RISK MANAGEMENT OF RESOURCE ALLOCATION IN GRID COMPUTING

K.R Palanisamy,
Assistant Professor,
Department of Computer Science,
Mahendra Arts and Science College,
Kallipatti, Namakkal, Tamilnadu.

A.Murugan,
Assistant Professor,
Department of Computer Science,
Mahendra Arts and Science College,
Kallipatti, Namakkal, Tamilnadu.

Abstract: The risk of failure is an important property of a Grid resource, especially when scheduling jobs optimally in relation to resources so as to achieve a business objective. However, in Grid computing, user-centric scheduling algorithms ignore the risk factor and mostly address the minimization of the cost of the resource allocation, or the overall deadline by which the job must be executed completely. Therefore, we propose a novel user-centric scheduling algorithm for scheduling Bag of Tasks (BoT) applications. The algorithm, which aims to meet user requirements, takes into account the risk of failure, the cost of resources and the job deadline. With this in mind, through simulation, we demonstrate that the algorithm provides a near-optimal solution for minimizing the cost of executing BoT jobs. Also, we show that the execution time of the proposed algorithm is very low, and is therefore suitable for solving scheduling problems in real-time.

Keywords: Grid Computing , job scheduling, Resource management

I.INTRODUCTION

Computer scientists in the mid-1990s began to explore a new technology known as metacomputing. . The I-WAY project—which was introduced in the ACM/IEEE conference on Supercomputing 1995 and aims to unifying the resources of large US supercomputing sites—was the first step in the field [1]. The I-WAY project was essential to the understanding and progress of the emerging new technology [2]. The evolution from metacomputing through to Grid computing occurred with the introduction of middleware, which was designed in order to function as a wide-area infrastructure to support data-intensive applications and diverse online processing [3]. Grid computing has become the alternative to the traditional tightly coupled computer systems. Currently, the Grid is defined as the coordinated sharing of resources and solving problems in dynamic, multi-institutional virtual organizations. This sharing must be controlled with clear boundaries regarding what will be shared, who are permitted to share, and the conditions under which sharing occurs, as well as whether the resources are hardware, software, or users [3][4]. The sharing should also be carried out with the use of standard, open, and general-purpose protocols and interfaces, and should deliver nontrivial quality of services (QoS) [4].

II. GRID APPLICATIONS

The interest in Grids is motivated by the novel uses of computers to solve complex applications. These applications provide the useful information and services for the reality of Grids.

Four general classes of applications that runs on Grid systems is given in [5]. It is summarized as follows:

- **Distributed supercomputing:** (also known as Meta computing): these systems solve very large and intensive problems with the use of multiple computers to achieve greater processing power. Many of the existing Grid systems and their applications are based on this class.
- **Real-time widely distributed instrumentation systems:** these systems involve real-time data sources. These systems rely on distributed-storage, network-based caches, agent-based monitoring, and generalized access control.
- **Data-intensive computing:** these systems are both data and compute intensive. These applications focus on processing and analyzing information and require terabytes or peta bytes of data to be processed and stored.
- **Tele immersion:** these systems combine advance display technologies, computers, and networks to create shared virtual environments for collaborative design, education, and entertainments.

III. TYPES OF GRID SYSTEMS

Notably, the types of Grid system are not identical; essentially, they vary widely in terms of both function and purpose. Grid systems into three categories [6]:

- **Computational Grids [7]:** denotes systems which have higher total computational capacity available for single applications than the capacity of any constituent machine in the system. Computational Grids are amongst the first type of Grid systems. An important objective of Computational Grids is to benefit from the under-utilised computational resources through sharing.
- **Data Grid [8]:** denotes systems which provide an infrastructure for synthesising new information from data repositories, such as data warehouses or digital libraries, which are distributed in a wide area network. Many scientific applications require access to a large amount of data; therefore,

data Grids are important when striving to increase the performance and to thereby achieve high throughput.

- **Service Grid [9]:** denotes systems which provide services that are not provided by any single machine. This category is further subdivided as on-demand, collaborative, and multimedia Grid Systems. An on-demand Grid category dynamically aggregates different resources so as to provide new services, e.g. allocating new machines to a simulation. A collaborative Grid connects users and applications into collaborative workgroups.

IV. GRID ARCHITECTURE

Grid architecture organizes components into layers. Components within each layer share common characteristics. Figure 1 is taken from [15], and describes a high level view of the Grid architecture. The architecture contains five layers and the following is a brief description of each one [5,10].

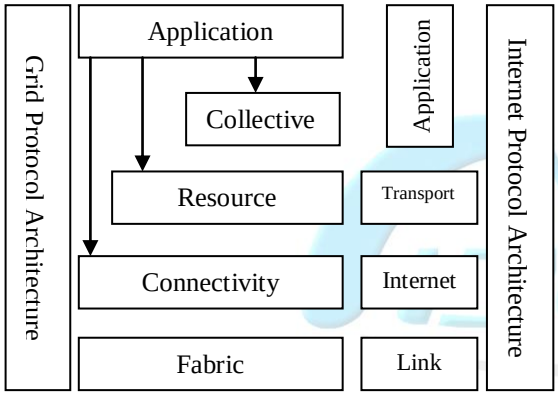

Figure 2: Grid Architecture

Grid Fabric Layer

The Grid fabric layer provides access to shared resources; these resources can be physical or logical. Notably, there is tight interdependence between functions implemented on the fabric and the supported sharing operations, which means richer fabric functionality enables sophisticated sharing operation. On the other hand, light fabric simplifies the development of the Grid. At a minimum, resources should implement introspection mechanisms that allow discovery of their structure, state, and capability, on the one hand, and resource management mechanisms that provide control of delivered QoS, on the other.

The shared resources can be divided into three main types of resources.

Computational Resources, these are the physical machines that do the processing. Four types of computational resources are suggested in [8], and are summarised her.

- **End user systems:** These are common computer machines; they have a single-functional entry and are homogeneous.
- **Clusters:** These are a group of linked computers, working together closely and are most often highly homogeneous. Clusters are usually deployed in order to improve performance and/or availability over that of a single computer, whilst typically being much more cost-effective than single computers of comparable speed or availability.
- **Intranets:** These are large local networks of resources within a single organization; they are diverse and heterogeneous by nature, and different parts of the network may be under different administration, which results in less global knowledge regarding the resources.
- **Extranets:** These are networks of Intranets. They span multiple organizations and are more heterogeneous than Intranets and have less global knowledge available.
- **Storage Resources**: These are dedicated storage machines which can hold very large amounts of data. This may be a simple file system or a large and complex database.
- **Network Resources**: These are the cable switches and routers that make the physical network. The network is measured by capacity (bandwidth) and latency.

Grid Connectivity Layer

This layer defines core communication and authentication protocols. The communication protocols are to enable the exchange of data between resources. Authentication protocols, which are built on the communication protocols, are required to provide secure mechanisms for checking users and resources.

Grid Resource Layer

This layer is built on the protocols of the connectivity layer, and defines protocols for secure negotiation, initiation, monitoring, control, accounting, and payment of sharing resources. The two primary protocols on this layer are information protocols, and management protocols.

Grid Collective Layer

This layer contains protocols and services not linked with a specific resource but instead containing interaction across collection of resources. This can enable the implementation of a wide variety of sharing behaviours without placing new requirements on the resources being shared.

Grid Application Layer

This layer contains the user applications. The applications are implemented by calling services defined at any layer.

V. GRID RESOURCE MANAGEMENT

Grids provide cost-effective and easily scalable resources, although the commercial uptake of Grid computing has remained slow. Current Grid middleware (e.g. Globus Toolkit) still follows the best-effort approach; there is a risk that users do not get any guarantee that their SLA will be fulfilled.

Furthermore, Grid resource providers are not attracted either: for a resource provider, agreeing on an SLA without enough information about the state of resources and the availability of devices introduces a chance of violating the SLA, which can then result in a penalty fee. Furthermore, there is a risk attached to system failure, service unavailability, insufficient resources, etc., which might lead to SLA violation. Importantly, without a method for assessing the risk of accepting an SLA, providers are only able to make uncertain decisions regarding suitable SLA offers. Furthermore, users would like to evaluate the risk of a provider violating an SLA so that they are able to make decisions concerning which Grid resource provider to select and the acceptable cost/penalty fee associated with the SLA[11].

The provider computes the risk for each resource and subsequently allocates the resources to the user job. If the resulted allocation fails to satisfy the user requirements, resource reservation is revisited; if it does satisfy the user requirements, the provider then sends back the SLA, updated with cost and penalty fee and pre-commit. The user either commits to the SLA or rejects it. Figure 2 provides an overview of components in the resource provider infrastructure. The user sends an SLA request to the provider with the job requirements (1). The provider's Resource Manager requests the Reservation & Allocation component to reserve the required resources (2). The Reservation & Allocation component reserves the physical resources (3) and forward for each reserved resource the time and duration of the reservation to the Risk Assessor (4). The Risk Assessor computes for each resource the risk of failure based on the resource historical information stored in a database (5). The Monitoring component is responsible for updating the information in the database. The Risk Assessor returns the risk of failure information to the Resource Manager (6). Finally, the Resource Manager sends the SLA response back to the user (7), either accepting or rejecting the SLA.

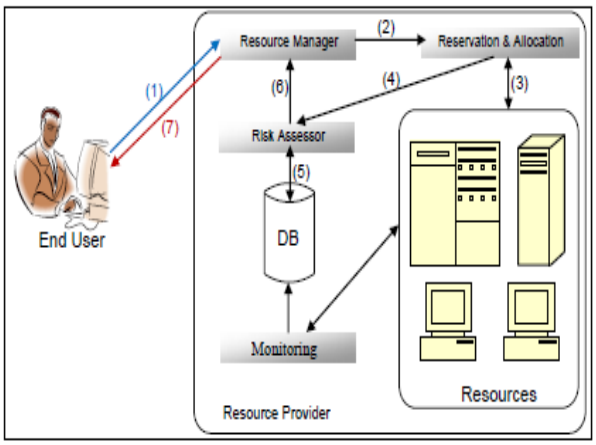

Figure 2: Components in Resource Provider

VI.RISK MANAGEMENT IN GRID

Risk management is a paramount importance in computer systems are less or non-critical, such as web servers or email servers, and the risk of faults of such systems is also lower than the risk of faults in critical systems. Nevertheless, a malfunction of non-critical systems might still result in losses of devices or money; therefore, risk management on such systems needs to balance between the cost of the risk management process and the expected loss as a result of faults. The risk management process cost should always be lower than the expected loss, otherwise it is more profitable not to implement such a process. Grid systems fall into the arena of non-critical systems.

Risk management can be carried out at various phases during the lifetime of a Grid system, i.e. from the development of a Grid infrastructure, through to the deployment and testing phase, right up to the operational phase. The rest of this section is devoted to review approaches adopted for risk management in computer systems in general, and Grid systems in particular.

a) Risk Identification

There are different sources of risks in Grid systems, depending on the systems phases: for example, in the development phase, there is a risk of software development failure; in the operational phase, there is a risk of hardware failure, information security breaches, etc. Each phase has various different risks associated with it, and events causing those risks need to be identified. This paper does not address risk specifically, although it does identify three factors relevant for reliability: software failures, hardware availability, and network failures [12]. Software failures and network failures are modelled in a probabilistic way, whilst hardware availability is modelled as the Mean Time To Failure (MTTF), divided by the sum of MTTF and Mean Time To Repair (MTTR). Other factors relevant for reliability are ignored in this project.

The scenario-based risk identification approach [13], and identifies two risk items: the risk to the resource provider, and the risk to the broker. The risk to the resource provider is the violation of users' SLAs, which is influenced significantly by resources failure. A source analysis is used to identify the resource failure, which can be internal, such as hardware failure, problems in software components, version problems in used software systems, power supply problems, etc., or external, such as no delivery on external contracts, natural disasters, etc. The risk to the broker is the unreliable methods used by the resource provider to assess the risk of failure. The broker plays a mediator role between Grid providers and users: its primary task includes the assignment of the user jobs to certain resource providers in order to minimize the overall possibility of failure in carrying out those jobs. Importantly, the broker aims to minimize the aggregate risk of failure of all tasks under its management.

b) Risk Assessment

A fundamental concept in risk assessment is the concept of Risk Exposure (RE), sometimes referred to as risk impact [14]. RE is defined as:

RE = Prob (UO) * Loss (UO)

Where Prob (UO) is the probability of an unsatisfactory outcome and Loss (UO) is the loss to the parties affected if the unsatisfactory outcome occurs. RE is then used to produce a ranked ordering of the risk items identified.

In consideration of software development projects, the probability and the loss of an unsatisfactory outcome are accessed via application of the qualitative risk analysis technique.

The aim of this survey is to serve as a checklist of the most important risks for project managers to focus on. [15] the four risk categories proposed in namely Customer Mandate, Scope & Requirements, Execution and Environment. A survey of 507 project managers, representing multiple industries, indicated the extent to which each risk item was present during their most recently completed projects. A scale from 1–7 is utilized so as to represent the presence of a risk item; higher numbers represent a higher presence and lower numbers a lower presence. The result identifies the risk associated with the Scope & Requirements and Execution categories to be the most critical, and that the Environment category is not of great importance.

c) Resources Failures

The source of a failure falls in one of the following: human errors and environments, such as power outages, hardware failure, software failure, network failure and unknown failures. They find that the time between failure at individual nodes—as well as at an entire system—is fit well by a gamma or Weibull distribution with decreasing hazard rate. The observation that the time between failures is best fitted by a distribution with decreasing hazard rate is evidence. It has considered the availability of CPUs in a Grid environment and analyse availability traces recorded from all the clusters. The finding is that the best fit distribution is with a shape parameter > 1. The reason for that is that many of today's Grids comprise computing resources grouped in clusters, the owners of which may share them only for limited periods of time. Often, many of a Grid's resources are removed by their owner from the system—either individually or as complete clusters—in order to serve other tasks and projects; thus, the unavailability of CPUs is not owing to a system failure but rather their unavailability by their owner. Most of the previous studies considered only short-term availability data [16,17].

d) Risk Response

The risk to software development projects—as well as the risk to information security—is usually treated at the design phase. The aim is to lower both the likelihood and the impact of an undesired event. The Software Engineering for Service-Oriented Overlay Computers (SENSORIA) project [18] provides tools to enable developers to model their Grid applications at a very high level of abstraction with the use of service-oriented extensions of the standard UML, or domain-specific service-oriented modelling languages to translate into hidden formal representations by automated model transformations. Furthermore, such tools are able to perform early performance analysis, check the functional correctness of services, and accordingly predict the bottlenecks in collaborating services.

The responses to the risk of resources failure are to lower the probability of the failure or to lower the impact. The probability of failure can be lowered by investing in new infrastructures, advanced monitoring services, and experienced system administrators, etc. Importantly, the impact of a failure can be lowered through the use of fault-tolerance mechanisms, such as reserve idle resources and check pointing. Check pointing is the process of periodically saving sufficient information about application or resource state to avoid having to restart the application from the beginning [19]. The advantage of combining the checkpointing with PoF is that checkpointing will be carried out frequently in relation to those resources with high PoF, and less frequently concerning those resources with low PoF. [20].

VII. FACING RESOURCE FAILURE AND IMPROVE SCHEDULING

Grid environments provide computing platforms for solving large-scale computational and data-intensive problems in science, engineering, and commerce. They can be very cost-effective and easily scalable yet, owing to resource heterogeneity and to the lack of accurate resource information, scheduling jobs in such systems can be challenging. In this chapter, the problem of scheduling Bag of Tasks (BoT) application on Grid resources is modelled using Mixed Integer programming. An efficient algorithm for solving the scheduling problem is presented. The algorithm is evaluated with the use of a simulation, allowing a wide range of possible scenarios to be considered.

Grid users submit their application to resource providers through the use of SLAs. The SLA has the user application information, as well as the user requirements and constraints. Notably, requirements can include the type of hardware, the type of operating system, or even a business objective, such as minimising the costs associated with executing the application. Moreover, a constraint could be the deadline by which the application results should be delivered. Once the resource provider receive an SLA, it is translated into an allocation problem whereby the application is allocated to resources for executing, ensuring that, during the execution time, the user requirements and constraints are being fulfilled.

Application Model and Scheduling : The type of applications which are executed on Grid systems can vary from long running computationally intensive simulations to high demand and high priority time critical transaction based executions, to real-time interactive visualizations. Notably, the majority of these applications are sequential applications, often submitted in the form of Bags of Tasks (BoT). According to BoT jobs account for up to 96% of the CPU time consumed in Grid environments. BoT jobs are composed of sequential, independent tasks where there is no communication or dependencies amongst tasks. Examples of BoT jobs include

Monte Carlo simulations, massive searches (such as key-breaking), image manipulation applications, data mining algorithms, and parameter-sweep applications [21]. Therefore, the type of applications which this thesis is targeting is BoT jobs.

Executing BoT jobs involves processing *N* independent tasks on *M* distributed resources where *N* is, typically, much larger than *M*. For each task $n \in N$ its computation time is known. Scheduling the tasks to resources appears simple, but complexities arise when users place their desired constraints. The job owners submit their BoT jobs and requirements in real time (in the reset of the chapter the job owners are referred to as users); therefore, the scheduler must find the tasks assignment efficiently and effectively for each user. The scheduling is carried out in real-time, and the users' BoT are assigned as first in, first out (FIFO). If an assignment is found which has satisfied the user requirements, the user BoT job is then accepted; otherwise, the job is rejected.

DRFC ALGORITHM

In the DRFC algorithm, the interest is directed to striking a balance between the objective function and the constraints in order to reduce the BoT execution costs. Therefore, tasks should be allocated to the cheapest suitable resources whenever possible. The cost per time unit does not reflect the true cost of processing, especially when resources have different processing abilities; therefore, the DRFC algorithm will start by calculating the true processing cost for each resource. This is defined as the resource processing ability, and is measured in million instructions per second (MIPS) and divided by the resource price/time unit.

True Processing Cost = Resource Processing Ability(MIPS)/ Resource Cost per Time Unit

The DRFC algorithm sorts the resources in decreasing order, based on the true cost of processing. It is clear that tasks cannot be assigned to resources with ROF higher than the user desired ROF level; therefore, such resources are removed from the sorted list.

The next step in the DRFC algorithm is to arrange the tasks, within a single BoT job, in decreasing order, based on executing time, to be assigned to resources. Starting from the first task in the sorted tasks list, the task needs to be assigned to the first resource in the resources list, if feasible, based on the values of *tjk*, *Aj*, *Uj* and *D*. Subsequently, the task is then removed from the tasks list and the resource variables are updated accordingly. If the task cannot be assigned to the resource, it can be kept within the list, at which point the next task can be considered and the assignment repeated. Once the DRFC goes through the entire tasks list, if there are tasks in the tasks list, then go to the next resource in the resources list, start from the beginning, and repeat the process. This is repeated until the tasks list is empty and a schedule is found or the resources list is empty, before the tasks list, and the BoT job is rejected.

// The number of tasks in the BoT job is *e*
// The number of resources in the resource provider domain is *n*
// The MIP parameters are used in the pseudocode
Step 1: // Compute the true processing cost (TPC) for each resource for (Resource1 to Resourcen)

TPC Resource $_i$ = Resource$_i$ Processing Ability(MIPS)/ Resource$_i$ Cost per Time Unit

Step 2: // Sort the resources in decreasing order based on TPC
Step 3: Remove all Resources with ROF > *JR*
Step 4: // Sort the tasks in decreasing order based on execution time

tj1>tj2>tj3>….> tje

Step 5: // Assign the tasks to resources Start from the first Resource in the Resources list (*j = 1*)
Start from the first Task in the Tasks list (*k = 1*)
Total cost = 0
While (the Resources list is in not empty)
{
While (the Tasks list is not empty)
{
if (*tjk + Aj <= D)* then
{
Assign the task to the resource
Remove the task from the Tasks list
Update *Aj* & *Uj*
Total cost += *tjk * cj*
Move to the next task
}
else
Move to the next task in the Tasks list
}
if (the Tasks list is empty) then
Break
else
Move to the next Resource in the Resource list
}
if (the Resource list is empty) then
The BoT job cannot be assigned and therefore rejected
else
{
The assignment for the BoT job is found
The cost of executing the BoT job is Total cost
}

Figure 3: The DRFC Algorithm.

The approach applied to assign tasks to resources—known as the greedy approach—has a number of advantages over other approaches. For example, the algorithms in [21] assign the tasks to resources in the order in which they appear in the BoT job. This approach is not efficient for two reasons: firstly, it is inconsistent and a BoT job—scheduled on the same resources—will have different assignments if the order of tasks in the BoT job is changed; and secondly, it does not fully utilise the resources, and a BoT job might be rejected, although an

assignment is feasible. In order to illustrate these limitations, a simple example is given.

Assume there are two resources with the same processing ability, and a BoT job is submitted for processing with 100 time units as a deadline. Both resources are suitable for executing the tasks; Resource A is available from the time the BoT job is submitted, whilst Resource B is available after 50 time units. Resource A is cheaper than Resource B; thus, tasks will be assigned to Resource A first. Let's assume that the time the BoT job was submitted is 0. The BoT job has three tasks, which are to be carried out in the following order:

1. Task 1 execution time is 30 time unit.
2. Task 2 execution time is 40 time unit.
3. Task 3 execution time is 70 time unit.

Assigning the tasks to resources in the order in which they appear will result in the rejection of the BoT job. Figure 64 shows that, after Task 1 and 2 are assigned, Resource A's available time is 30 time units and Resource B's available time is 50 time units. Both resources are not able to execute Task 3, which requires 70 time units.

The above assignment approach rejects the BoT job, despite there being two possible schedules. Accordingly, Task 1 and 3 should be assigned to Resource A, and Task 2 to Resource B ; otherwise, Task 2 and 3 should be assigned to Resource A and Task 1 to Resource B. The latter is better as the cheaper resource is used for a longer period. Essentially, using the greedy approach will result in the latter assignment, and will always be consistent regardless of the tasks order.

VIII.CONCLUSION

The work presented in this paper introducing of Grid computing as the broad. Grid systems which enable access to heterogeneous resources are discussed. Furthermore, **Service Level Agreements** are presented as languages which formalize QoS requirements, and a discussion on a number of specifications actively used within the Grid research domain is presented. A discussion of Grid resource management identifies a number of limitations in Grid resources scheduling. We introduce the definition of risk and its application in the real world. Methods for risk assessments including qualitative and quantitative are defined. A discussion of risks affecting Grid systems narrows the research to assessing the Grid resources risk of failures. In order to highlight risk assessment in Grid computing, a number of assessment methods applied in the field are described. Finally work of resources failure to overcome the DRFC algorithms limitations and the Mixed Integer Programming model to minimise the cost of executing a BoT job whilst guaranteeing the user's requirements are presented. DRFC algorithm to determine a near-optimal solution in an acceptable time frame is described.

IX.REFERENCES

[1]. Roure, D.D., et al., The Evolution of the Grid, in Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox, and A.J.G. Hey, Editors. 2003, J. Wiley: New York. p. 65-100.

[2]. Foster, I., et al., Software Infrastructure for the I-WAY High-Performance Distributed Computing Experiment, in Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox, and A.J.G. Hey, Editors. 2003, J. Wiley,: New York. p. 101-115.

[3]. Foster, I. and C. Kesselman, The Grid in a Nutshell, in Grid resource management : state of the art and future trends, J. Nabrzyski, J.M. Schopf, and J. Weglarz, Editors. 2004, Kluwer Academic Publishers: Boston. p. 3-13.

[4]. Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid, in Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox, and A.J.G. Hey, Editors. 2003, J. Wiley,: New York. p. 169-197.

[5]. The grid : blueprint for a new computing infrastructure, ed. I. Foster and C. Kesselman. 1999, San Francisco: Morgan Kaufmann Publishers.

[6]. Krauter, K., R. Buyya, and M. Maheswaran, A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. Software: Practice and Experience, 2002. 32(2): p. 135-164.

[7]. Foster, I. and C. Kesselman, Chapter 2: Computational Grid, in The grid : blueprint for a new computing infrastructure. 1999, Morgan Kaufmann Publishers: San Francisco. p. 15-53.

[8]. Chervenak, A., et al., The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets Journal of Network and Computer Applications, July 2000. 23(3): p. 187-200.

[9]. Foster, I., et al., The Physiology of the Grid, in Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox, and A.J.G. Hey, Editors. 2003, J. Wiley,: New York. p. 217-249.

[10]. Berman, F., G. Fox, and T. Hey, The Grid: Past, Present, Future, in Grid Computing : Making the Global Infrastructure a Reality, F. Berman, G. Fox, and A.J.G. Hey, Editors. 2003, J. Wiley,: New York. p. 9-50.

[11]. Foster, I., C. Kesselman, and S. Tuecke, The Open Grid Services Architecture, in The grid : blueprint for a new computing infrastructure, I. Foster and C. Kesselman, Editors. 2004, Morgan Kaufmann: Amsterdam ; Boston. p. 215-258.

[12]. D.A6a Predictable / Manageable Service Engineering Methodology and Prediction Services. September 2010 Available from: http://sla-at-soi.eu/wp-content/uploads/2009/07/D.A6a-M26-PredictableServiceEngineeringMethodology.pdf.

[13]. Risk Management Evaluation. 31/10/2006 Available from: http://www.assessgrid.eu/fileadmin/AssessGrid/usermounts/publications/deliverables/AssessGrid_D.1.2_Risk_Management_Evaluation.pdf.

[14]. ISO/IEC 27005:2008 Information technology -- Security techniques -- Information security risk management. 2008 Available from: http://www.iso.org/iso/catalogue_detail?csnumber=42107.

[15]. Kavanaugh, G.P. and W.H. Sanders. Performance Analysis of Two Time-Based Coordinated Checkpointing Protocols. in Proceedings Pacific Rim International Symposium on Fault-Tolerant Systems. 1997. Taipei , Taiwan.

[16]. Nurmi, D., J. Brevik, and R. Wolski, Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments in Euro-Par 2005 Parallel Processing. 2005, Springer Berlin / Heidelberg.

[17]. Nadeem, F., R. Prodan, and T. Fahringer. Characterizing, Modeling and Predicting Dynamic Resource Availability in a Large Scale Multi-purpose Grid. in 8th IEEE International Symposium on Cluster Computing and the Grid, CCGRID '08. 2008. Lyon, France

[18]. Software Engineering for Service-Oriented Overlay Computers (SENSORIA). Available from: http://www.sensoria-ist.eu.

[19]. Young, J.W., A First Order Approximation to the Optimum Checkpoint Interval. Communications of the ACM, 1974. 17(9): p. 530-531.

[20]. Zhang, Y., et al., Performance Implications of Failures in Large-Scale Cluster Scheduling, in Job Scheduling Strategies for Parallel Processing, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Editors. 2005, Springer Berlin / Heidelberg. p. 233-252.

[21]. Silva, F.A.B.d. and H. Senger, Improving Scalability of Bag-of-Tasks Applications Running on Master-Slave Platforms. Parallel Comput., 2009. 35(2): p. 57-71.

[22]. Buyya, R., M. Murshed, and D. Abramson. A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids. in Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02). 2002. Las Vegas, USA.

