A Decentralized Utility-based Scheduling Algorithm for Grids

João Luís Vazão Vasques
joao.vasques@ist.utl.pt

Instituto Superior Técnico
Av. Prof. Doutor Aníbal Cavaco Silva - 2744-016 Porto Salvo
INESC-ID

Abstract. Grid systems have gain tremendous importance in past years since application requirements increased drastically. The heterogeneity and geographic dispersion of grid resources and applications places some difficult challenges such as job scheduling. A scheduling algorithm tries to find a resource for a job that fulfills the job's requirements while optimizing a given objective function. Utility is a measure of a user's satisfaction that can be seen as a objective function that a scheduler tries to maximize. Many utility as an objective scheduling algorithms have been proposed. However, the proposed algorithms do not consider partial requirement satisfaction by awarding an utility based on the total fulfillment of the requirement and follow centralized or hierarchical approaches suffer problems concerning scalability and fault tolerance. Our solution proposes a decentralized scheduling architecture with utility based scheduling algorithm that considers partial requirements satisfaction to overcome the shortcomings of actual solutions.

Keywords: grids, utility scheduling, partial utility, resource discovery

1 Introduction

The idea of having access to computational power as we have to electricity is not new. In 1961, John McCarthy stated that "computation may someday be organized as a public entity." The term "Grid" was chosen because of the parallel that was made to the electric grid. An electrical grid provides resources, i.e. electricity, to many heterogeneous entities in a distributed and geographic dispersed environment. Grid computing follows the same principle but with different participants since it provides computational power to users instead of electricity. A Grid is formed by many heterogeneous resources. Sets of resources that share common sharing rules and conditions are called Virtual Organizations (VO).

Grid computing had its breakthrough on the 1990's which coincided with the boom of the Internet. The massification of the internet combined with the constant increase of network bandwidth computing power of devices (see the transistors Moore's law) and decrease of resources cost opened the doors to Grid computing.

The computational power required by science nowadays is huge. Genetic studies, large macroeconomic simulations and physicists trying to find the origin of the universe are examples of investigation areas that need to have access to a lot of computational power, i.e. computational resources. Due to the continuous growing need of science for computational resources it is important to have mechanisms that assure that shared resources are used in an efficient and fair way. For this reason, grid scheduling is a very important problem that has been widely studied by the computer science community. The purpose of grid scheduling is to allocate a job to a resource, fulfilling the job's requirements while optimizing resource utilization.

Many solutions have been propose to address this problem. However, they do not consider partial requirement fulfillment and rely on a centralized or hierarchical architectures that have problems of scalability and fault tolerance. To address this shortcomings we propose a grid scheduling algorithm that considers partial requirement fulfillment based on information provided by the user and considers the grid as a structured peer-to-peer network where the peers are the VOs.

The rest of the document is organized as follows: in section 2 we present what we pretend to achieve, the study and analysis of the related is work is done in section 3, in section 4 we present our solution to address the shortcomings mentioned before, the evaluation methodology and metrics are described in section 5, in section 6 we present some concluding remarks. There is also an appendix section. In appendix A we present some metrics that are used to evaluate the performance of scheduling algorithms. In appendix B we present a calendarization that we are going to follow during the implementation.

2 Objectives

The goal is this work is to develop a decentralized utility based scheduling algorithm for grid environments. The algorithm will incorporate partial requirement fulfillment based on information provided by the user that, to the best of our knowledge, no other utility based grid scheduling algorithm uses. With this new scheduling approach, we pretend the optimize resource utilization and maximize user's utility. By using a decentralized P2P architecture we plan to address the problems of scalability that are inherent to most of the solutions that have been proposed.

3 Related Work

This sections describes the most relevant research work for the definition of our utility-based scheduler, organized according to a top-down approach. In section 3.1 we present an overview of the existing middleware for grids. Next, on section 3.2 we describe how resource discovery is performed is grids. Finally, in section 3.3 we describe important scheduler aspects, such as scheduling phases, classes of algorithms and some scheduling algorithms.

3.1 Grid Middleware

Middleware aims to provide abstractions to programmers by shielding them from the complexity of the grid. Next we present some of the most important grid middleware systems. We will cover some important aspects of grid middleware such as how an application is defined, what kind of applications are supported, what is the grid architecture that the middleware was built for.

3.1.1 Condor-G Condor-G [30] aims at providing users access to computational resources at many sites, which is a challenging issue due to the wide variety of grid resources.

Condor-G combines the intra-domain resource and computational management methods of Condor [57] with the inter-domain resource management protocols of the Globus Toolkit [31] . From Condor comes important aspects related to intra-domain resource and job management, such as resource discovery, job submission, job allocation and scheduling [35]. From Globus project it uses the Globus Toolkit protocols to address the remote resource access issue, namely: the Grid Security Infrastructure (GSI) [29] for authentication and authorization allowing the system to authenticate a user just once; the Grid Resource Allocation and Management (GRAM) [17] for remote submission of a computational request; the Monitoring and Discovery System (MDS) [16] for getting information about grid resources; and the Global Access to Secondary Storage (GASS) [5] for data transfer.

The Condor-G agent, also named computational management service, allows the user to treat the grid as a local resource and gives the possibility to perform operations such as submitting jobs, query a job's status, be informed of job termination and access job's logs. Condor-G agent can be accessed by a personal desktop agent, which uses the Globus protocols described above to interact with the machines on the Grid.

Jobs and resources are announced through the use of ClassAds, a set of uniquely named expressions, e.g. attributes formed by pairs of (name, values), that is assembled using a semi-structured data model [57]. Condor-G supports two kinds of application types: Bag-of-Tasks (BOT) and Message Passing Interface (MPI). A BOT application consists of multiple independent tasks with no communication among each other. A MPI application is composed of multiple tasks with inter-task communication. When a user submits a job, the job is passed to the agent's scheduler, which is responsible for job scheduling, monitoring, fault-tolerance and credential management [49]. The scheduling operations are performed using the Matchmaking mechanism [30, 36] using a centralized matchmaker. Resources and users express their characteristics and the constrains to the matchmaker. The matchmaker uses the information from the ClassAds to select the appropriate resource. When the scheduler receives a job request, it creates a Condor-G GridManager deamon which is responsible for managing and submitting all the jobs of a single user. The GridManager terminates when all the user's jobs are completed. Each job submission request of the GridMan-

ager results in the creation of a Globus Job Manager on the selected resource(s). Condor-G follows a centralized scheduling approach.

In [38], Jacob et al. propose a multi dimensional matchmaking framework to overcome some of the Condor-G's matchmaking shortcomings, such as lack support for parallel jobs and non-consideration of dynamic information.

3.1.2 Nimrod-G Nimrod-G [6] is a Grid middleware for building and mananing large computacional experiments over distributed resources [49] that supports dealine and economy-based computations. Experiments are described using a simple declarative parametric modeling language (DPML). Nimrod-G supports BOT and MPI applications.

Nimrod-G uses the Globus [28] middleware services for dynamic resource discovery and dispatching jobs. The main components on the Nimrod-G architecture are: Client or User Station, Parametric Engine, Scheduler, Dispatcher and Job-Wrapper.

The User Station is a user-interface whose function is to control and supervise a specific experience (list status of all jobs). The user can vary time and cost parameters while the scheduling is taking place.

The Parametric Engine receives the experiment plan described by the declarative parametric modeling language and is responsible for maintaining the state of the experiment, creation of jobs, maintenance of job status and interacting with clients, scheduler and dispatcher.

The Scheduler is responsible for resource discovery, resource selection and job assignment. Nimrod-G's Scheduler is organized in a hierarchical way unlike Condor-G which follows a centralized approach. The resource discovery algorithm interacts with the Grid Information Service to get a list of the authorized machines and to keep track of the resources status. Nimrod-G scheduling approach is based on computational economy [2, 7, 49]. Nimrod-G was one of the first grid middleware using the computational economy approach to grid scheduling. Computational economy can be handled in two ways. First, the system can work on the user's behalf and try to complete the assigned work within a given timeline and cost. Second, the user can negotiate for resources and find out if the job can be performed. Important parameters of computational economy are the resource cost (set by its owner), the price the user is willing to pay and the deadline for the execution completion. Nimrod-G's scheduling objective is to maximize utility.

The Dispatcher initiates the execution of a task on the resource selected by the Scheduler and periodically updates the task execution status to the Parametric Engine.The Dispatcher is also responsible for starting the Job-Wrapper.

The Job-Wrapper responsible for setting up the environment on the selected resource for a task [2], starting the execution of the task on the selected resource and sending the results back to the Parametric Engine via Dispatcher.

3.1.3 GrADS GrADS is the abbreviation of Grid Application Development Software [4]. GrADS aims to provide programming tools and execution environ-

ments for development of applications on the Grid. GrADS supports MPI and workflow applications.

In GrADS, an application must be encapsulated into Configurable Object Program (COP) which can be optimized for execution on a specific collection of resources [4,15]. A COP includes the application code, a Mapper that determines how to map tasks to a set of resources and a Resource Selection Performance Model that can be used to estimate the performance of the application on a set of resources [15, 46].

The system relies upon performance contracts that specify the expected performance of each application as a function of available resources. When a COP is sent to the execution environment, the system must determine available resources and secure them to the COP. The Monitoring and Discovery System (MDS) retrieves the application requirements and filters the resources that can not be used [39]. The Performance Modeler [59] uses the information retrieved from the MDS and the Application Performance Model to determine the resources for the application execution. The scheduler ranks each qualified resource for each application component [15]. After the ranking, a performance matrix is constructed and used by scheduling heuristics to map the COP into resources. Three scheduling heuristics have been applied in GrADS: Min-Min, Max-Min and Sufferage. When the resources are selected, GrADS will start the COP on those resources. The execution is tracked the Contract Monitor [15, 59] that detects anomalies and may call a rescheduler if necessary. GrADS follows a centralized scheduling architecture.

3.1.4 Askalon Askalon [23] is a Grid middleware for application development and computing environment whose goal is to provide an invisible Grid to application developers. Askalon provides four tools to the user: Scalea, Zenturio, Aksum and PerformanceProphet.

Scalea [58] is a performance instrumentation, measurement, and analysis tool. Zenturio [48] is a tool designed to specify and automatically conduct large sets of experiments, supporting multi-experience performance analysis. Aksum [25] is a tool for performance analysis that helps programmers to understand performance problems such as message passing and mixed parallel programs. The PerformanceProphet helps the users in terms of modeling and predicting the performance of behavior of distributed and parallel applications. Unlike other middleware systems such as Condor-G and Nimrod-G, Askalon is design as a set of distributed grid services using web services.

Askalon supports workflow applications. A workflow application can be modeled as a Direct Acyclic Graph (DAG) where the tasks are the nodes and the dependencies between tasks are the arcs among the nodes. The user can describe workflows using the XML-based Abstract Grid Workflow Language (AGWS) [22]. Askalon's Resource Manager, GridARM [24], provides user authorization, resource management, resource discovery and advanced reservation. Resource discovery and matching are performed based on the constrains provided by the Scheduler.

Askalon's Scheduler has a centralized architecture and processes the workflow specification described in AGWL, converts it to an executable form and maps it onto available resources [24]. The Scheduler uses GridARM to get information about the Grid resources and maps the workflow onto resources using a Genetic Algorithm based on user-defined QoS parameters. After that, a dynamic scheduling algorithm takes in consideration aspects such as machine crashes or CPU and network load and performs a reschedule if necessary. The Execution Engine is responsible for controlling the execution of a workflow based on the information provided by the Scheduler.

3.1.5 Pegasus Pegasus [18, 19] is part of the GridPhyN project [63] and is a system that maps complex scientific workflows onto Grid resources.

Pegasus uses the Globus Toolkit [31] GRAM [17] for remote job submission and management; Monitoring and Discovery Service (MDS) [16] to get information about the state of resources; Replica Location Service (RLS) [13] to get information about the data available at the resource.

Pegasus uses DAGMan and Condor-G [30] to submit jobs on Globus-based resources. There are two main components in Pegasus: Pegasus Workflow Mapping Engine (PWME) and DAGMan workflow executor for Condor-G. PWME receives an abstract workflow description and generates an optimized concrete workflow . An abstract workflow describes the computation in terms of logical files and logical transformations and indicate the dependencies between the workflow components and can be described using Chimera's [27] Virtual Data Language (VDL). A concrete workflow is an executable workflow that DAGMan can process. First, Pegasus queries the MDS to get information about the availability of the resources. The next step consists in reducing the workflow to only contain the necessary tasks for the final product. This is done by querying the RLS for replicas of the required data. Next, Pegasus queries the Transformation Catalog (TC) to find the location of the logical transformation (software components) defined on the workflow. The information obtained is used to make scheduling decisions (random selection, round robin, min-min). It is possible add new scheduling algorithms to Pegasus. Pegasus has an option that clusters jobs together in case the are small jobs assigned to the same resource. this point resource selection has been done. The information about the application and the selected resources is used to build a concrete workflow which is sent to DAGMan. DAGMan will follow the dependencies of tasks and submit to Condor-G that will dispatch them to selected resources.

3.1.6 Comparison Table 1 summarizes the most relevant properties of the different middleware systems that were described. Using information from [49] five categories have been chosen to make the classification: Application Type, Application Definition, Scheduling Architecture, Scheduling Objective and Self-Optimization. The Scheduling Objective has two categories identified by the letters d and r meaning deadline and resource utilization respectively.

Systems	Application Type	Application Definition	Scheduling Architecture	Scheduling Objective	Self-Optimization
Condor-G	BOT/MPI	ClassAd	Centralized	Load Balancing	Dynamic rescheduling
Nimrod-G	BOT/MPI	DPML	Hierarchical	Utility/Optimization(d)	Dynamic rescheduling
GrADS	MPI/Workflow	COP	Centralized	Optimization(d)	Dynamic rescheduling
Askalon	Workflow	AGWL	Centralized	Utility/Optimization(r)	Performance prediction
Pegasus	Workflow	VDL	Centralized	Optimization(r)	Dynamic rescheduling

Table 1. Middleware classification

3.2 Resource Discovery in Grids

Resource discovery is the process of searching and locating resource candidates that are suitable for executing jobs. The dynamic and heterogeneous nature of the Grid makes efficient resource discovery a challenging issue. In this section we will present some of the most important approaches of resource discovery and some implementations.

3.2.1 Classification of Resource Discovery Systems Resource discovery in Grids can be classified in three main categories, regarding their fundamental architecture: centralized, hierarchical and decentralized/peer-to-peer (P2P).

- **Centralized:** in the centralized category, resource discovery is performed by querying a unique server. This solution is adopted by some systems such as Condor-G [30] and Askalon [23]. Having a central server provides easier interfaces to manage grid resources. This approach has many disadvantages such as single point of failure that compromises the availability of the whole system, lack of scalability and can easily lead to bottleneck in frequent updates or requests. Web services grid management tools are profusely used in centralized resource discovery mechanisms.
- **Hierarchical:** in the centralized category, resource discovery is performed by querying servers laid out as a hierarchy. This solution is adopted by Nimrod-G [6]. The hierarchical approach was adopted by some systems to overcome the problems caused by centralized resource discovery. This approach is more scalable and reduces the bottleneck problem when compared to the centralized one. However, single point of failure still exists since failure of one server may cause that a large part of the nodes become invisible to queries. Like in centralized solutions, web services are very used in hierarchical resource discovery.
- **Decentralized or P2P:** Grid and P2P environments are very similar in some aspects such as: large scale (millions of shared resources), lack of global

centralized authority and strong diversity of resources. One of the first works to propose a P2P approach to grid resource discovery was [34] by Iamnitchi et al. P2P is more scalable and fault tolerant than the two previous approaches and shares many characteristics with grids. For these reasons, P2P approach to resource discovery is adequate to the grid.

3.2.2 Resource Description and Matching In this section we will present some solutions to the resource discovery problem in grid environments.

Centralized resource discovery solutions use a central server to discover resources. Condor-G's matchmaking mechanism [30, 36] uses a "Matchmaker" (central server) where resources advertise their specifications and users their requirements using ClassAds. The "Matchmaker" acts as a "yellow page" that finds the appropriate set of resources for a user request. In [41] Kaur et al. propose a centralized resource discovery mechanism for grids which relies on web services. The proposed solution has four main components: UDDI rich Query Model, Grid Web Services Description Language (GWSDL), SOAP and HTTP. The UDDI rich Query Model uses the UDDI standard to discover grid services by maintaining resource information as key-value pair in the UDDI database. GWSDL is an extended version of WSDL that is used to describe grid services. SOAP is used for communication between web services in the grid. HTTP provides an easy to use interface to post and get requests. In [51], Morris et al. present a web services based grid middleware, UNICORE 6, that uses web services to describe grid resources. Each resource is defined by Web Services Resource Framework (WS-RF), an open standard, and stored in a central database (UNICORE User Database). UNICORE provides web services for job submission, access to storage resources and file transfers. Jobs are described using the job submission description language (JSDL).

Hierarchical resource discovery solutions use a hierarchy of servers to discover resources. Nimrod-G [6] follows a hierarchical resource discovery approach. In [32] Ramos et al. propose a solution for resource discovery in grids based on Globus Toolkit (GT3) using web services. The authors propose a hierarchical topology that divides the grid into Virtual Organizations (VO). Each VO has master and slave nodes. The master nodes are responsible for updating the resource database and the slave nodes for retrieving information from resources under their master command. Resource discovery is performed using a configuration file which includes the requested resource information. The configuration is used to generate an XML file that is distributed to all the slaves and the resource search is initiated. Each slave checks if the request is satisfied and returns the information to its master.

Decentralized or P2P resource discovery aims to overcome some limitations of centralized and hierarchical solutions such as scalability and fault tolerance. In [11] the authors propose a P2P three layer resource discovery model for grids. The model is built on a structured P2P system and uses a Distributed Hash Table (DHT) to map nodes and data objects to overlay network. When a client makes a request, it will be carried out to find the resource on the local grid.

If the resource is found, the result will be returned to the user. If the resource is not found, the request will be send to another grid (node) using the P2P virtual layer between nodes. The first layer in the model is formed by the IS root node. The root node is just a node that uses the same resource requests as the other high performance nodes. The second layer is formed by super nodes. Each administrative domain needs to have its own super node, which is registered in the IS root node. The super node provides resource information about its domain, accepts tasks from the upper layer. The super node sends the its results to the root node. Authors propose Chord [55] or Gnutella [1] to manage the second layer. The third layer is composed of the various domains resources This layer can be managed using Chord or Gnutella. In [44] Ma et al. propose a resource discovery model with three layers. The bottom layer is the resource layer. Each resource of a virtual organization (VO), super peer, must register in its super peer. The intermediate layer is formed by super peers. Each super peer saves information from its peers (resources) and exchange that information with other super peers. The upper layer is formed by Super peer Agents. Super peer Agents are resource services of a region and can take charge of one or more similar super peers. The resource discovery uses a DHT and an Ant colony optimization (ACO) algorithm. When a specific resource can not be found in a VO, located via Chord [55], managed by a super peer the solution uses ACO to find the peer which has the required resource.

Table 2 provides a resume and classification of resource discovery in grids. A higher number of "+" means that an approach is more successful on a particular aspect than the others. The reliability aspect is measured in terms of single point of failure. The dynamism aspect is related to resources dynamically joining and leaving the grid.

	Centralized	Hierarchical	P2P
Scalability	+	++	+++
Dynamism	+	++	+++
Reliability	+	++	+++
Server Bottleneck	+++	++	+
Example	Condor-G [30]	Ramos et al. [32]	Chen et al. [11]

Table 2. Resource Discovery classification

3.3 Grid Job Scheduling

In this subsection will be described the following important issues about Grid scheduling: objective functions, phases of Grid scheduling and classification of Grid scheduling algorithms.

3.3.1 Phases of Grid Scheduling The Grid scheduling process can be divided into three main phases [53]: resource discovery where a list of potential

resources is created, system or resource selection were a set of resources is chosen and task execution where the tasks are executed and monitored. Figure 1 was taken from [53] and shows the three main phases and the steps that make them.

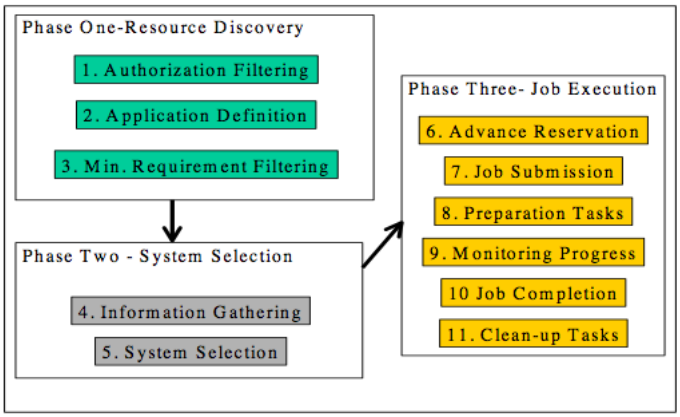

Fig. 1. Grid scheduling phases

- **Resource Discovery:** the first stage is to know which resources are available. The first step consists in determining the set of resources that a user has access to. The is done consulting the Grid Information System (GIS). At the end of this step, the user will have a list of resources that he/she can access. The next step is the application requirement definition. In this step the user specifies a set of requirements for the job in order to filter the set of resources. An example of how requirements can be specified is the ClassAd used by Condor [57]. The next step is to do a minimal requirement filtering. The goal is to eliminate the resources that do not meet the minimal requirements.
- **System Selection:** in this phase the goal is to select a single resource to schedule the job. This is done is in two steps: gather dynamic information and system selection. Gather dynamic information is important in order to make the best mapping between job and resource.
  Information can be obtained by consulting the GIS and the local resource scheduler. The system selection consists in choosing a resource with the gathered information. One of the approaches to resource selection is Condor Matchmaking [50, 57].
- **Task Execution:** the first step, advanced reservation, is optional. The goal is to make the best use of the system. Advanced reservation difficulty depends on the considered resource. When the resource or resources are chosen, the task needs to be submitted. Globus Grid Resource Allocation and Management (GRAM) is used by middleware systems such as Condor-G for job

submission. The next step is preparation. In this step a set of operations take place to prepare the resource to run the task. The following step is monitoring. Once the task is started it is important to keep track of its progress. By monitoring tasks, the scheduler can conclude that a given task in not making progress and may reschedule it. The next step is job completion were the user is notified when a task or job finishes. The final step is the cleanup were temporary files are removed and the user collects information from the resource that will be used to analyze the results.

3.3.2 Classes of Scheduling Algorithms The are many scheduling algorithms. In order to compare them and classify them, a classification needs to be made. In [8], Casavant et al propose a hierarchical taxonomy for scheduling algorithms in general-purpose parallel and distributed systems [20]. Grid falls into a subset of this taxonomy since it is a special kind of the systems that are considered in [8]. Due to the nature of the Grid, some new characteristics such as batch, immediate, adaptive and preemptive scheduling need to be considered in Grid scheduling algorithms. In the following, there are described the main types of scheduling in Grids.

- **Local vs Global**: at the highest level, scheduling can be divided into local and global [8]. Local scheduling operates on a single processor scenario. The scheduler is responsible for the allocation and execution of processes in the CPU [20]. Global scheduling allocate processes to multiple processors to optimize a system-wide performance goal [20]. Considering what was said before it is obvious to conclude that Grid scheduling is global.

- **Static vs Dynamic**: In [60] and [61], Xhafa et all state that exist two main aspects to determine the dynamics of Grid scheduling: dynamics of job execution and dynamic of resources. Dynamics of job execution refers to the situation of job failure. Dynamics of resources refer to the possibility of resources join and leave the Grid and changes of local resource usage policies. In static scheduling the information about the Grid's resources is available at schedule time, every task is assigned once to a resource and there are not job or tasks failures. With static scheduling it is possible to estimate computation costs before the task execution and to have a global view of costs and tasks [20]. These estimations cannot be done in scenarios with nodes can fail or become isolated. Since these situations can occur very often mechanisms such as rescheduling [15] were introduced to smooth the problem.
  In dynamic scheduling cost estimation is difficult [20], jobs can fail and resources can join and leave the Grid in an unpredictable way [60]. Dynamic scheduling has two components: system state estimation and decision making. System state estimation is responsible for collect information about the Grid and build an estimate. This estimate will be the base to the decision of mapping a task to a resource. Since it is not possible to estimate computation costs before execution load balancing is used as an alternative to ensure

the system well functioning.

- **Centralized vs Decentralized vs Hierarchical**: The scheduling responsibility can be delegated on one centralized scheduler or be shared by multiple distributed schedulers. On the centralized approach there is only one scheduler for the Grid. In the centralized approach it is possible to monitor all the resources state which makes easier to create efficient schedulers [60]. Another advantage of centralized scheduling is the easy management [42] and implementation of schedulers. However, centralized scheduling approaches have a single point of failure [61], lack of scalability [20, 42, 60, 61] and lack of fault-tolerance [20,42,60]. Condor [57,61] uses a centralized scheduler based on the ClassAd matchmaker [50]. On the decentralized approach there is no central scheduler that controls the resources. In this approach, local schedulers play an important role since the scheduling requests are sent to them. These type of schedulers take in consideration important issues such as fault-tolerance, scalability and multi-policy scheduling. In the hierarchical approach schedulers are organized in an hierarchical way. This approach is more scalable and fault-tolerant than the centralized approach although, it does not scale and it is not fault-tolerant as the decentralized approach.

- **Immediate vs batch**: In the immediate approach, jobs are schedule as they enter the system [61] using the system's scheduling algorithm. Jobs do not wait for the next time interval when the scheduler will get activated [61] On the other way, in the batch approach, jobs are grouped in batches and scheduled as a group [61]. In the batch approach the scheduler can use job and resource characteristics better than immediate schedulers since it has the time between the activation of the batch scheduler.

- **Adaptive**: This approach uses information regarding the current status of the resources and predictions of their future status to avoid a decrease of performance. Rescheduling is an adaptive scheduling where running jobs are migrated to other resources. In [47], Othman et al refer that the Grid must be able to recognize the state of resources and propose an adaptable resource broker. An example of an adaptive scheduling algorithm can be found on Huedo et al. work [33].

3.3.3 Classic Scheduling Algorithms In this section we present some of the classical scheduling algorithms in Grids and distributed systems.

First Come First Served In First Come First Served algorithm, jobs are executed according to the arriving time order [43]. This algorithm has a major disadvantage. When a large job is on the waiting queue, the jobs behind it must wait a long time for the large job to finish. This situation is called convoy effect.

Round Robin In the Round Robin algorithm each job is assigned a time interval, called quantum, during which it is allowed to run [56]. If a job cannot be completed in a quantum it will return to the queue and wait for the next

Design Choice	Approaches
Dynamics	Dynamic Static
Architecture	Centralized Hierarchical Decentralized
Mode	Immediate Batch

Table 3. Classes of scheduling algorithms

round [43]. Round Robin has the advantage that a job does not need to wait for the previous job to complete to execute. The only challenging issue with this algorithm is to find a suitable length for the quantum [56].

Minimum Execution Time The Minimum Execution Time (MET) algorithm assigns each task to the resource that performs it with the minimum execution time [45]. MET does not consider whether the resource is available or not at the time (ready time) [21, 45, 52] and can cause severe imbalance in load across resources [21, 45, 52]. The main advantage of the algorithm is that it gives to a task the resource that performs it in the smallest amount of time [45]. MET takes $O(m)$ time to map a task to a resource [21].

Minimum Completion Time The Minimum Completion Time (MCT) algorithm assigns a task to the resource that obtains the earliest completion time for that task [21, 45, 52]. The expression for the completion time can be found in appendix A, equation 3. MCT has the following disadvantage: the resource that was assigned to a task may not have the minimum execution time for it [21, 45, 52]. MCT takes $O(m)$ time to map a task to a resource [21].

Min-min The Min-min algorithm has two phases [21]. On the first phase, the completion time of all unassigned tasks on all available machines is used to calculate to minimum completion time of a task T on a machine M [52]. On the second phase, the task with the minimum completion time is chosen, removed from the task list and assigned to the corresponding resource [21]. The process is repeated until all tasks are mapped to a resource. We can conclude that jobs the can be completed earliest have higher priority than the others [37, 43, 52]. Min-min takes $O(s^2m)$ time to map a task to a resource [21].

Min-max The Min-Max algorithms has two phases [37, 52] and uses the minimum completion time (MCT) for the first phase and the minimum execution time (MET) for the second phase as metrics. The first phase of Min-Max is the same as the Min-min algorithm. The second phase the task whose $\frac{MET_{\{fastest\,machine\}}}{MET\{selected\,machine\}}$ has the maximum value will be selected for mapping [37]. The task is removed from the unassigned list, resource workload is updated and the process is repeated until the list is empty [52]. The intuition of this algorithm is that we select resources and tasks from the first step that the resource can execute the task with a lower execution time in comparison with other resources [37].

Max-min The Max-min has two phases as Min-min has [21]. The first phase is equal to the Min-min algorithm [21,37,52]. On the second phase, the task with the maximum completion time is chosen, remove from the task list and assigned to the corresponding resource [52]. The process is repeated until all tasks are mapped to a resource. Max-min can be combined with Min-min in scenarios where the are tasks of different lengths [43]. Max-min takes $O(s^2m)$ time to map a task to a resource [21].

Sufferage The sufferage of a task is the difference between its second minimum completion time and its first minimum completion time [37, 52]. These completion times are calculated considering different resources [43]. In the Sufferage algorithm the criteria to assign a task to a resource is the following: assign a resource to a task that would suffer the most if that resource was not assigned to it [43, 45]. The sufferage value of a task is the difference between its second earliest completion time and its earliest completion time [43, 45]. Once a task is assigned to a resource it is removed from the list of unassigned tasks and the process is repeated until there are no tasks in the unassigned list. Sufferage takes $O(s^2m)$ time to map a task to a resource [21].

Largest Job on Fastest Resource - Shortest Job on Fastest Resource The Largest Job on Fastest Resource - Shortest Job on Fastest Resource (LJFR-SJFR) algorithm allocates the largest job on the fastest resource in order to reduce makespan (equation 1) and the smallest job to fastest resource in order to reduce the flow time, appendix A equation 2, [37, 52]. On the first phase of the algorithm is the same as the Max-min algorithm with one difference, LJFR-SJFR does not consider all the jobs (N) on this phase. Let $0 < m < N$ be the number of considered jobs on the first phase. At the end of the first phase, m jobs are assigned to m machines. On the second phase, the remaining jobs are assigned using Min-min and Max-min methods alternatively i.e. SJFR followed by LJFR [37, 52].

3.3.4 QoS and Utility-based Scheduling Algorithms

QoS are constrains or bounds that are related to the provided service. QoS appeared on aspects related to telephony and computer networks such as service response time, loss, signal-to-noise ratio, cross-talk, echo, etc. In the Grid environment there are some different QoS aspects to consider such as deadline, price, execution time, overhead.

Utility is a concept, originally from economics, that evaluates the satisfaction of a consumer while using a service. In a Grid environment, utility can be combined with QoS constrains in order to have a quantitative evaluation of a user's satisfaction and system performance.

The classical scheduling algorithms presented in the previous section do not consider QoS or utility demands. Next, we present some QoS and utility scheduling algorithms for Grid environments.

In [3] Amuda et al. propose a QoS priority-based scheduling algorithm. The algorithm assumes that all the necessary information about resources, jobs and priority values to be available and is designed for batch mode independent tasks.

The task partition divides the tasks into two groups (high and low) using the priority value as a QoS parameter. After the task division, the scheduler classifies the tasks into four categories: 1A - low complexity and high priority, 1B - low complexity and high priority, 2A - high complexity and low priority and 2B - low complexity and low priority. After task classification the resources are divided into two groups: high processing speed systems (group 1) and hybrid systems (group 2). High processing tasks go to group 1 and the others to group 2. Tasks with higher priority are scheduled first and equally on all the machines.

In [9] Chauhan et al. propose two algorithms for QoS-based task scheduling: QoS Guided Weighted Mean Time-min (QGWMT) and QoS Guided Weighted Mean Time Min-Min Max-Min Selective (QGWMTMMS).

QGWMT is a modification of the Weighted Mean Time-min [40] algorithm that considers network bandwidth as a QoS parameter. First, the algorithm divides the tasks in two groups: high and low QoS. Tasks from the high QoS group are scheduled first. For each group, the algorithm calculates the performance of resources. Next, for each task on a group, the algorithm calculates the task's weighted mean time (WMT). The task with the higher WMT is selected. Then, the algorithm choses the resource that gives the earliest completion time to the selected task and maps the task to it. This process is repeated, for each group, until all tasks are mapped.

QGWMTMMS is a modification of the Weighted Mean Time Min-Min Max-Min Selective [10] algorithm using network bandwidth as a QoS parameter. First, the algorithm creates n priority groups and assign tasks to them according based on their QoS demands. Tasks from higher priority groups are scheduled first. First, the algorithm calculates the expected execution time of each task on all resources. For each group, it calculates the weight of the resources in that group, the weighted mean time for each task in the group and the standard deviation of the completion time of unassigned tasks. If the relative standard deviation is less than the critical value of relative standard deviation (calculated by experiments) the task with the minimum WMT is selected for mapping. Otherwise, the task with the higher WMT is selected for mapping.

In [12] Chen proposes an economic grid resource scheduling based on utility optimization that uses a universal flexible utility function that addresses QoS requirements of deadline and budget. The paper assumes that a grid is hierarchical and that the user submits the assignment to a Grid Resource Manager (GRM). The GRM is at the top of the hierarchy, on the second level there are the Domain Resource Managers (DRM) that are responsible for Computing Nodes (CN) or other DRM. The algorithm starts by the GRM getting utility information from all DRM and by calculating the rate of throughput and average response delay. Then, the algorithm finds out which of DRM has the maximum utility value (MUV) and selects it to be the scheduling node. If MUV is not unique, them the DRM which has the greatest variance is chosen. If the nodes on the next level of the chosen DRM are not CN them the process is repeated. Otherwise, the algorithm finds the node which has the maximum utility value and give it the user assignment.

In [14] Chunlin et al. propose an optimization approach for decentralized QoS-based scheduling based on utility and pricing. The authors consider two types of agents in the proposed scheduling model: Grid resource agents that represent the economic interests of the resources and Grid task agents that represent the interests of the Grid user. Grid resources can be divided into computational resources (CPU speed, memory size, storage capacity) and network resources (bandwidth, loss rate, delay and jitter). Task agents specify their resource requirements using a simple and declarative utility model. The Grid is seen as a market where the task agents act as consumers that and resources as providers that compete with each other to maximize their profit. Due to the fact that the its not realistic that the Grid knows all the utility functions of the task agents, although it is mathematical tractable, and it requires global coordination of all users, the authors propose a decomposition of the problem in two problems (task agent optimization and resource agent optimization) by adopting a computational economy framework. The proposed solution allows multi-dimensional QoS requirements that can be formulated as a utility function that is weighted sum of each dimension's QoS utility function. Three QoS dimensions are considered: payment, deadline and reliability. The scheduling is done by solving the subproblems via an iterative algorithm. In each iteration, each player (task agent and resource agent) trade with each other to find a global optimum solution to the system while trying to maximize their own utility. The process stops when all arrive at the same solution.

Table 3.3.4 presents a classification of all the scheduling algorithms described in this document using some criteria describe in section 3.3.2. The symbol * in the table means that there was not possible to evaluate the corresponding criteria of a particular algorithm using the information provided in the author's paper.

Algorithms	Order-Based	Heuristic	QoS	Utility	Adaptative	Dynamics	Mode	Complexity
FCFS	Yes	No	No	No	No	Static	Batch	*
Round Robin	Yes	No	No	No	No	Static	Batch	*
MET	No	Yes	No	No	No	Static	Immediate	$O(m)$
MCT	No	Yes	No	No	No	Static	Immediate	$O(m)$
Min-Min	No	Yes	No	No	No	Static	Batch	$O(s^2m)$
Min-Max	No	Yes	No	No	No	Static	Batch	$O(s^2m)$
Max-Min	No	Yes	No	No	No	Static	Batch	$O(s^2m)$
Sufferage	No	Yes	No	No	No	Static	Batch	$O(s^2m)$
LJFR-SJFR	No	Yes	No	No	No	Static	Batch	$O(s^2m)$
Amuda et al.	No	Yes	Yes	No	No	Static	Batch	*
Chaughan et al. 1	No	Yes	Yes	No	No	Static	Batch	$O(s^2 + sm)$
Chaughan et al. 2	No	Yes	Yes	No	No	Static	Batch	$O(s^2 + sm)$
Chen	No	Yes	Yes	Yes	No	Static	*	*
Chunlin et al.	No	Yes	Yes	Yes	No	Static	*	*

Table 4. Classification of scheduling algorithms

3.4 Resume and Discussion

In this related work section we presented different topics related to grid scheduling. First, we described and classified some of the existing grid middleware systems. All the described systems have a centralized or hierarchical architecture. With the continuous growth of grids these architectures will face problems such as scalability, fault tolerance and server bottleneck. Next we made a classification and described some resource discovery approaches. Due to dynamics of the grid we concluded that the P2P approach was most adequate to grid environments. After resource discovery we presented grid scheduling. We started the section by presenting the stages of grid scheduling. Then we presented some important criteria for classification of scheduling algorithms. Next, we described and classified two different types of scheduling algorithms: no QoS or utility constrains and with QoS and/or utility constrains. Considering the algorithms with QoS and utility constrains we concluded that all solutions consider only complete requirement fulfillment when calculating utility.

In our solution we will consider scalability and bottleneck issues by proposing a P2P topology to the grid. Our utility scheduling algorithm will consider partial requirement fulfillment, i.e. partial utility.

4 Proposed Solution

A high level representation of the proposed solution's architecture is depicted in Figure 2. In our solution we organize the grid as structured P2P network where the peers are Virtual Organizations (VO). A study performed by Zhang et al. [62] shows that some centralized solutions such as Globus MDS fail to scale beyond 300 concurrent users i.e. the throughput begins to decline below acceptable levels. We present a decentralized architecture since it has more advantages when compared to centralized or hierarchical approaches.The pros and cons of each approach were discussed in the Related Work section. In each peer there is a Local Scheduler (LC) that is responsible for all scheduling operations of a the considered VO. Each LC will implement our utility-based scheduling algorithm. In 4.1 we describe in detail each of the architecture's entities and the interactions between them. In 4.2 we describe our utility-based scheduling algorithm. In 4.3 we present the technologies that will support our work.

4.1 Architecture in Detail

In this section we will present the network topology of the grid and what information is exchanged between the entities that we present on Figure 2.

Resources send information about their current state to the LC. Resources' information can be classified in two categories: static and dynamic. Static information does not change over time. Some examples of static information are: operating system, processor, number of cores, disk memory, RAM, etc. Dynamic information is likely to change over time. Some examples of static information

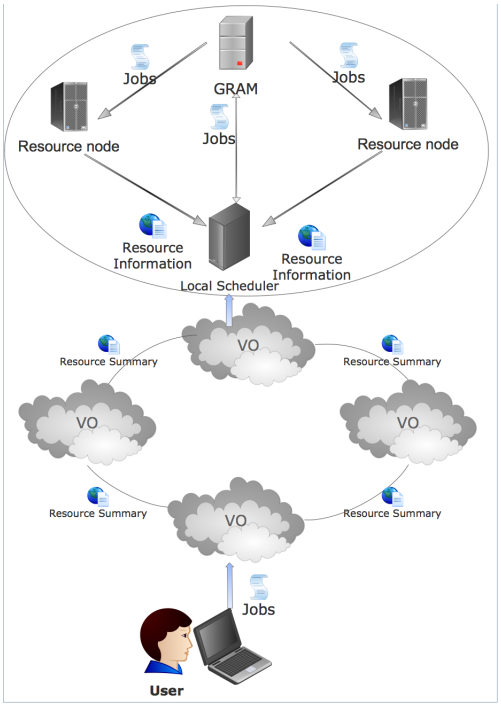

Fig. 2. High Level Architecture

are: CPU occupation (per core), number of allocated jobs, free disk memory, free RAM, etc. The information is, periodically, sent to the resources using XML format. VOs exchange information about their resource state periodically. The exchanged information is a resume of the VO's resources state that includes average resource utilization, resource with the less CPU occupation, resource that more available memory (disk and RAM).

The grid is organized in a structured P2P network. We use Chord [55] for routing the exchanged messages between VOs. Chord is based on a Distributed Hash Table (DHT). Each node in the network has a unique identifier (key) and can be located using a consistent Hash function. According to the order of the node identifiers, Chord forms a logical ring topology and simply routes a key through a sequence of other nodes toward the destination node. Chord's routing complexity in a P2P network with n nodes is $O(log\ n)$. By using this routing approach each VO only needs to have the identifiers of other VOs.

4.2 Utility Scheduling Algorithm

As mentioned before, VOs exchange summaries of resource information. When a LC receives a job, it calculates the utility value of each of its resources. Next, the LC uses the information from other VOs to calculate an estimate of that VO's

utility. If any of those estimates is greater than the maximum value of utility of local resources, then the job is forwarded to that specific VO. Otherwise, the job is assigned to the resource that has the higher utility value. The pseudo-code for the described procedure is presented in Algorithm 1

Algorithm 1 pseudo-code to be executed by the Local Scheduler

```
localUtil = []
req ← job.Requirements
resumes ← VOsResumes
for all r ∈LocalResources do
  localUtil.add(CalculateUtility(r,req))
end for
bestLocalUtil = max(localUtil)
for all res ∈resumes do
  u = CalculateUtility(res,req)
  if u > bestLocalUtil then
    ForwardJob(job,res.getVO)
    exit
  end if
end for
AssignJob(bestLocalUtil.getResource)
exit
```

Next, we will describe how to calculate the utility value of a resource used in Algorithm 1 in *CalculateUtility(resource,requirement)*. Our solution is based on the work of Silva et al. [54]. In our solution, users describe job requirements using XML. For each requirement, it will be possible for the user to define **intervals with different values of utility**. Using this feature will lead to a more flexible evaluation of resources using partial utility. We define two basic operators: *and*, *or*. The *or* operator selects the resource having the highest utility. The *and* operator indicates that all requirements must be met and the result is a combined aggregation of the combined satisfaction of requirements.

In our solution we consider **four classes of users** : guest, registered in other VO, registered in the VO, VO administrator. Each class defines a way to calculate utility for operators *and* and *not* .

Guest users have the most flexible policy. The *and* operator will return the weighted sum of all partial utilities of a resource. This policy follows a best-effort approach. Users that are **registered in other VO** have more privileges than guests so their policy tries to satisfy their requests in a more favorable and balanced maner. The operator *and* will return the product of all partial utility values of a resource. Users that are **registered in the VO** have a policy that aims to minimize their dissatisfaction. The *and* operator will return the minimum partial utility value of a resource. **Administrators** have the most rigid policy of all classes. Operator *and* is used when the user is not willing to accept resources that do not fulfill one or more requirements. The *or* operator,

unlike all other classes that gives the requirement with the maximum utility, will test the requirements in the order they appear on the job requirement file.

4.3 Implementation Issues

There will be two phases of implementation. On the first phase we will implement our solution using a grid simulator. The simulator will allow us to have an idea of how the solution will behave in a scenario with many VOs with different kinds of resources. The chosen simulator was GridSim because it is widely used by many authors and has some relevant functions such as the possibility of organize entities in a network topology.

The second phase we will implement our algorithm on a grid. We will use the Web Services Resource Framework (WSRF) of Globus Toolkit. WSRF allows access to Globus components such as GRAM or file transfer (Reliable File Transfer Service) using a web services interface. Resources will use web services to send information about their status to the LC. After the LC decided the resource to run a specific job it will ask GRAM to submit the job to that resource.

5 Evaluation Methodology

In this section we present the metrics and the test environment that we will use to evaluate our solution. We will use the following metrics to evaluate the performance of our algorithm.

- Number of requests successfully allocated using partial utility compared with the discarded requests on conventional solutions;
- Average utility of the system;
- Overall resource utilization;
- Job makespan;
- Number of idle resources;
- Total weighted completion time;
- Comparison of the average satisfaction of each class of users

Our tests will be run on the GridSim simulator to evaluate the scalability and behavior of the solution for large number of VOs and users. GridSim will be the tool to make a comparison between centralized schedulers approach and our decentralized solution. We will also implement our solution in a grid environment to assert how the algorithm behaves in a non-simulation scenario.

More information on makespan, resource utilization and total weighted completion time can be found on appendix A.

6 Conclusions

We started this document by presenting a global view about scheduling in grids. We described and classified some grid middleware solutions, resource discovery

approaches and scheduling algorithms. The analysis of all theses topics allowed us to have the necessary knowledge to identify some aspects that have not been explored: decentralized grid scheduling algorithm and partial requirement fulfillment.

Once identified the shortcomings we proposed a solution that considers the grid as a structured P2P network where the peers are VO and where local schedulers of each VO interact exchanging information about the state of their resources and use a method to calculate utility that considers partial requirement fulfillment. Finally, we presented the metrics the will support the evaluation of our solution.

References

1. Gnutella website. http://www.gnutella.com/, January 2007.
2. David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18:1061–1074, 2002.
3. T. Amudha and T.T. Dhivyaprabha. Qos priority based scheduling algorithm and proposed framework for task scheduling in a grid environment. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 650 –655, June 2011.
4. F. Berman. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15(4):327–344, November 2001.
5. Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. Gass: a data movement and access service for wide area computing systems. In *Proceedings of the sixth workshop on I/O in parallel and distributed systems*, IOPADS '99, pages 78–88, New York, NY, USA, 1999. ACM.
6. R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: an Architecture for a Resource Management and Scheduling System in a Global Computational Grid. *Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, pages 283–289 vol.1, 2000.
7. Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing 2 . Players in the Grid Marketplace. *Marketplace*, pages 1–27.
8. T.L. Casavant and J.G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *Software Engineering, IEEE Transactions on*, 14(2):141 –154, feb 1988.
9. Sameer Singh Chauhan and R C Joshi. Qos guided heuristic algorithms for grid task scheduling. *International Journal of Computer Applications*, 2(9):24–31, 2010.
10. S.S. Chauhan and R.C. Joshi. A weighted mean time min-min max-min selective scheduling strategy for independent tasks on grid. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 4 –9, feb. 2010.
11. Dong Chen, Guiran Chang, Xiuying Zheng, Dawei Sun, Jiajia Li, and Xingwei Wang. A novel p2p based grid resource discovery model. *Journal of Networks*, 6(10), 2011.
12. Juan Chen. Economic grid resource scheduling based on utility optimization. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 522 –525, April 2010.

13. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A framework for constructing scalable replica location services. In *Supercomputing, ACM/IEEE 2002 Conference*, page 58, Nov. 2002.
14. L Chunlin and L Layuan. An optimization approach for decentralized qos-based scheduling based on utility and pricing in grid computing. *Concurrency Computation Practice And Experience*, 19(1):107–128, 2007.
15. K Cooper, A Dasgupta, K Kennedy, C Koelbel, A Mandal, G Marin, M Mazina, F Berman, H Casanova, A Chien, H Dail, X Liu, A Olugbile, O Sievert, H Xia, L Johnsson, B Liu, M Patel, D Reed, W Deng, and C Mendes. New Grid Scheduling and Rescheduling Methods in the GrADS Project. *International Journal of Parallel Programming*, 33:209–229, 2005.
16. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 181 –194, 2001.
17. Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82. Springer Berlin - Heidelberg, 1998.
18. Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In Marios Dikaiakos, editor, *Grid Computing*, volume 3165 of *Lecture Notes in Computer Science*, pages 131–140. Springer Berlin - Heidelberg, 2004.
19. Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13:219–237, July 2005.
20. Fangpeng Dong and Selim G Akl. Scheduling Algorithms for Grid Computing : State of the Art and Open Problems. *Components*, pages 1–55, 2006.
21. K. Etminani and M. Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1 –7, sept. 2007.
22. T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with agwl: an abstract grid workflow language. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 676 – 685 Vol. 2, may 2005.
23. Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto, and Hong-Linh Truong. Askalon: a tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience*, 17(2-4):143–169, 2005.
24. Thomas Fahringer, Radu Prodan, Rubing Duan, Juurgen Hofer, Farrukh Nadeem, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wieczorek. Askalon: A development and grid computing environment for scientific workflows. In Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 450–471. Springer London, 2007.
25. Thomas Fahringer and Clovis Seragiotto. Automatic search for performance problems in parallel and distributed programs by using multi-experiment analysis. In

Sartaj Sahni, Viktor Prasanna, and Uday Shukla, editors, *High Performance Computing - HiPC 2002*, Lecture Notes in Computer Science, pages 151–162. Springer Berlin - Heidelberg, 2002.

26. Pavel Fibich and Hana Rudov. Model of Grid Scheduling Problem. *Centrum*, 2001.
27. I. Foster, J. Vockler, M. Wilde, and Yong Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 37 – 46, 2002.
28. Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.
29. Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, CCS '98, pages 83–92, New York, NY, USA, 1998. ACM.
30. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing*, pages 55–63.
31. "Globus". http://www.globus.org/.
32. T. Gomes Ramos and A.C. Magalhaes Alves de Melo. An extensible resource discovery mechanism for grid computing environments. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 115 – 122, May 2006.
33. E. Huedo, R.S. Montero, and I.M. Llorente. Experiences on adaptive grid scheduling of parameter sweep applications. In *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, pages 28 – 33, Feb. 2004.
34. Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A peer-to-peer approach to resource discovery in grid environments. In *In High Performance Distributed Computing*. IEEE, 2002.
35. E. Imamagic, B. Radic, and D. Dobrenic. An Approach to Grid Scheduling by using Condor-G Matchmaking Mechanism. *28th International Conference on Information Technology Interfaces, 2006.*, (3):625–632, 2006.
36. Emir Imamagic, Branimir Radic, and Dobrisa Dobrenic. An Approach to Grid Scheduling by Using Condor-G Matchmaking. *Journal of Computing and Information Technology*, pages 329–336, 2006.
37. H. Izakian, A. Abraham, and V. Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 8 –12, april 2009.
38. Japhynth Jacob, Elijah Blessing Rajsingh, and Isaac Balasingh Jesudasan. Dynamic multi dimensional matchmaking model for resource allocation in grid environment. In Dhinaharan Nagamalai, Eric Renault, and Murugan Dhanuskodi, editors, *Trends in Computer Science, Engineering and Information Technology*, volume 204 of *Communications in Computer and Information Science*, pages 179–185. Springer Berlin Heidelberg, 2011.
39. Yi Jian and Yanbing Liu. The State of the Art in Grid Scheduling Systems Chongqing University of Posts and Chongqing University of Posts and School of Computer Science , UEST of China. *System*, (Icnc), 2007.

40. Zhang Jinquan, N. Lina, and Jiang Changjun. A heuristic scheduling strategy for independent tasks on grid. In *High-Performance Computing in Asia-Pacific Region, 2005. Proceedings. Eighth International Conference on*, pages 6 pp. –593, July 2005.
41. Eep Kaur and Jyotsna Sengupta. Resource discovery in web-services based grids.
42. Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, February 2002.
43. Yun-Han Lee, Seiven Leu, and Ruay-Shiung Chang. Improving job scheduling algorithms in a grid environment. *Future Generation Computer Systems*, 27(8):991–998, October 2011.
44. Shaohui Ma, Xinling Sun, and Zuhua Guo. A resource discovery mechanism integrating p2p and grid. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 7, pages 336 –339, July 2010.
45. M. Maheswaran, S. Ali, H.J. Siegal, D. Hensgen, and R.F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pages 30 –44, 1999.
46. A. Mandal, A. Dasgupta, K. Kennedy, M. Mazina, C. Koelbel, G. Marin, K. Cooper, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling workflow applications in grads. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 790 – 797, april 2004.
47. A. Othman, P. Dew, K. Djemame, and I. Gourlay. Adaptive grid resource brokering. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pages 172 –179, sept. 2003.
48. Radu Prodan and Thomas Fahringer. Zenturio: a grid middleware-based tool for experiment management of parallel and distributed applications. *Journal of Parallel and Distributed Computing*, 64(6):693 – 707, 2004.
49. Mustafizur Rahman, Rajiv Ranjan, Rajkumar Buyya, and Boualem Benatallah. A taxonomy and survey on autonomic management of applications in grid computing environments. *Library*, (May):1990–2019, 2011.
50. R. Raman, M. Livny, and M. Solomon. Resource management through multilateral matchmaking. *Proceedings the Ninth International Symposium on High-Performance Distributed Computing*, pages 290–291.
51. M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M. Shahbaz Memon, A. Shiraz Memon, B. Demuth, T. Lippert, D. Snelling, S. van den Berghe, V. Li, M. Drescher, A. Geiger, G. Ohme, A. Vanni, C. Cacciari, S. Lanzarini, P. Malfetti, K. Benedyczak, P. Bala, R. Ratering, and A. Lukichev. Web services interfaces and open standards integration into the european unicore 6 grid middleware. In *EDOC Conference Workshop, 2007. EDOC '07. Eleventh International IEEE*, pages 57 –60, oct. 2007.
52. Rajendra Sahu. Many-Objective Comparison of Twelve Grid Scheduling Heuristics. *International Journal*, 13(6):9–17, 2011.
53. Jennifer M Schopf. *Chapter 1 Ten Actions when Grid Scheduling The User as a Grid Scheduler*.
54. J.N. Silva, P. Ferreira, and L. Veiga. Service and resource discovery in cycle-sharing environments with a utility algebra. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1 –11, april 2010.
55. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31:149–160, August 2001.

56. Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.
57. Douglas Thain, Todd Tannenbaum, and Miron Livny. *Condor and the Grid*, pages 299–335. John Wiley and Sons, Ltd, 2003.
58. Hong-Linh Truong and Thomas Fahringer. Scalea: A performance analysis tool for distributed and parallel programs. In Burkhard Monien and Rainer Feldmann, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin / Heidelberg, 2002.
59. Sathish S Vadhiyar and Jack J Dongarra. A Metascheduler For The Grid. *Distributed Computing*, 2002, 2002.
60. Fatos Xhafa and Ajith Abraham. Meta-heuristics for Grid Scheduling Problems. pages 1–37, 2008.
61. Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for Grid scheduling problems. *Future Generation Computer Systems*, 26(4):608–621, April 2010.
62. Xuechai Zhang, J.L. Freschl, and J.M. Schopf. A performance study of monitoring and information services for distributed systems. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 270 – 281, June 2003.
63. Yong Zhao, Michael Wilde, Ian Foster, Jens Voeckler, James Dobson, Eric Gilbert, Thomas Jordan, and Elizabeth Quigg. Virtual data grid middleware services for data-intensive science. *Concurrency and Computation: Practice and Experience*, 18(6):595–608, 2006.

A Metrics and Criteria of Grid Scheduling

Makespan can be defined as the finishing time of the last task. It is one of the most popular optimization criteria. Small values of makespan indicate that the scheduler is operating in an efficient way [26]. Considering the makespan as the only critera does not imply the optimization of other objectives.

$$makespan = \max\{F_i\,,\ i = 1, \ldots, N\} \tag{1}$$

Where F_i is the finish time of the i^{ith} task and N is the total number of tasks. The objective function consists in minimizing the maximum value of makespan

Flow Time is the sum of the finishing times of tasks [52]. Flow time measures the response time of the Grid system.

$$flow\ time = \sum F_i\,,\ i = 1, \ldots, N \tag{2}$$

Where F_i is the finish time of the i^{ith} task and N is the total number of tasks.The objective function consists in minimizing the flow time.

Completion Time is a time that a machine will take to finalize the processing of the previous assigned tasks and the planned tasks. This criteria requires knowing, for each machine, the ready time and expected time to complete time the assigned tasks. The following equation calculates the completion time of machine m in the ETC model [60, 61].

$$completion\,time[m] = ready\_times[m] + \sum_{j \in Tasks | schedule[j]=m} ETC[j][m] \quad (3)$$

Where *schedule* is the schedule for machine m and ETC is the Estimated Time to Compute. The objective function consists in minimizing the completion time of all machines.

Resource Utilization is a very important objective. Resource utilization is challenging due to the disparity of computational resources in the Grid. The following expression calculates the average utilization of resources, considering an ETC model [52, 60, 61].

$$average\,resource\,utilization = \frac{\sum_{i \in Machines} completion[i]}{makespan \times nr\_machines} \quad (4)$$

Where *completition* is the completition time of the i^{th} machine and *nr_machines* is the number of machines. The objective function consists in maximizing the resource utilization of the Grid system.

Total Weighted Completion Time is used when user's tasks have priorities. This criterion is implemented associating weights to the tasks. The following expression calculates the total weighted completion time.

$$Total\,weighted\,completion\,time = \sum_{i \in Tasks} w_i F_i \quad (5)$$

Where w_i is the weight of the i^{th} task and F_i is the finishing time of the i^{th} task.

B Planning

Here we present our planning schedule for the implementation of the solution. During the development process we will use ConceptDraw PROJECT for MacOS X, a project management software, to monitor the evolution of our work.

Planning		
Tasks	**Details**	**Duration**
Introduction to GridSim	- Code Study - Tutorials and Examples	January (2 week) Janurary (1 week)
Implementation on GridSim	- Set up network topology - Implement algorithm	February (1 week) February (2 weeks)
Implementation on Grid environment	- Install Globus Toolkit - Set up network topology - Study WSRF and Globus API's - Implement algorithm	March (1 week) March (1 week) March (1 week) March (1 week) April (4 weeks)
Conclusion of Implementation	- Conclude any unimplemented functionality	May (4 weeks)
Performance Measurements	- Evaluate implemented solution	June (4 weeks) July (2 weeks)
Thesis final report writing	- Write thesis report	July (2 weeks) August (4 weeks)
Review and Submission	- Report review and submission	September (2 weeks)
Documentation	- Document design choices, code and tests	January - September
Bi-weekly meetings	- Analyze the progress of the work	January - September

Table 5. Planning schedule