing* of resources but the said sharing occurs on behest of a contributor only via an economic motivator. Except, in the case of special interest groups or institutes involved in academic research. We therefore shall discuss some fair market pricing models that have been exercised for similar resource sharing business endeavors. Two such prominent pricing models are the Market Maker and the Double Auction Model[9].

A. Market Maker Model[9]

This pricing model is based on the basic principle of supply-demand. A central entity or arbitrator gathers and monitors the supply-demand characteristics of a designated commodity. This entity is deemed as the "market-maker" for the commodity and regulates a fair market price for the same. The Market Maker pools together the resources and categorizes them as per "advertised" flavors. The buyers or consumers for the resource must pay the "Market-Maker Suggested Retail Price" or MSRP in order to complete a transaction. The continuous availability and sale of resources dictates the MSRP for the particular resource; besides shared factors such as

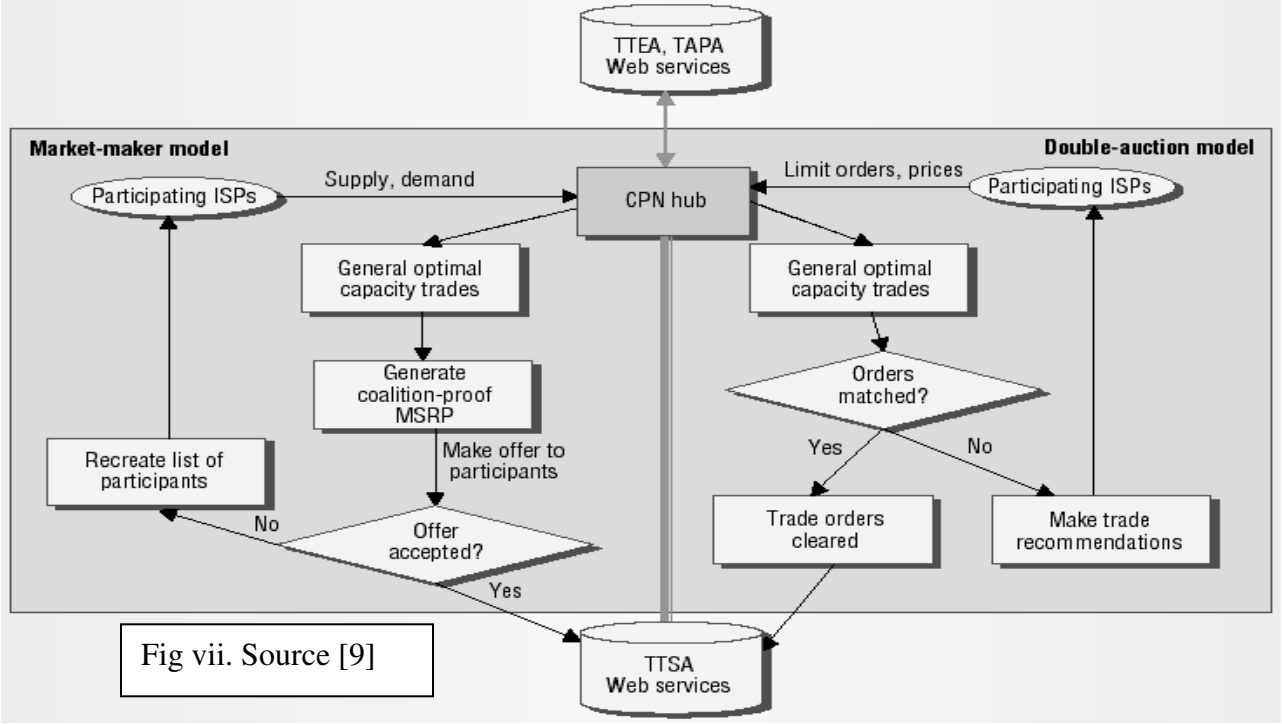

Fig vii. Source [9]

"reliability/guarantee", speed, latency etc. The resource characteristics or flavors increase the complexity of the pricing model. A simple determination based on searching the availability & reliability index of the resource is not sufficient. A complex analytical linear programming model is required to weigh in all the flavors and determine an appropriate MSRP.

Double Auction Model[9]

In this model the "single market maker" is removed from the equation. Instead each commodity owner represents himself and directly negotiates with the buyer. In such a model, where there a number of buyers and sellers each transparent to the others existence; the pricing structure is much more dynamic. Each buyer can advertise a custom set of flavors exacting a different price for a similar commodity. Allowing a more flexible auctioning model where the buyers get to balance the cost against their custom needs. This model can prove very attractive in a heterogeneous Grid system where the contributors as well as the consumers are highly diverse; and demand a variety of services and resources thereof. But such a market can also be quite volatile in its cost structure, as opposed to, a relatively homogeneous environment, where the Market Maker model can prove to offer much more stability.

The pricing models could potentially be exercised within the same physical grid; allowing the users to choose the model they would like to work with. Further more, the GQoSE does not restrict the pricing structure to these two models alone. In fact any number of pricing engines could be introduced to suit the needs of the grid community that is being served.

IX Use Cases and Frequently Asked Questions

What if a Resource fails to perform according to Guarantee?

Punish the resource by reducing Reliability Index and Price offering

What if a Resource performs better than expected?

Reward the resource by recording the higher Reliability Index and increasing the cost for procuring the resource in the history

What if the Grid Resource Consumer goes down?

Job Execution/Monitor Sub-system destroys Job

What should be the Guarantee Index of a new node?

Starting value can be fixed, suggested 0.5

What if there is a special resource/application?

The model is scalable to incorporate custom resources and the flavours thereof

How do I implement the Sub-system?

This is left to the discretion of the implementer. This paper only proposes the architecture

What if a node claims to have characteristics which are falsified?

History accounts for falsification & adjusts Indexes

Can credits be transferred between GQoSE's?

This would be a factor of the pricing model currently being exercised

How do Guarantee Factor and Price relate?

A resource with a lower Guarantee would mean that GRC are sceptical to use its resource. In order to map this to price, we can use a simple relation:

Price = Market Price of Resource * Reliability factor. However, the implementer is free to choose any relation desired

Can different nodes have different Guarantee Factors of the same resource depending upon their location or other characteristics?

Yes

What if the resources were in not use when a Resource Provider "leaves"?

This could impact Price structure (Supply/Demand)

What if the resources were in use when a Resource Provider "leaves"?

Fault Tolerance kicks in and Drops Reliability factor for provider and records this in history

X Conclusion

In this paper we proposed a method to incorporate a guarantee model and its architectural components combining it with the concepts of Grid Economics. We showed how it fits into the OGSA[23] architectural paradigm. However, a conscious attempt was made not to bind the architecture to a particular implementation. While a number of recommendations were made for the particular sub-components, the flexibility of the architecture comes from its ability to adapt to the needs of the particular grid systems, as long as the major subcomponents and their overall responsibilities are addressed. To that intent, the responsibilities for each subcomponent within the system were discussed, in an attempt to surface the various design considerations that must be kept in mind during actual implementation.

XI Future Work

Our architectural proposal was designed to fit into the OGSA model based Grids. The OGSA architecture presents a well structured approach with separate Tiers designated for particular roles. This offers an inherent "extensibility" for the addition of architectural components such as ours. However, in case of non-OGSA based grids it is quite possible that our proposed model would need to be modified and adapted to work with the new environment.

References

1. End-to-End Quality of Service for High-End Applications
2. Grid Economics: http://www.zurich.ibm.com/grideconomics/
3. Grid Economics: 10 Lessons from Finance Raj Buyya
4. Economics of Grid Computing and Web Services
5. Globus – http://www.globus.org
6. Virtual Organisations www.cs.adelaide.edu.au /~waltz/research/vo.pdf
7. Reliability Models http://www.weibull.com/SystemRelWeb/series_system.htm
8. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems
9. Scaling Web Services with Capacity Provision Networks – Andrew Whinston et al.
10. Market-Based Optimization Algorithms for Distributed Systems
11. Cactus Application : Performance Predictions in Grid Environments
12. QoS as Middleware: Bandwidth Reservation System Design
13. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment
14. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation
15. Qualis: the Quality of Service Component for the Globus Meta-computing System
16. Predicting Application Run Times Using Historical Information
17. Nimrod-G and Virtual Lab Tools for Data Intensive Computing on Grid: Drug Design Case Study
18. Management += Grid
19. SLA Management and Resource Modeling for Grid Computing
20. Towards a Unified Towards a Unified Monitoring and Performance Analysis System for the Grid
21. Performance evaluation on grid
22. The Physiology of the Grid
23. Open Grid Services Infrastructure (OGSI) Version 1.0 *(draft)*